~ *Проект 2, модуль №3* ~

Предисловие

В рамках этого проекта вам предстоит разработать консольное приложение для решения конкретной задачи. Этот проект имитирует реальную рабочую ситуацию, где вам необходимо разработать программное обеспечение в соответствии с техническим заданием.

Цель проекта – применить все знания и навыки, полученные в течении курса, для создания функционального и качественного программного продукта. Вам предстоит самостоятельно спроектировать архитектуру приложения, реализовать необходимую функциональность, протестировать и отладить ваше решение.

У проекта есть базовая часть, а потом 2 ветки развития:

- A-side (темная сторона)

  Эта ветка создана для тех, кто хочет получить проект, который не выходит за рамки пройденного на лекциях. Максимальная оценка при таком выборе ограничивается 8 баллами.

- B-side (светлая сторона)

  Эта ветка для тех, кто хочет поработать с чем-то новым. Ограничения по оценке на данной ветке нету.

Можно выбрать ТОЛЬКО одну ветку. Выбранную ветку **обязательно** укажите в комментариях при сдаче проекта. Все, что будет сделано в рамках ветки, которая не была выбрана не будет оцениваться.

ВАЖНО: Проект должен быть выполнен в строгом соответствии с техническим заданием и общими требованиями к учебным работам. Внимательно изучите все разделы этого документа перед началом работы. Если у вас возникают вопросы, задавайте их в форме Q&A по проекту.

Общие слова

Что делаем?

Проект предполагает самостоятельную разработку ~~консольного~~ приложения в соответствии с техническим заданием (см. раздел «Основная задача» ниже). Вам потребуется:

- Внимательно изучить техническое задание и общие требования к работе.
- Спроектировать архитектуру приложения, включая классы, методы и структуры данных.
- Разработать программный код на языке C#, реализующий всю необходимую функциональность.
- Провести тестирование и отладку приложения для обеспечения его корректной работы.
- Подготовить отчет о выполненной работе, включающий в себя
  - какую ветку вы выбрали
  - формат входных файлов
  - общее описание меню
  - описание структуры проекта
- Сдать в SmartLMS заархивированную папку с решением (Solution) Visual Studio, содержащую проект(ы) с вашим решением и другие необходимые файлы.

Когда сдаем?

Определяется датами, назначенными в SmartLMS.

Что сдаём?

На проверку предоставляется заархивированная папка с решением Visual Studio, в которую включен(ы) проект(ы), через который(е) реализована программа. В названии решения и архива обязательно указать фамилию автора.

Ограничения

В основной и дополнительной задачах **требуется** (да, это блокирующие критерии):

- весь программный код написан на языке программирования C# с учётом использования .net 8.0;
- представленная к проверке библиотека классов решает все поставленные задачи и успешно компилируется.
- в приложении реализован цикл повторения решения, позволяющий повторить работу на других данных без завершения сеанса работы;
- запрещено использовать:
  - NuGet-пакеты, использование которых явно не указано в техническом задании или не согласовано с преподавателем;
  - Прочий публично доступный код / библиотеки, не относящиеся к стандартным библиотекам .NET, без явного указания авторства и согласования с преподавателем;

Общие требования к работе

Ваше приложение должно:

- Соответствовать заданию (см. раздел «Основная задача» ниже).
- Быть работоспособным: Запускаться и выполнять все заявленные в техническом задании функции без критических ошибок.
- Обеспечивать корректный ввод данных в соответствии с требованиями технического задания (если это необходимо). Ввод может быть из консоли, из файла, или генерироваться внутри программы (например, для демонстрации).
- Обеспечивать корректный вывод результатов работы в соответствии с требованиями технического задания (если это необходимо). Вывод может быть в консоль, в файл, или визуализироваться в консоли. Формат вывода должен быть четким и понятным.
- Обрабатывать возможные ошибки ввода данных и исключительные ситуации: Программа не должна «падать» при некорректном вводе пользователя или при возникновении непредвиденных ситуаций. Сообщения об ошибках должны быть информативными и позволять пользователю понять причину проблемы.
- Иметь чистый и хорошо структурированный код, соответствующий принципам объектно-ориентированного программирования (если применимо).
- Содержать подробные комментарии в коде, объясняющие логику работы каждого важного участка кода.
- Иметь понятную структуру проекта (разделение на папки, файлы, классы, методы).
- Быть оформлено в соответствии с общепринятыми стандартами оформления кода на C# (стиль, отступы, именование переменных и т.д.).

Также обратите внимание на примечания в конце файла.

~ *Индивидуальные варианты* ~

~ *Вариант №1* ~

Базовая часть

Разработать консольное приложение «Планировщик путешествий». Приложение должно:

1. Хранить данные о планируемых поездках.

   Каждая поездка должна содержать информацию о:
   - Названии города/страны
   - Дате начала и окончания поездки
   - Списке достопримечательностей для посещения
   - Бюджете поездки

2. Путь к файлу:
   - Путь к файлу с поездками должен запрашиваться у пользователя.
   - Приложение должно проверять корректность пути и обрабатывать ошибки.

