



# UNIVERSIDADE FEDERAL DO PARANÁ ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Rafael Fernando Nunho Correr-GRR20235365

Douglas Cleverson Fróis GRR – 20236486

Sally Ai Saito GRR - 20236485

Desenvolvimento WEB I

Prof. Roberson Cesar Alves de Araujo

CURITIBA-PR

2024

# 1. Introdução

## Visão Geral

O projeto "MEMRISE" é uma aplicação web desenvolvida para ajudar os usuários a gerenciar suas tarefas diárias. Ele oferece funcionalidades como cadastro de usuários, login, criação, edição e exclusão de tarefas, bem como acompanhamento do progresso das tarefas.

## Objetivos

O objetivo da aplicação é fornecer uma plataforma intuitiva e eficiente para que os usuários possam organizar suas tarefas diárias de forma eficaz, acompanhando seu progresso ao longo do tempo.

## Público-Alvo

O público-alvo da aplicação são pessoas que desejam uma maneira simples e eficiente de gerenciar suas tarefas diárias, seja em âmbito pessoal ou profissional.

## Contexto

A aplicação resolve o problema de organização de tarefas diárias em um contexto em que a sobrecarga de informações e a falta de tempo são comuns. Ela oferece uma maneira conveniente e centralizada de gerenciar tarefas, ajudando os usuários a manterem-se produtivos e organizados.

## 2. Guia de Instalação

### - Requisitos do Sistema

- Sistema Operacional: Windows, Linux ou macOS
- Banco de Dados: SQL Server
- Node.js (v14.x ou superior)
- .NET 6 SDK
- Visual Studio Code (ou IDE de sua preferência)

### - Dependências

- Entity Framework Core
- React.js
- Vite
- React Hook Form

### - Passo a Passo da Instalação

1. Clone o repositório do projeto “<https://github.com/WindFox8/Memrise.git>”.
2. Abrir a pasta Memrise - Back no VS Code e utilizar os seguintes comandos no terminal:
  - dotnet ef database update
  - dotnet watch run
3. Abrir a pasta Memrise - Front no VS Code e utilizar os seguintes comandos no terminal:
  - npm install
  - npm run dev

## 3. Guia de Uso

### - Primeiros Passos

Após a instalação, acesse a aplicação através do navegador web e faça login com sua conta de usuário. Se ainda não tiver uma conta, você pode se cadastrar na página inicial.

## - Funcionalidades Principais

- **Cadastro de Usuário:** Permite que novos usuários se cadastrem fornecendo nome, email e senha.
- **Login de Usuário:** Autentica os usuários existentes com email e senha.
- **Gerenciamento de Tarefas:** Permite criar, editar, excluir e marcar tarefas como concluídas ou pendentes.
- **Visualização de Estatísticas:** Mostra estatísticas como número de tarefas concluídas, tarefas para o dia e taxa de conclusão.

## 4. Guia de Desenvolvimento

### - Estrutura do Código

- O backend segue uma arquitetura em camadas, com separação clara de responsabilidades entre Controllers, Services e Repositories.
- O frontend é estruturado com base nos princípios de componentização do React.js, organizando os componentes em pastas dedicadas dentro do diretório **src**.

### - Configuração do Ambiente de Desenvolvimento

- Configure o ambiente de desenvolvimento instalando as dependências listadas no item 2.
- Utilize o Visual Studio Code ou outra IDE de sua preferência para editar o código.

- Execute o backend e o frontend simultaneamente durante o desenvolvimento.

## 5. Referência de API

### - Endpoints

#### Tarefas

- **GET /api/Tarefas/Usuario/{idUserario}**: Retorna todas as tarefas de um usuário específico.
  - Método HTTP: GET
  - Parâmetros de URL: idUsuario (int)
  - Resposta: Uma lista de objetos **Tarefa**.
- **GET /api/Tarefas/{id}**: Retorna os detalhes de uma tarefa específica.
  - Método HTTP: GET
  - Parâmetros de URL: id (int)
  - Resposta: Objeto **Tarefa** contendo os detalhes da tarefa.
- **POST /api/Tarefas/CriarTarefa**: Cria uma nova tarefa.
  - Método HTTP: POST
  - Corpo da Requisição: Objeto **Tarefa** contendo os detalhes da nova tarefa.
- **PATCH /api/Tarefas/{id}/inverter-prioridade**: Inverte a prioridade de uma tarefa específica.
  - Método HTTP: PATCH
  - Parâmetros de URL: id (int)

- **PATCH /api/Tarefas/{id}/inverter-finalizado:** Altera o estado de finalização de uma tarefa específica.
  - Método HTTP: PATCH
  - Parâmetros de URL: id (int)
- **DELETE /api/Tarefas/{id}:** Exclui uma tarefa específica.
  - Método HTTP: DELETE
  - Parâmetros de URL: id (int)

