📘 Documentação Técnica do Projeto Tarefas

📄 Capa

Projeto: Tarefas – Gerenciador de Tarefas em CLI

Tecnologia: Node.js (JavaScript ES Modules)

Tipo: Aplicação de Linha de Comando (CLI)

Finalidade: Projeto educacional para treinamento técnico em JavaScript

Autor: Desenvolvedor do Projeto

Data: Documento técnico

📑 Índice

🕐 Objetivos do Projeto

Este projeto tem como objetivo principal **o desenvolvimento técnico e prático em JavaScript com Node.js**, aplicando conceitos amplamente utilizados no mercado de trabalho.

Objetivos específicos:

- Aplicar JavaScript moderno com **ES Modules** (`import/export`)
- Utilizar **programação assíncrona** com `async/await`
- Implementar estruturas de dados eficientes como **Map**
- Trabalhar com **persistência de dados local** usando JSON
- Criar interfaces interativas em terminal (CLI)

• Organizar código em **arquitetura modular**
• Controlar fluxo de navegação entre menus

🧰 Tecnologias e Bibliotecas Utilizadas

🕰️ Node.js

Ambiente de execução JavaScript no lado do servidor, responsável por permitir o uso de arquivos, terminal e módulos nativos.

🕰️ JavaScript (ES Modules)

Utilização de módulos modernos (`import` / `export`), promovendo organização, reutilização e encapsulamento do código.

🕰️ @clack/prompts

Biblioteca utilizada para criar **menus interativos no terminal**, permitindo seleção de opções, entrada de texto e controle de cancelamento.

Principais recursos usados: - `select()` – menus de escolha - `text()` – entrada de texto - `isCancel()` – detecção de cancelamento - `log`, `outro` – mensagens formatadas

🕰️ chalk

Biblioteca usada para **estilizar textos no terminal**, aplicando cores e destaques visuais, principalmente para status das tarefas.

🕰️ fs (File System)

Módulo nativo do Node.js responsável pela leitura e escrita do arquivo `tasks.json`, garantindo persistência dos dados.

🕰️ path

Módulo nativo utilizado para manipulação segura de caminhos de arquivos.

🗂️ Estrutura de Diretórios

```
TAREFA/
│
├── node_modules/
├── src/
│   ├── index.js
│   ├── manager/
│   │   └── tasks.js
│   └── menus/
```

```
│       ├── main.js
│       ├── create.js
│       ├── list.js
│       └── update.js
│
├── tasks.json
├── package.json
├── package-lock.json
```

📄 Descrição Geral dos Arquivos

🕰️ `src/index.js`

Ponto de entrada da aplicação

Responsável por: - Importar bibliotecas externas - Exibir a introdução do programa - Iniciar o menu principal

---

🕰️ `src/manager/tasks.js`

Módulo central de gerenciamento de tarefas

Responsável por: - Persistência dos dados (`tasks.json`) - Criação, leitura, atualização e exclusão de tarefas - Conversão entre `Map` e `Array` - Aplicação de cores conforme o status da tarefa

---

🕰️ `src/menus/main.js`

Menu principal da aplicação

Responsável por: - Exibir as opções principais - Direcionar o fluxo para os submenus - Encerrar o programa

---

🕰️ `src/menus/create.js`

Menu de criação de tarefas

Responsável por: - Solicitar o nome da tarefa - Validar duplicidade - Criar e salvar a tarefa

---

🕰️ `src/menus/list.js`

Menu de listagem de tarefas

Responsável por: - Listar tarefas existentes - Permitir selecionar uma tarefa - Encaminhar para o menu de atualização

---

🕰️ `src/menus/update.js`

Menu de atualização de tarefas

Responsável por: - Exibir detalhes da tarefa - Alterar nome - Alterar status - Excluir tarefa

---

🔍 Detalhamento Técnico por Arquivo (Nível Técnico)

A seguir está a descrição detalhada de cada arquivo, com explicação técnica das funções, objetos e responsabilidades dentro do sistema.

📄 `src/manager/tasks.js`

Importações

- `fs`: leitura e escrita de arquivos
- `path`: manipulação de caminhos
- `chalk`: estilização de texto no terminal

Inicialização do arquivo

- Verifica se `tasks.json` existe
- Cria o arquivo caso não exista
- Lê o conteúdo e converte para objeto JavaScript

Estrutura de Dados

```
const tasks = new Map();
```

- `Map` permite chave única (nome da tarefa)
- Facilita busca, atualização e remoção

Objeto `taskManager`

`tasks`

- Armazena todas as tarefas em memória

`save()`

- Converte o Map em Array
- Salva no arquivo `tasks.json`

`create(taskData)`

- Recebe um objeto de tarefa
- Adiciona ao Map
- Persiste no arquivo

`toArray()`

- Converte Map para Array
- Usado para listagens e salvamento

`colorStatus(status)`

- Aplica cores conforme o status:
- Em andamento → Amarelo
- Concluída → Azul
- Cancelada → Vermelho