3. Предоставлять пользователю возможность:
   - Просмотра всех пунктов маршрута (в виде списка: ID, Дата, Город, Страна, Описание).
   - Добавления нового пункта маршрута. Запрашивать у пользователя дату, город, страну и описание. ID присваивается автоматически.
   - Редактирования пункта маршрута. Пользователь выбирает пункт по ID и может изменить любое поле.
   - Удаления пункта маршрута. Пользователь выбирает пункт по ID.

4. Выводить информацию о поездках в консоль в удобном формате. Главное, чтобы вся информация о поездке была выведена на экран.

5. Сохранять изменения обратно в файл при завершении работы приложения.

[К варианту]

A-side

Основная часть

1. Фильтрация и сортировка поездок:
   - Фильтрация по городу/стране, диапазону дат, бюджету.
   - Сортировка по дате начала, дате окончания, бюджету.
2. Статистика:
   - Вывод общего количества поездок.
   - Вывод количества поездок по каждой стране.
   - Вывод средней стоимости поездки.
3. Поиск поездок:
   - Поиск по названию города/страны, достопримечательностям.

Дополнительная часть

1. Транспорт:
   - Добавить поле «Транспорт» к каждому пункту маршрута (возможные значения: Самолёт, Поезд, Автобус, Автомобиль, Пешком, Другое). Предусмотреть выбор из списка.
   - Реализовать фильтрацию по типу транспорта.
   - Добавить вывод информации по транспорту при выводе на консоль и при экспорте.
2. Проживание:
   - Добавить поле «Проживание» к каждому пункту маршрута (возможные значения: Отель, Хостел, Апартаменты, У друзей/родственников, Другое, Нет). Предусмотреть выбор из списка.
   - Реализовать фильтрацию по типу проживания.
   - Добавить вывод информации по проживанию при выводе на консоль и при экспорте.
3. Заметки:
   - Добавить возможность добавления многострочных заметок к каждому пункту.
   - Добавить вывод информации по заметке при выводе на консоль и при экспорте.

[К варианту]

B-side

Основная часть

1. Визуализация с помощью Spectre.Console:
   - Таблица: Отображение списка поездок в виде интерактивной таблицы с возможностью прокрутки, фильтрации и сортировки.
   - Карта: Отображение на карте мира городов/стран, в которые запланированы поездки.
2. Интеграция с 2GIS:
   - Реализовать возможность получения информации о достопримечательностях из 2GIS.
   - Отображать информацию о достопримечательностях (описание, рейтинги) в консоли.
3. Импорт/Экспорт:

   Реализовать импорт/экспорт данных маршрута в форматы JSON и CSV.

   В одной «ячейке» CSV хранить информацию в формате объекта/массива нельзя. При необходимости сделайте импорт/экспорт нескольких CSV файлов для различных сущностей.
4. Интеграция с API погоды:

   - При добавлении нового пункта маршрута или редактировании существующего, автоматически запрашивать прогноз погоды для указанного города и даты, используя API OpenWeatherMap.
   - Сохранять полученную информацию о погоде (температура, осадки, облачность) в данных пункта маршрута. Предусмотреть обработку ошибок (например, недоступность сервиса погоды).
   - Отображать прогноз погоды для каждого пункта маршрута при просмотре детальной информации.
   - Предусмотреть «ручной» запрос погоды (пользователь нажимает кнопку, и приложение обновляет данные о погоде для выбранного пункта).

Дополнительная часть

1. Автоматическое составление плана поездки: Создать функцию автоматического планирования поездки, которая позволяет пользователю:

   - Выбрать интересующие достопримечательности и даты посещения.
   - Указать желаемое время пребывания в каждом месте.

   Система должна:

   - Автоматически построить стремящийся к оптимальности маршрут с учетом расстояний и времени работы объектов.
   - Если невозможно посетить все места за один день, включить в маршрут возвращение в отель.
   - Предоставить пользователю краткую сводку маршрута: начальная и конечная точки, время в пути, расстояние и время отправления.
2. Генерация PDF-отчета о поездке:

   Реализовать генерацию PDF-отчета по всему маршруту, содержащего информацию о пунктах маршрута, датах, транспорте, проживании, погоде, расстояниях и заметках. Использовать библиотеку для работы с PDF (например, PdfSharp).