## Usuários

- **POST /api/Usuarios/Cadastrar:** Registra um novo usuário na aplicação.
  - Método HTTP: POST
  - Corpo da Requisição: Objeto **Usuario** contendo os detalhes do novo usuário.
- **POST /api/Usuarios/Login:** Realiza a autenticação de um usuário existente.
  - Método HTTP: POST
  - Corpo da Requisição: Objeto **Credenciais** contendo o email e senha do usuário.
  - Resposta: Objeto **Usuario** contendo os detalhes do usuário autenticado.
- **GET /api/Usuarios/VerificarEmail?email={email}:** Verifica se um determinado email já está cadastrado na aplicação.
  - Método HTTP: GET
  - Parâmetros de URL: email (string)
  - Resposta: **true** se o email já estiver cadastrado, **false** caso contrário.
- **DELETE /api/Usuarios/{id}:** Exclui um usuário específico.
  - Método HTTP: DELETE
  - Parâmetros de URL: id (int)

## Diretórios

- **POST /api/Diretorios/CriarDiretorio/{idUserario}:** Cria um novo diretório para um usuário específico.
  - Método HTTP: POST
  - Parâmetros de URL: idUsuario (int)
  - Corpo da Requisição: Objeto **Diretorio** contendo os detalhes do diretório a ser criado.
- **GET /api/Diretorios/Usuario/{idUserario}/Nomes:** Retorna os nomes dos diretórios pertencentes a um usuário específico.
  - Método HTTP: GET
  - Parâmetros de URL: idUsuario (int)
  - Resposta: Uma lista de objetos **DiretorioldNome**, contendo os IDs e nomes dos diretórios.
- **PUT /api/Diretorios:** Altera o nome de um diretório existente.
  - Método HTTP: PUT
  - Corpo da Requisição: Objeto **DiretorioldNome** contendo o ID e o novo nome do diretório.

- **DELETE /api/Diretorios/{id}**: Exclui um diretório específico, incluindo todas as tarefas associadas.
  - Método HTTP: DELETE
  - Parâmetros de URL: id (int)

### Códigos de Status

- **200 OK**: Requisição bem-sucedida.
- **201 Created**: Recurso criado com sucesso.
- **204 No Content**: Requisição bem-sucedida sem conteúdo para retornar.
- **400 Bad Request**: A requisição possui um formato inválido ou faltam parâmetros obrigatórios.
- **401 Unauthorized**: Falha na autenticação do usuário.
- **403 Forbidden**: O usuário não possui permissão para acessar o recurso solicitado.
- **404 Not Found**: Recurso não encontrado.
- **500 Internal Server Error**: Erro interno do servidor.

## 6. Componentes

### - LeftAside

O componente **LeftAside** é responsável por exibir os diretórios do usuário, oferecendo opções para adicionar, editar e deletar diretórios, além de permitir a filtragem das tarefas.

### Props

- **taskChange**: Função para atualizar a lista de tarefas após alterações nos diretórios.
- **filter**: Número representando o filtro atual aplicado às tarefas.
- **setFilter**: Função para definir o filtro das tarefas.

### Estado

- **directories**: Lista de diretórios do usuário.
- **error**: String de erro, se houver algum problema ao buscar os diretórios.
- **showDirForm**: Booleano indicando se o formulário de adição de diretório está sendo exibido.
- **dirUpdate**: Booleano para sinalizar uma atualização na lista de diretórios.
- **editDirId**: ID do diretório em edição.

### Funções

- **toggleDirForm**: Alterna a exibição do formulário de adição de diretório.
- **dirListChange**: Função para sinalizar uma mudança na lista de diretórios.
- **deleteDirectory**: Função assíncrona para deletar um diretório específico.

- **handleButtonClick**: Função para lidar com o clique nos botões de filtragem de tarefas.

## - **dirForm**

O componente **dirForm** é responsável por exibir um formulário para adicionar um novo diretório.

### Props

- **toggleDirForm**: Função para alternar a exibição do formulário de adição de diretório.
- **dirListChange**: Função para atualizar a lista de diretórios após adicionar um novo diretório.

### Estado

- **isLoading**: Booleano indicando se o formulário está carregando.
- **creationFailed**: Booleano indicando se a criação do diretório falhou.

### Efeitos

- **fetchDirectories**: Busca os diretórios do usuário.

## - **EditDirForm**

O componente **EditDirForm** é responsável por exibir um formulário para editar um diretório existente.

### Props

- **dirId**: ID do diretório a ser editado.
- **dirNome**: Nome atual do diretório a ser editado.
- **toggleEditForm**: Função para alternar a exibição do formulário de edição de diretório.
- **dirListChange**: Função para atualizar a lista de diretórios após editar um diretório.