📄 `src/menus/main.js`

Função `mainMenu()`

- Exibe menu principal
- Usa `select()` para escolha interativa
- Opções:
- Criar tarefa
- Listar tarefas
- Sair

Fluxo de Controle

- `switch(option)` direciona para o menu correspondente
- `isCancel()` trata cancelamento do usuário

📄 `src/menus/create.js`

Função `createTaskMenu()`

Processo:

1. Solicita o nome da tarefa
2. Valida se o usuário cancelou
3. Verifica duplicidade
4. Cria objeto da tarefa
5. Salva usando `taskManager.create()`
6. Retorna ao menu principal

Estrutura da Tarefa

```
{
  name: string,
  status: "em andamento",
  createdAt: ISODate
}
```

📄 `src/menus/list.js`

Função `listTaskMenu()`

- Verifica se há tarefas cadastradas
- Converte tarefas em opções do menu
- Exibe status colorido
- Permite retornar ao menu principal

Integração

- Chama `updateTaskMenu()` ao selecionar tarefa

📄 `src/menus/update.js`

Função `updateTaskMenu(taskName)`

Exibição de Informações

- Nome
- Status
- Data de criação formatada

Opções Disponíveis

- Alterar nome
- Alterar status
- Deletar
- Voltar

Alterar Nome

- Valida duplicidade
- Atualiza chave do Map
- Persiste alterações

Alterar Status

- Exibe apenas status diferentes do atual
- Atualiza a tarefa selecionada

Excluir

- Remove do Map
- Atualiza arquivo

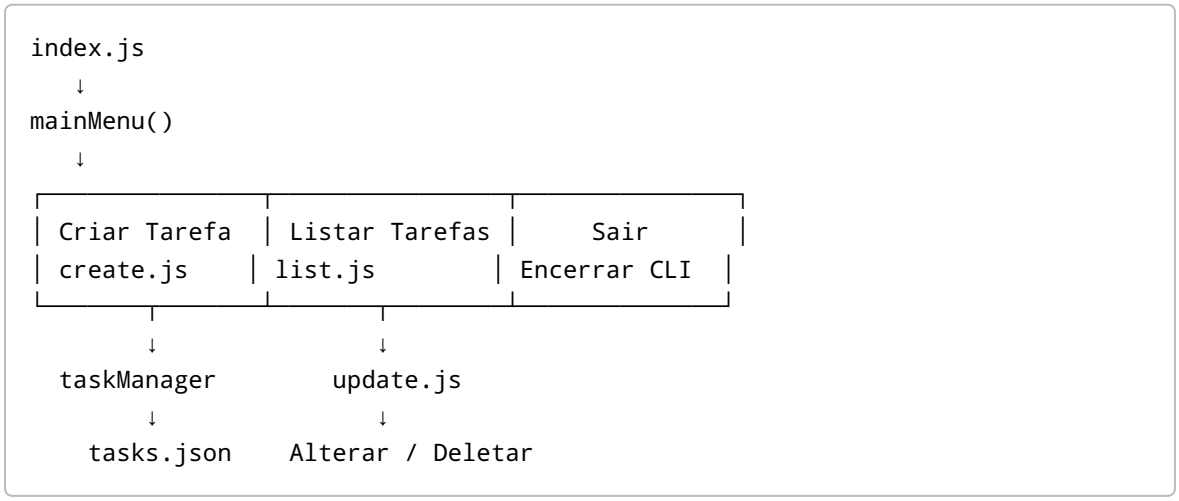
🕐 Fluxo de Execução do Programa

1. O usuário executa o arquivo `index.js`, que atua como ponto de entrada da aplicação.
2. O sistema exibe uma mensagem de introdução no terminal.
3. O `mainMenu()` é chamado, exibindo o menu principal.
4. O usuário escolhe uma das opções disponíveis:
5. Criar tarefa → direciona para `createTaskMenu()`
6. Listar tarefas → direciona para `listTaskMenu()`
7. Sair → encerra o programa
8. As ações do usuário manipulam o `taskManager`, que gerencia os dados em memória e no arquivo `tasks.json`.
9. Após cada operação, o sistema retorna ao menu apropriado, mantendo o fluxo contínuo da aplicação.

Conceitos Técnicos Trabalhados

- JavaScript moderno (ES Modules)
- Programação assíncrona (`async/await`)
- Estrutura de dados `Map`
- Persistência de dados com JSON
- Manipulação de arquivos com `fs`
- Interfaces interativas em terminal (CLI)
- Organização modular do código
- Controle de fluxo e navegação entre menus

📐 Diagrama Lógico Textual do Sistema



🔗 Conclusão

Este projeto demonstra uma aplicação CLI completa, funcional e bem estruturada, sendo ideal para **aprendizado prático de JavaScript moderno**, organização de código e manipulação de dados persistentes.

O código está funcional e atende ao objetivo de aprendizado, não sendo necessária nenhuma modificação estrutural.

📌 **Documento elaborado para fins educacionais e de estudo aprofundado da linguagem JavaScript.**