[К варианту]

~ *Вариант №2* ~

- Базовая часть
- A-side
- B-side

Базовая часть

Разработать консольное приложение «Менеджер задач». Приложение должно:

1. **Хранить данные о задачах пользователя.**
   - Каждая задача должна содержать информацию о:
     - **Названии задачи**
     - **Описании задачи**
     - **Дате создания задачи**
     - **Сроке выполнения задачи**
     - **Статусе задачи** (например, «Активна», «Выполнена», «Отложена»)
     - **Приоритете задачи** (например, «Высокий», «Средний», «Низкий»)
   - Приложение должно читать задачи из файла при запуске.
   - Путь к файлу задач должен запрашиваться у пользователя.
   - Приложение должно проверять корректность пути и обрабатывать ошибки.
2. **Предоставлять пользователю возможность:**
   - Просмотра списка всех задач (в виде списка: ID, Название, Статус, Приоритет, Deadline).
   - Добавления новой задачи с указанием всей необходимой информации. ID задачи присваивается автоматически.
   - Редактирования существующей задачи. Пользователь выбирает задачу по ID и может изменить любое поле.
   - Изменения статуса задачи (например, пометить задачу как «Выполнена»).
   - Удаления задачи. Пользователь выбирает задачу по ID.
   - Фильтрации задач по статусу и приоритету.
3. **Выводить информацию о задачах в консоль в удобном формате.** Главное, чтобы вся информация о задаче была выведена на экран.
4. **Сохранять изменения обратно в файл при завершении работы приложения.**

[К варианту]

A-side

Основная часть

1. **Расширенная фильтрация и сортировка задач:**
   - Фильтрация по статусу, приоритету, наличию/отсутствию срока выполнения, диапазону дат создания.
   - Сортировка по дате создания, сроку выполнения, приоритету, названию.
   - Возможность комбинировать фильтры и сортировки.
2. **Статистика:**
   - Вывод общего количества задач.
   - Вывод количества задач по каждому статусу (например, сколько «Активных», «Выполненных» и т.д.).
   - Вывод количества задач по каждому приоритету.
   - Вывод количества задач, выполненных в срок и просроченных (если у задачи указан срок выполнения).
3. **Поиск задач:**
   - Поиск задач по названию и описанию (частичное совпадение).

Дополнительная часть

1. **Категории задач:**
   - Добавить поле «Категория» к каждой задаче (например, «Работа», «Личное», «Учеба»). Предусмотреть возможность выбора из списка предустановленных категорий, а также добавления новых категорий пользователем.
   - Реализовать фильтрацию задач по категории.
   - Добавить вывод категории в информацию о задаче при просмотре и экспорте.
2. **Напоминания:**
   - Реализовать возможность установки напоминаний о приближающемся сроке выполнения задачи. Напоминание должно срабатывать за определенное время до срока (настраивается пользователем, например, за час, за день).
   - Реализовать вывод уведомлений о напоминаниях в консоль при запуске приложения.
3. **Экспорт/импорт:**
   - Реализовать экспорт и импорт данных о задачах в/из форматов CSV и JSON.
   - В одной «ячейке» CSV хранить информацию в формате объекта/массива нельзя. Сделайте экспорт нескольких CSV файлов, если необходимо разделить сущности.

[К варианту]

B-side

Основная задача

1. **Визуализация с помощью Spectre.Console:**
   - **Таблица:** Отображение списка задач в виде интерактивной таблицы с возможностью прокрутки, фильтрации и сортировки прямо в таблице. Добавить стилизацию таблицы.
   - **Календарь:** Отображение календаря на месяц, в котором отмечены дни, когда есть задачи со сроком выполнения. При наведении на день показывать количество задач на этот день.
2. **Диаграммы и графики:**
   - Реализовать визуализацию статистики по задачам в виде диаграмм (например, breakdown chart для соотношения статусов задач, столбчатая диаграмма для задач по приоритетам). Использовать Spectre.Console для отрисовки диаграмм.
3. **Интерактивный интерфейс:**
   - Использовать Spectre.Console.AnsiConsole.Prompt для создания более интерактивного меню управления приложением (навигация по задачам, выбор действий и т.д.).

Дополнительная задача

1. **Расширенные напоминания:**
   - Реализовать отправку напоминаний не только в консоль, но и другими способами, например:
     - **Уведомления на рабочем столе** (DesktopNotifications).
     - **Отправка email-уведомлений** (потребуется настройка SMTP-сервера).
     - **Отправка уведомлений в Telegram** (потребуется создание бота и получение токена).
   - Предоставить пользователю возможность выбора способа уведомлений и настройки времени напоминаний.
2. **Анализ продуктивности:**
   - Реализовать функцию анализа продуктивности пользователя на основе данных о задачах. Например:
     - **Анализ выполнения задач по дням недели/месяцам.**
     - **Оценка средней скорости выполнения задач.**
     - **Выявление наиболее часто откладываемых задач.**
   - Визуализировать результаты анализа с помощью графиков и диаграмм в консоли.

~ *Вариант №3* ~

- Базовая часть
- A-side
- B-side

Вариант №3

[К варианту]

Базовая часть

Разработать консольное приложение «Анализатор продаж». Приложение должно:

1. Хранить продажи.
   - Каждая продажа должна содержать информацию о:
     - ID транзакции
     - Дата транзакции
     - ID товара
     - Наименование товара
     - Количество
     - Цена за единицу
     - Регион
   - Приложение должно читать задачи из файла при запуске.
   - Путь к файлу задач должен запрашиваться у пользователя.
   - Приложение должно проверять корректность пути и обрабатывать ошибки.
2. Предоставлять пользователю возможность:
   - Просмотра всех транзакций (в виде списка).
   - Добавления новой транзакции. Запрашивать у пользователя все необходимые данные. ID транзакции генерировать автоматически.
   - Удаления транзакции (по ID транзакции).
   - Редактировать транзакции (по ID).
3. Выводить информацию о поездках в консоль в удобном формате. Главное, чтобы вся информация о продаже была выведена на экран.
4. Сохранять изменения обратно в файл при завершении работы приложения.