### Estado

N/A

### Funções

N/A

## - **EditTaskForm**

O componente **EditTaskForm** é responsável por exibir um formulário para editar uma tarefa existente.



## Props

- **task**: Objeto contendo os detalhes da tarefa a ser editada.
- **closeForm**: Função para fechar o formulário de edição de tarefa.
- **taskListChange**: Função para atualizar a lista de tarefas após a edição.

## Estado

N/A

## Funções

N/A

## - MainContent

O componente **MainContent** é responsável por exibir as tarefas, lidar com operações de tarefas e gerenciar as funcionalidades de filtragem e busca de tarefas.

## Props

- **taskChanged**: Booleano indicando se a lista de tarefas foi alterada.
- **filter**: Número representando o filtro atual aplicado às tarefas.
- **setTodayTasks**: Função para definir as tarefas de hoje.
- **setTotalTasks**: Função para definir o número total de tarefas.
- **setTasksFinished**: Função para definir o número de tarefas concluídas.

## Estado

- **tarefas**: Lista de tarefas do usuário.
- **directories**: Objeto mapeando IDs de diretórios para nomes de diretórios.
- **error**: String de erro, se houver algum problema ao buscar tarefas.
- **showTaskForm**: Booleano indicando se o formulário de tarefa está sendo exibido.
- **editingTask**: Objeto da tarefa sendo editada, se houver uma tarefa em edição.
- **filteredTasks**: Lista de tarefas filtradas de acordo com o filtro e termo de busca.
- **searchTerm**: String do termo de busca inserido pelo usuário.

## Funções

- **toggleTaskForm**: Alterna a exibição do formulário de tarefa.
- **filterTasks**: Filtra as tarefas de acordo com o filtro selecionado e o termo de busca.

## - RightAside

O componente **RightAside** é responsável por exibir informações sobre o usuário, como o progresso das tarefas e as tarefas para hoje.

## Props

- **todayTasks**: Lista de tarefas para hoje.
- **totalTasks**: Número total de tarefas.

- **tasksFinished**: Número de tarefas concluídas.

## Estado

N/A

## Funções

- **deleteAccount**: Função assíncrona para deletar a conta do usuário.
- **logout**: Função para deslogar o usuário.

## - TaskForm

O componente **TaskForm** é responsável por exibir um formulário para adicionar uma nova tarefa.

## Props

- **toggleTaskForm**: Função para alternar a exibição do formulário de tarefa.
- **taskListChange**: Função para atualizar a lista de tarefas após adicionar uma nova tarefa.

## Estado

- **isLoading**: Booleano indicando se o formulário está carregando.
- **creationFailed**: Booleano indicando se a criação da tarefa falhou.
- **directories**: Lista de diretórios disponíveis para a nova tarefa.

## Efeitos

- **fetchDirectories**: Busca os diretórios do usuário.

## - Dashboard

O componente **Dashboard** é responsável por exibir o layout principal da aplicação, incluindo o menu lateral e o conteúdo principal.

## Props

N/A

## Estado

N/A

## Funções

N/A

## - CadastroForm

O componente **CadastroForm** é responsável por exibir um formulário para que os usuários possam se cadastrar na aplicação.

### Props

- **toggleForm**: Função para alternar entre o formulário de cadastro e o de login.
- **loginUser**: Função para enviar os dados de cadastro para a API.

### Estado

- **formData**: Objeto contendo os dados inseridos pelo usuário no formulário de cadastro.

### Funções

- **handleChange**: Função para atualizar o estado conforme o usuário digita no formulário.
- **handleSubmit**: Função para lidar com o envio do formulário de cadastro.

## - LoginForm

O componente **LoginForm** é responsável por exibir um formulário para que os usuários possam fazer login na aplicação.

### Props

- **toggleForm**: Função para alternar entre o formulário de cadastro e o de login.
- **loginUser**: Função para enviar os dados de login para a API.

### Estado

- **formData**: Objeto contendo os dados inseridos pelo usuário no formulário de login.

### Funções

- **handleChange**: Função para atualizar o estado conforme o usuário digita no formulário.
- **handleSubmit**: Função para lidar com o envio do formulário de login.

## - Home

O componente **Home** é responsável por exibir a página inicial da aplicação, contendo opções de login e cadastro.

### Props

N/A

### Estado

N/A

## Funções

N/A

## Códigos de Status

- **200 OK:** Requisição bem-sucedida.
- **401 Unauthorized:** Falha na autenticação do usuário.
- **404 Not Found:** Recurso não encontrado.

## Referências

- [Documentação do Entity Framework Core](#)
- [Documentação do React.js](#)
- [Documentação do .NET 6](#)