Инструкция «Описание проекта»

Проект: To-Do менеджер на Go для командной строки

Результаты:

- Освоение базовых навыков разработки CLI-приложения на Go
- Знакомство с парсингом флагов и подкоманд
- Навыки работы с форматами хранения данных: JSON и CSV
- Организация структуры Go-проекта и модульного кода
- Базовые навыки тестирования и обработки ошибок

1. Цели проекта

1. Разработать простое консольное приложение на языке Go для управления задачами.
2. Научиться оформлять команды и флаги для пользовательского интерфейса в терминале.
3. Освоить чтение/запись файлов в формате JSON и CSV для хранения данных.
4. Научиться структурировать проект по пакетам и разделять логику.

2. Функциональные требования

Обязательный функционал

- **add** — добавить новую задачу с текстовым описанием
- **list** — вывести список задач с возможностью фильтрации:
- --filter all (по умолчанию) — все задачи;
- --filter done — только выполненные;
- --filter pending — только невыполненные.
- **complete** — отметить задачу выполненной по её уникальному ID
- **delete** — удалить задачу по ID
- **export** — экспортировать текущий список задач в файл (JSON или CSV)
- **load** — импортировать (загрузить) задачи из указанного файла (JSON или CSV) и сохранить в основной файл хранения перезаписав его содержимое

3. Форматы данных

JSON (основной формат хранения)

- Файл tasks.json хранит массив объектов с атрибутами:
- id (int) — уникальный идентификатор;
- description (string) — текст задачи;

- done (bool) — признак выполнения;

Пример:

```
[
  {"id":1, "description":"Купить продукты", "done":false},
  {"id":2, "description":"Изучить Go", "done":true}
]
```

CSV (для импорта/экспорта)

- Файл разделён запятыми, первая строка — заголовок.

Пример:

ID,Description,Done

1,Купить продукты,false

2,Изучить Go,true

4. Рекомендуемая структура проекта

```
todo-app/
├── cmd/
│   └── todo/
│       ├── main.go              # точка входа: парсинг аргументов, запуск команд
├── internal/
│   ├── todo/
│   │   ├── task.go              # модель Task и методы
│   │   └── manager.go           # бизнес-логика (Add, List, Complete, Delete)
│   └── storage/
│       ├──  json_storage.go     # функции LoadJSON, SaveJSON
│       └──  csv_storage.go      # функции LoadCSV, SaveCSV
├── go.mod                       # модули Go
└── README.md                    # документация и примеры использования
```

5. Советы по реализации

Описание cmd/todo/main.go

1. Считайте команду из os.Args[1] и аргументы в args := os.Args[2:].
2. Подгрузите текущие задачи:

```
tasks, err := storage.LoadJSON(dataFile)
```

3. Для каждой команды создайте flag.FlagSet, задайте нужные флаги и вызовите

```
Parse(args)
 case "list":
  listCmd := flag.NewFlagSet("list", flag.ExitOnError)
  filter := listCmd.String("filter", "all", "all, done,
pending")
  err = listCmd.Parse(args)
  if err != nil {
   fmt.Println(err)
   os.Exit(1)
}
```

4. В switch cmd:
   - **add**: проверка *desc, вызов todo.Add и storage.SaveJSON
   - **list**: флаг --filter, вызов todo.List и печать результатов
   - **complete/delete**: флаг --id, вызов соответствующей функции и сохранение
   - **load**: флаг --file, определение формата по расширению (filepath.Ext), загрузка через LoadJSON/LoadCSV, сохранение в tasks.json с помощью SaveJSON
   - **export**: флаги --format, --out, вызов SaveJSON/SaveCSV

5. При ошибках используйте fmt.Println и os.Exit(1).

Пример вызова:

```
./todo add --desc="Купить книги"
./todo load --file=tasks_export.csv
```

Создание файла при первом запуске:

- В LoadJSON проверяйте наличие файла с помощью os.Stat, и при отсутствии создавайте пустой через SaveJSON

Документация:

- Опишите установку и запуск в README.md.
- Приведите примеры команд с ожидаемым выводом.

Обработка ошибок:

- Всегда проверяйте ошибки
- Возвращайте осмысленные сообщения пользователю

Сигнатуры ключевых функций:

```
// internal/todo/manager.go
func Add(tasks []Task, desc string) []Task
func List(tasks []Task, filter string) []Task
func Complete(tasks []Task, id int) ([]Task, error)
func Delete(tasks []Task, id int) ([]Task, error)

// internal/storage/json_storage.go
func LoadJSON(path string) ([]todo.Task, error)
func SaveJSON(path string, tasks []todo.Task) error
```

```
// internal/storage/csv_storage.go
func LoadCSV(path string) ([]todo.Task, error)
func SaveCSV(path string, tasks []todo.Task) error
```

Работа с JSON и CSV:

- **Чтение JSON:**
  1. Используйте os.Stat для проверки существования файла.
  2. При отсутствии создайте пустой файл через SaveJSON.
  3. Считайте содержимое os.ReadFile.
  4. Разберите массив задач через json.Unmarshal.
- **Запись JSON:**
  1. Сериализуйте tasks в формат JSON с отступами json.MarshalIndent.
  2. Запишите байты в файл с правами 0644 через os.WriteFile.
- **Чтение CSV:**
  1. Откройте файл через os.Open.
  2. Считайте все записи через csv.NewReader(...).ReadAll().
  3. Пропустите заголовок (первая строка).
  4. Преобразуйте строки: strconv.Atoi для ID, strconv.ParseBool для Done.
  5. Сформируйте срез []todo.Task.
- **Запись CSV:**
  1. Создайте (или перезапишите) файл через os.Create.
  2. Инициализируйте csv.NewWriter и запишите заголовок []string{"ID","Description","Done"}.
  3. Для каждой задачи сформируйте строку []string{strconv.Itoa(ID), Description, strconv.FormatBool(Done)}.
  4. Не забудьте writer.Flush().