—

P.S. ID товара **не** подразумевает, что в программе должен храниться справочник товаров.

[К варианту]

A-side

Основная часть

1. Фильтрация и сортировка транзакций:
   - Фильтрация по дате (диапазон дат).
   - Фильтрация по ID товара.
   - Фильтрация по наименованию товара (частичное совпадение).
   - Фильтрация по региону.
   - Сортировка (по возрастанию/убыванию):
     - По дате.
     - По ID транзакции.
     - По сумме транзакции (Количество x Цена за единицу).
   - Возможность комбинировать фильтры и сортировки.
2. Простая статистика:
   - Общее количество транзакций.
   - Общая сумма продаж.
   - Средняя сумма транзакции.

Дополнительная часть

1. Категории товаров:
   - Добавить поле «Категория товара» к каждой транзакции. Предусмотреть предопределенный список категорий (например, «Электроника», «Бытовая техника», «Одежда», «Продукты») и возможность добавления новых категорий пользователем.
   - Реализовать фильтрацию по категории товара.
   - Добавить вывод категории в информацию о транзакции.
2. Скидки:
   - Добавить поле «Скидка» (в процентах) к каждой транзакции.
   - При расчете суммы транзакции учитывать скидку.
   - Добавить возможность фильтрации транзакций по наличию скидки (есть/нет).
   - Рассчитывать и выводить общую сумму скидок за выбранный период.
3. Расширенная статистика:
   - Общая сумма продаж по каждому товару.
   - Общее количество проданных единиц по каждому товару.
   - Средняя цена товара (с учетом всех транзакций).

B-side

Основная задача

1. Визуализация с помощью Spectre.Console:
   - Таблица: Отображение транзакций в табличном виде с возможностью прокрутки, фильтрации и сортировки прямо в таблице.
   - Гистограмма: Отображение суммы продаж по дням за выбранный период.
   - Breakdown Chart: Отображение распределения продаж по категориям товаров за выбранный период.
2. Аналитика по регионам
   - Выводить информацию по регионам в табличном виде.
   - Возможность сортировки по регионам.
3. Интеграция с API Центробанка РФ (cbr.ru):
   - Добавить поле «Валюта» к каждой транзакции (RUB, USD, EUR, ...).
   - При добавлении транзакции в валюте, отличной от RUB, автоматически запрашивать курс валюты на дату транзакции с помощью API Центробанка РФ.
   - Пересчитывать сумму транзакции в рубли, используя полученный курс.
   - Сохранять как сумму в исходной валюте, так и сумму в рублях.
   - Предоставить пользователю возможность просматривать общую сумму продаж в разных валютах (с пересчетом по текущему курсу или курсу на дату транзакции).

Дополнительная задача

1. Продвинутая аналитика и прогнозирование:
   - ABC-анализ товаров: Разделить товары на три категории (A, B, C) на основе их вклада в общую сумму продаж (A - наиболее ценные, C - наименее ценные). Визуализировать результаты ABC-анализа.
   - XYZ-анализ товаров: Разделить товары на три категории (X, Y, Z) на основе стабильности спроса (X - стабильный спрос, Z - непредсказуемый спрос). Визуализировать результаты.
   - Прогнозирование продаж: Реализовать простой метод прогнозирования продаж на следующий месяц (например, метод скользящего среднего или экспоненциального сглаживания). Не требуется реализовывать сложные модели машинного обучения.
2. Генерация отчетов:
   - Реализовать генерацию отчетов в формате HTML (с таблицами и графиками) на основе данных о продажах и результатов аналитики. Использовать шаблонизатор (например, RazorEngineCore) для создания HTML-отчетов.

~ *Вариант №4* ~

- Базовая часть
- A-side
- B-side

[К варианту]

Базовая часть

Разработать консольное приложение «Менеджер проектов». Приложение должно:

1. Хранить данные о проектах и задачах.
   - Программа должна хранить информацию о:
     - Проектах:
       - ID проекта
       - Название проекта
       - Описание проекта (опционально)
     - Задачах
       - ID задачи
       - ID проекта (к которому относится задача)
       - Название задачи
       - Описание задачи (опционально)
       - Статус задачи (например, «Не начата», «В работе», «Завершена», «Отложена»)
   - Приложение должно читать задачи из файла при запуске.
   - Путь к файлу задач должен запрашиваться у пользователя.
   - Приложение должно проверять корректность пути и обрабатывать ошибки.
2. Предоставлять пользователю возможность:
   - Просмотра списка проектов.
   - Просмотра списка задач для выбранного проекта.
   - Добавления нового проекта.
   - Добавления новой задачи к выбранному проекту.
   - Редактирования проекта (название, описание).
   - Редактирования задачи (название, описание, статус).
   - Удаления проекта (со всеми его задачами).
   - Удаления задачи.
3. Выводить информацию в консоль в удобном формате. Главное, чтобы вся информация о проектах и задачах была выведена на экран.
4. Сохранять изменения обратно в файл при завершении работы приложения.

