**Лабораторна робота №3. Todo-list**

**Виконавець: Гончаренко Микита Тв-12**

Main.rs – це головний файл програми що взаємодіє з іншими її файлами

У функції main():

* Завантажується список завдань за допомогою функції load\_tasks(). Якщо завантаження не вдалося (наприклад, файл порожній або відсутній), ініціалізується порожній список.
* Далі запускається основний цикл програми, в якому постійно відображається меню та чекається взаємодії користувача. Це забезпечує безперервну роботу програми, поки вона не буде закрита.
* Перед завершенням програми всі поточні завдання зберігаються за допомогою функції save\_tasks().

cli.rs - Цей модуль відповідає за реалізацію командного інтерфейсу через який користувач може взаємодіяти з програмою. В ньому знаходиться меню, яке дозволяє додавати, видаляти, редагувати завдання, позначати їх як виконані та переглядати всі завдання.

**Основні функції:**

1. **show\_menu(tasks: &mut Vec<Task>)**:
   * Виводить меню з варіантами дій, що користувач може вибрати.
   * Залежно від вибору користувача, викликаються відповідні функції для роботи з завданнями.
   * Користувач може вибрати одну з наступних опцій:
     + Додати завдання.
     + Видалити завдання.
     + Редагувати завдання.
     + Позначити завдання як виконане.
     + Показати всі завдання.
     + Вийти з програми.
2. **add\_task(tasks: &mut Vec<Task>)**:
   * Додає нове завдання до списку.
   * Виводить запити на введення назви та опису завдання.
   * Генерує унікальний ID для кожного нового завдання (ID дорівнює останньому ID + 1 або 1, якщо завдання перше).
   * Додає завдання до вектору.
3. **delete\_task(tasks: &mut Vec<Task>)**:
   * Видаляє завдання з списку за ID.
   * Запитує у користувача ID завдання для видалення.
   * Якщо завдання з таким ID знайдено, воно видаляється з вектору.
4. **edit\_task(tasks: &mut Vec<Task>)**:
   * Дозволяє редагувати існуюче завдання за ID.
   * Запитує у користувача нову назву та опис для завдання.
   * Змінює значення назви та опису у завданні.
5. **mark\_completed(tasks: &mut Vec<Task>)**:
   * Позначає завдання як виконане.
   * Запитує у користувача ID завдання.
   * Якщо завдання знайдено, змінюється його статус на "виконане".
6. **list\_tasks(tasks: &Vec<Task>)**:
   * Виводить список усіх завдань з їхніми ID, назвами, описами та статусами (виконано/не виконано).
   * Якщо список завдань порожній, виводить відповідне повідомлення.

Storage.rs - Цей модуль відповідає за збереження та завантаження списку завдань у/з файлу формату JSON. Ось детальний опис кожної функції:

**Опис модулю**

1. **save\_tasks(tasks: &Vec<Task>) -> io::Result<()>**:
   * Ця функція відповідає за збереження списку завдань у файл tasks.json.
   * Використовує метод File::create для створення нового файлу або перезапису існуючого.
   * Для запису даних використовується BufWriter, який ефективно буферизує запис у файл.
   * Функція перетворює список завдань у формат JSON за допомогою serde\_json::to\_writer і записує його в файл.
   * Якщо запис у файл пройшов успішно, функція повертає Ok(()).
2. **load\_tasks() -> io::Result<Vec<Task>>**:
   * Ця функція відповідає за завантаження списку завдань з файлу tasks.json.
   * Якщо файл існує, відкривається за допомогою File::open, і дані зчитуються за допомогою BufReader, щоб забезпечити буферизацію при читанні.
   * Використовує serde\_json::from\_reader для десеріалізації даних у структуру Vec<Task>.
   * Якщо файл не існує, функція повертає порожній вектор (Vec::new()), що дозволяє почати з чистого списку завдань.
   * Якщо операція успішна, повертається вектор завдань, в іншому випадку — помилка.

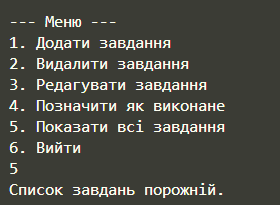
Task.rs - Цей код визначає структуру Task, яка представляє завдання з певними властивостями, а також містить методи для роботи з цією структурою. Ось детальний опис коду:

**Опис структури Task та її методів**

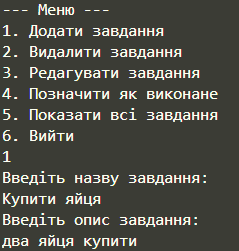
1. **Структура Task**:
   * Структура містить 4 поля:
     + id: u32 — унікальний ідентифікатор завдання.
     + title: String — назва завдання.
     + description: String — опис завдання.
     + completed: bool — статус завдання (виконано чи ні).
2. **Атрибути #[derive(Serialize, Deserialize, Debug)]**:
   * Serialize та Deserialize: дозволяють структурам серіалізуватися (перетворюватися в JSON) і десеріалізуватися (відновлюватися з JSON) за допомогою бібліотеки serde. Це необхідно для збереження завдань у файлі формату JSON.
   * Debug: дозволяє використовувати макрос println!("{:?}", task) для виведення структури у консоль у зручному для дебагу вигляді.
3. **Метод new**:
   * Статичний метод для створення нового завдання.
   * Приймає параметри id, title, та description і ініціалізує нове завдання з цими значеннями, при цьому за замовчуванням встановлює статус виконання (completed) в false.
4. **Метод mark\_completed**:
   * Цей метод змінює значення поля completed на true, позначаючи завдання як виконане.

**Приклад виконання:**

Головне меню та ми намагаємось вивести список товарів але він пустий



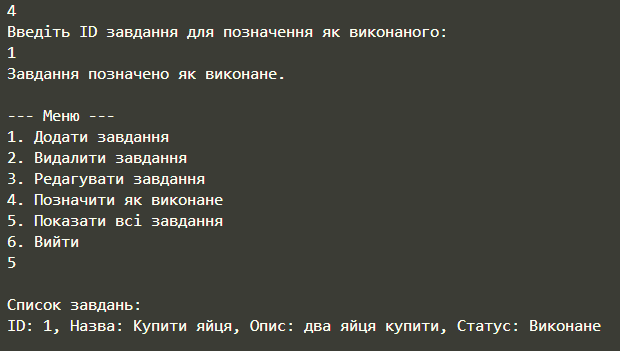
Додаємо нове завдання



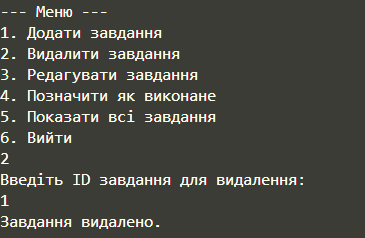
Тепер список не порожній



Можна позначити завдання як виконане



Також можна його видалити



**Висновок**

Розроблений додаток для управління завданнями дозволяє користувачам зручно створювати, редагувати, видаляти та позначати завдання як виконані. Застосування мови програмування Rust дозволяє досягти високої продуктивності та безпеки при обробці даних. Окремо варто відзначити використання бібліотеки serde для серіалізації та десеріалізації даних у форматі JSON, що дозволяє зберігати завдання в локальних файлах і відновлювати їх при перезапуску програми.