[К варианту]

[К варианту]

A-side

Основная часть

1. Фильтрация и сортировка:
   - Фильтрация задач по статусу.
   - Сортировка задач по ID, названию, статусу (внутри выбранного проекта).
   - Возможность комбинировать фильтры и сортировки.
2. Приоритеты задач:
   - Добавить поле «Приоритет» к задаче (например, «Высокий», «Средний», «Низкий»). Предусмотреть выбор из списка.
   - Фильтрация и сортировка по приоритету.
3. Поиск
   - Поиск задач по названию (частичное совпадение)

Дополнительная часть

1. Сроки выполнения задач:
   - Добавить поля «Дата начала» и «Дата окончания» (deadline) к задаче.
   - Визуально выделять просроченные задачи (например, красным цветом).
   - Добавить возможность фильтрации по просроченным задачам
   - Добавить возможность сортировки по дате окончания.
2. Исполнители задач:
   - Добавить поле «Исполнитель» к задаче.
   - Предусмотреть возможность ведения списка исполнителей (в отдельном файле или в рамках файла с задачами/проектами).
   - Фильтрация задач по исполнителю.
3. Подзадачи:
   - Реализовать возможность добавления подзадач к задачам (иерархическая структура).
   - Отображать подзадачи при просмотре информации о задаче.

[К варианту]

B-side

Основная задача

1. Визуализация с помощью Spectre.Console:
   - Таблицы: Отображение списков проектов и задач в табличном виде.
   - Дерево: Отображение иерархической структуры задач и подзадач (если реализованы подзадачи).
   - Progress Bar: Отображение общего прогресса выполнения проекта (на основе статусов задач).
2. Kanban доска
   - Отображать задачи проекта в виде Kanban-доски.
3. Уведомления:

   Реализовать отправку уведомлений о приближающихся сроках выполнения задач (deadline) и просроченных задачах.

   Уведомления могут быть (достаточно двух из перечня):

   - В консоль (при запуске приложения).
   - На электронную почту (потребуется настройка SMTP-сервера).
   - В Telegram (потребуется создание бота и получение токена).

Дополнительная задача

1. Интеграция с GitLab/GitHub API:
   - Предоставить пользователю возможность связать проект в приложении с репозиторием на GitLab или GitHub (пользователь вводит URL репозитория).
   - Реализовать хотя бы две из следующих функций:
     - Синхронизация задач: Создавать issues на GitLab/GitHub на основе задач в приложении и наоборот (задачи в приложении на основе issues).
     - Связь с коммитами: Позволять пользователю указывать ID коммитов, относящихся к задаче. Отображать информацию о коммитах (автор, дата, сообщение) при просмотре задачи.
     - Управление версиями (Milestones/Releases): Связывать задачи с Milestones (GitLab) или Releases (GitHub).

Внимание: для работы с GitLab/GitHub API потребуется получение токена доступа. Предусмотрите безопасное хранение токена (например, не храните его в коде, а запрашивайте у пользователя или используйте переменные среды).

3. Экспорт данных:

   Реализовать экспорт данных о проектах и задачах в форматы:
   - CSV
   - JSON
   - HTML (с использованием шаблонизатора)

~ *Вариант №5* ~

- Базовая часть
- A-side
- B-side

[К варианту]

Базовая часть

Разработать консольное приложение «Интерактивный справочник городов». Приложение должно:

1. **Хранить данные о книгах.**
   - Каждый город должен содержать информацию как минимум о:
     - Название города
     - Страна
     - Население (опционально)
     - Координаты (широта, долгота) (обязательно)
   - Приложение должно читать задачи из файла при запуске.
   - Путь к файлу задач должен запрашиваться у пользователя.
   - Приложение должно проверять корректность пути и обрабатывать ошибки.
2. Предоставлять пользователю возможность:
   - Просмотра списка городов (название, страна).
   - Добавления нового города (запрашивать название, страну, опционально население, координаты).
   - Редактирования информации о городе (по названию).
   - Удаления города (по названию).
3. Выводить информацию в консоль в удобном формате. Главное, чтобы вся информация о городах была выведена на экран.
4. Сохранять изменения обратно в файл при завершении работы приложения.

[К варианту]

[К варианту]

A-side

Основная часть

1. Фильтрация и сортировка:
   - Фильтрация городов по стране.
   - Сортировка городов по названию (в алфавитном порядке), населению (если есть).
2. Поиск:
   - Поиск города по названию
3. Вывод подробной информации
   - При выборе города из списка, показывать подробную информацию: Название, Страна, Население (если есть), Координаты.

Дополнительная часть

1. Расчет расстояния:
   - Реализовать функцию расчета расстояния между двумя городами (по координатам). Использовать формулу гаверсинусов или другую подходящую формулу.
   - Предоставить пользователю возможность выбрать два города из списка и рассчитать расстояние между ними.
2. Валидация данных:
   - При добавлении/редактировании города проверять корректность введенных данных:
     - Название города и страны не должны быть пустыми.
     - Население должно быть целым положительным числом (если введено).
     - Координаты должны быть числами в допустимых диапазонах (-90..90 для широты, −180..180 для долготы).
3. Сохранение истории поиска:

   Сохранять в отдельный файл историю поиска городов (какие города искал пользователь).

[К варианту]

[К варианту]

B-side

Основная задача

1. Визуализация с помощью Spectre.Console:
   - Таблица: Отображение списка городов в табличном виде.
   - Интерактивный выбор: Использовать Spectre.Console.AnsiConsole.Prompt для выбора городов из списка (например, при расчете расстояния).
2. Работа с картой (текстовая):
   - Отображать города на карте в консоле

Дополнительная задача

1. Интеграция с OpenStreetMap:
   - Геокодирование: При добавлении нового города, автоматически определять координаты по названию города. Предлагать пользователю подтвердить найденные координаты или ввести их вручную.
   - Обратное геокодирование: При просмотре информации о городе, получать дополнительную информацию (например, адрес, тип объекта) по координатам, используя обратное геокодирование.
2. Интеграция с API погоды (например, OpenWeatherMap):

   При просмотре информации о городе, показывать текущую погоду в этом городе, используя API OpenWeatherMap (или аналогичный сервис). Отображать температуру, облачность, осадки, скорость ветра.
3. Импорт/Экспорт:
   - Реализовать импорт данных о городах из форматов CSV и JSON.
   - Реализовать экспорт данных о городах в форматы CSV и JSON.

[К варианту]

~ *Вариант №6* ~

- Базовая часть
- A-side
- B-side

Базовая часть

Разработать консольное приложение «Билетный кассир». Приложение должно:

1. **Хранить данные о мероприятиях и билетах.**
   - Программа должна хранить информацию о:
     - **Мероприятиях:**
       - **ID мероприятия** (автоматически генерируется)
       - **Название мероприятия**
       - **Дата и время проведения**
       - **Место проведения**
       - **Описание мероприятия**
       - **Цена билета**
       - **Общее количество билетов**
       - **Количество проданных билетов** (изначально 0)
     - **Билетах:** (на базовом уровне, достаточно хранить только количество проданных билетов на каждое мероприятие, детализации билетов не требуется)
   - Приложение должно читать данные о мероприятиях из файла при запуске.
   - Путь к файлу данных должен запрашиваться у пользователя.
   - Приложение должно проверять корректность пути и обрабатывать ошибки.
2. **Предоставлять пользователю возможность:**
   - Просмотра списка всех мероприятий (в виде списка: ID, Название, Дата и время, Место, Цена, Остаток билетов).
   - Добавления нового мероприятия с указанием всей необходимой информации. ID мероприятия присваивается автоматически.
   - Редактирования информации о существующем мероприятии. Пользователь выбирает мероприятие по ID и может изменить любое поле (кроме ID и количества проданных билетов).
   - Удаления мероприятия. Пользователь выбирает мероприятие по ID.
   - Покупки билета на мероприятие. Пользователь выбирает мероприятие по ID и указывает количество билетов для покупки. Приложение должно проверять наличие билетов и уменьшать количество доступных билетов при успешной покупке.
3. **Выводить информацию о мероприятиях и билетах в консоль в удобном формате.** Главное, чтобы вся необходимая информация была выведена на экран.
4. **Сохранять изменения (информацию о мероприятиях и количестве проданных билетов) обратно в файл при завершении работы приложения.**

A-side

Основная часть

1. **Фильтрация и сортировка мероприятий:**
   - Фильтрация мероприятий по дате (диапазон дат), месту проведения, ценовому диапазону.
   - Сортировка мероприятий по дате, цене, названию.
   - Возможность комбинировать фильтры и сортировки.
2. **Поиск мероприятий:**
   - Поиск мероприятий по названию и месту проведения (частичное совпадение).
3. **Категории мероприятий:**
   - Добавить поле «Категория» к каждому мероприятию (например, «Концерт», «Спектакль», «Спорт», «Фестиваль»). Предусмотреть возможность выбора из списка предустановленных категорий, а также добавления новых категорий пользователем.
   - Реализовать фильтрацию мероприятий по категории.
   - Добавить вывод категории в информацию о мероприятии при просмотре и экспорте.
4. **Отчетность:**
   - Вывод общего количества мероприятий.
   - Вывод количества мероприятий по каждой категории.
   - Вывод общей суммы выручки от проданных билетов (для всех мероприятий).

Дополнительная часть

1. **Возврат билетов:**
   - Реализовать возможность возврата билетов. Пользователь выбирает мероприятие по ID и указывает количество билетов для возврата. Приложение должно увеличивать количество доступных билетов при успешном возврате.
   - Вести историю возвратов (можно в отдельном файле или в памяти, детализация истории возвратов не обязательна).
2. **Скидки и промокоды:**
   - Добавить возможность задавать скидку в процентах для каждого мероприятия.
   - Реализовать возможность применения промокодов при покупке билетов. Промокод может давать фиксированную скидку или процентную скидку на мероприятие. Промокоды могут быть одноразовыми или многоразовыми (на усмотрение разработчика, можно упростить до многоразовых).
   - Учитывать скидки и промокоды при расчете стоимости билетов при покупке.
3. **Экспорт/импорт:**
   - Реализовать экспорт и импорт данных о мероприятиях (без информации о продажах/возвратах билетов) в/из форматов CSV и JSON.

B-side

Основная часть

1. **Визуализация с помощью Spectre.Console:**
   - **Таблица:** Отображение списка мероприятий в виде интерактивной таблицы с возможностью прокрутки, фильтрации и сортировки прямо в таблице. Добавить стилизацию таблицы, например, выделение мероприятий с малым количеством оставшихся билетов.
   - **Календарь:** Отображение календаря на месяц, в котором отмечены дни, когда проводятся мероприятия. При наведении на день показывать названия мероприятий в этот день.
2. **Интерактивная покупка билетов:**
   - Использовать Spectre.Console.AnsiConsole.Prompt для создания интерактивного процесса покупки билетов. Например:
     - Выбор мероприятия из списка с использованием интерактивного выбора.
     - Запрос количества билетов для покупки с валидацией ввода.
     - Подтверждение покупки и вывод информации о заказе.
3. **Генерация билетов:**
   - После успешной покупки билетов, генерировать текстовый «электронный билет» в консоли. Билет должен содержать:
     - Название мероприятия
     - Дата и время
     - Место проведения
     - Количество билетов
     - Уникальный номер билета (можно сгенерировать простой уникальный ID)
     - QR-код (можно сгенерировать простой QR-код, содержащий номер билета или ссылку на мероприятие, используя библиотеку QRCoder).

Дополнительная часть

1. **Система бронирования билетов:**
   - Реализовать возможность бронирования билетов на ограниченное время (например, 15 минут).
   - При бронировании билеты временно резервируются и не доступны для покупки другими пользователями. Если в течение заданного времени бронь не выкуплена, билеты снова становятся доступными.
   - Реализовать механизм управления бронями (просмотр активных броней, отмена брони).
2. **Аналитика продаж:**
   - Реализовать сбор статистики продаж билетов по мероприятиям, категориям, датам и времени.
   - Визуализировать результаты аналитики с помощью графиков и диаграмм в консоли (использовать Spectre.Console). Например:
     - График продаж билетов по дням недели/месяцам.
     - Диаграмма популярности категорий мероприятий.
     - Рейтинг мероприятий по выручке.
3. **Интеграция с платежной системой (имитация):**
   - Имитировать интеграцию с платежной системой. Вместо реальной оплаты, реализовать ввод данных «банковской карты» (можно произвольный формат для имитации) и вывод сообщения об «успешной оплате» или «ошибке оплаты».
   - При успешной «оплате» - подтверждать покупку билетов и генерировать билет. При «ошибке оплаты» - отменять покупку и выводить сообщение об ошибке.

~ *Вариант №7* ~

- Базовая часть
- A-side
- B-side

Базовая часть

Разработать консольное приложение «Von Dutch». Приложение должно:

1. **Хранить данные о словарях.**
   - Приложение должно поддерживать несколько языковых пар (например, Английский-Русский, Испанский-Английский, Французский-Английский, не менее 3 штук).
   - Для каждой языковой пары программа должна хранить словарь.
   - Словарь представляет собой набор пар «слово-перевод».
   - Данные словарей должны читаться из файлов при запуске приложения.
   - Формат файлов словарей – на усмотрение разработчика (например, CSV, JSON, простой текстовый файл).
   - Путь к папке со словарями должен запрашиваться у пользователя при первом запуске.
   - Приложение должно проверять наличие файлов словарей и обрабатывать ошибки.
2. **Предоставлять пользователю возможность:**
   - Выбора текущей языковой пары для перевода (например, через меню или ввод кода языковой пары).
   - Перевода слова с исходного языка на целевой язык. Пользователь вводит слово на исходном языке, приложение ищет перевод в словаре и выводит его в консоль.
   - Добавления новых слов и их переводов в текущий словарь. Пользователь вводит слово на исходном языке и его перевод на целевой язык.
   - Редактирования существующих слов и их переводов в текущем словаре. Пользователь выбирает слово для редактирования (например, по исходному слову) и может изменить перевод.
   - Удаления слов из текущего словаря. Пользователь выбирает слово для удаления.
3. **Выводить информацию о переводах и словарях в консоль в удобном формате.** Главное, чтобы вся необходимая информация была выведена на экран.
4. **Сохранять изменения в словарях (добавление, редактирование, удаление слов) обратно в файлы при завершении работы приложения.** Нужно быть **forever** готовыми к изменениям в словаре!

A-side

Основная часть

1. **Расширенный поиск и функциональность словаря:**
   - **Поиск перевода в обе стороны:** Пользователь может ввести слово как на исходном, так и на целевом языке, и приложение должно находить соответствующий перевод (если он есть в словаре).
   - **Нечеткий поиск:** Реализовать нечеткий поиск слов (например, с использованием алгоритма Левенштейна или похожего). Если точное соответствие не найдено, предлагать пользователю варианты наиболее близких слов из словаря, как будто приложение говорит: «**Close, but not quite**».
   - **Просмотр всего словаря:** Предоставить возможность просмотра всего словаря для выбранной языковой пары в виде списка (например, в алфавитном порядке исходных слов).
2. **История переводов:**
   - Сохранять историю запросов на перевод (например, в список или файл).
   - Предоставить возможность просмотра истории переводов (например, последние 10-20 запросов). Чтобы можно было вернуться и вспомнить «**what you said last night**».
3. **Статистика использования словаря:**
   - Вести статистику использования словаря: количество переводов, количество добавленных слов, количество отредактированных слов и т.д.
   - Выводить статистику в консоль.

Дополнительная часть

1. **Поддержка нескольких форматов словарей:**
   - Реализовать поддержку чтения и записи словарей в нескольких форматах (например, CSV, JSON, XML).
   - Предоставить пользователю возможность выбора формата словарей при первом запуске или в настройках.
2. **Перевод фраз и предложений (базовый уровень):**
   - Реализовать базовую возможность перевода фраз и предложений. Например, разделить фразу на слова, перевести каждое слово по словарю и собрать переведенную фразу. (Конечно, это будет скорее дословный перевод, как в ранних версиях Google Translate, но этого достаточно).
   - Если слово не найдено в словаре, оставлять его в исходном виде или предлагать варианты нечеткого поиска (из основной части A-side).

B-side

Основная задача

1. **Визуализация с помощью Spectre.Console:**
   - **Интерактивное меню:** Использовать Spectre.Console.AnsiConsole.Prompt для создания красивого и интерактивного меню управления приложением.
   - **Таблицы для словарей и истории:** Отображать словари и историю переводов в виде интерактивных таблиц Spectre.Console с возможностью пролистывания и стилизации.
2. **Продвинутый перевод фраз и предложений:**
   - Реализовать более продвинутый перевод фраз и предложений, используя, например, простые правила морфологии и синтаксиса для **хотя бы одной** языковой пары. Не ждем уровня нейронных сетей, но что-то чуть лучше дословного перевода, как будто «**doing it for fun**» с лингвистикой.
   - Можно использовать простые библиотеки: Nestor, Morpher.API, SharpNLP, OpenNLP
3. **Голосовой ввод и вывод (текстовый, на базовом уровне):**
   - Реализовать базовую возможность голосового ввода слова или фразы для перевода (Microsoft.CognitiveServices.Speech).
   - Реализовать текстовый вывод перевода голосом (например, используя System.Speech.Synthesis, если это доступно и не требует сложной настройки, или другую простую библиотеку для синтеза речи). Чтобы приложение могло «сказать» перевод.

Дополнительная задача

1. **Контекстный перевод (базовый уровень):**
   - Реализовать базовый уровень контекстного перевода. Например, для некоторых слов, которые имеют несколько значений в зависимости от контекста, предлагать пользователю варианты перевода в зависимости от фразы или предложения, в котором используется слово.
   - Можно использовать простые правила или наборы примеров для определения контекста.
2. **Обучение словаря:**
   - Реализовать возможность для пользователя «обучать» словарь. Например, если приложение не нашло перевод или предложило неверный перевод, пользователь может указать правильный перевод, и приложение должно запомнить его и использовать в дальнейшем.
   - Можно реализовать простую систему «оценки» переводов пользователем (например, «верно/неверно») и на основе этих оценок корректировать словарь или приоритезировать варианты перевода. Чтобы словарь становился **«better and better»** со временем.

~ *Примечания* ~

А можно свой формат файла для хранения файлов?

Да, конечно. Хоть SQLite используйте для хранения данных.

Если у вас не первый вариант - там нужно использовать указанный в условии формат.

Что делать, если файл уже существует?

Необходимо запросить подтверждение пользователя на перезапись файла.

Требования к фильтрации и сортировкам

Нужно реализовать возможность комбинировать фильтры и сортировки.

Например, установить набор фильтров:
- Сортировка №1
- Фильтрация №1
- Фильтрация №2
- Сортировка №2

И затем предоставить пользователю возможность удалить любой из пунктов. Например, первую фильтрацию, чтобы получилось:
- Сортировка №1
- Фильтрация №2
- Сортировка №2

Все действия применяются последовательно, от первого до последнего, в порядке добавления.