Documentação API:

Função para acessar o backend

```
async function requestData(endpoint, method = 'get', data = {}, withCredentials = false)
  try {
```

Endpoint: esse parâmetro é utilizado para acessar as rotas do backend. Todas as rotas fica em **/backend/routes**

Method: é o tipo da requisição que pode ser:

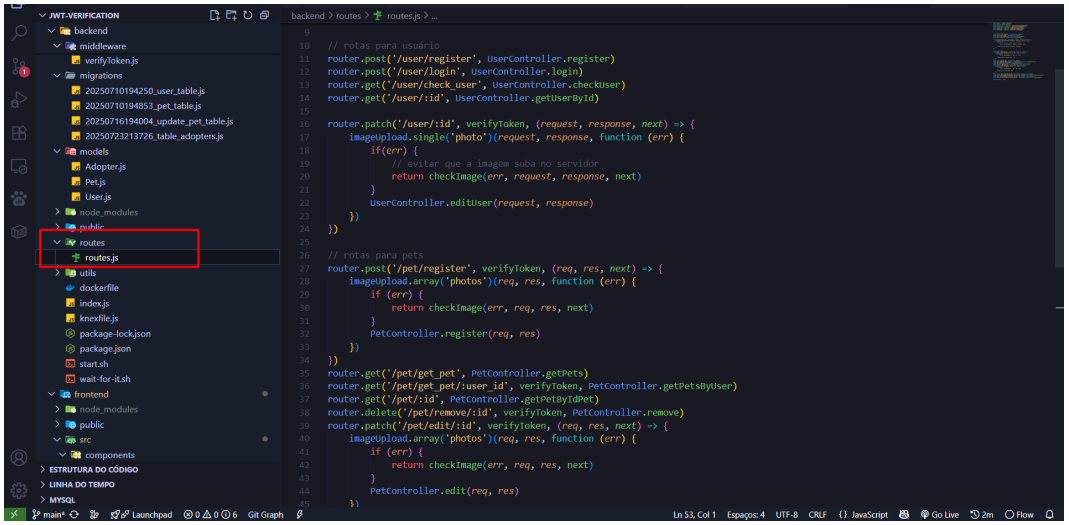
- **GET:** Usada para **buscar dados** de um servidor. Não altera nada, só consulta (ex.: pegar lista de usuários).
- **POST:** Usada para **criar novos dados** no servidor (ex.: cadastrar um usuário).
- **PUT:** Usada para **atualizar dados de forma completa**. Substitui o recurso inteiro (ex.: atualizar todas as infos de um usuário).
- **PATCH:** Usada para **atualizar parcialmente** um recurso (ex.: mudar só o email do usuário).
- **DELETE:** Usada para **excluir** um recurso do servidor (ex.: remover um usuário).

Data: formulário preenchido para ser mandado para o servidor (ex.: formulário de cadastro de usuário)

WithCredentials: você só ativa `withCredentials: true` quando precisa manter o usuário logado na comunicação frontend ↔ backend.

OBS: Os parâmetros de **data** e **withCredentials** são opcionais, vai ter rotas que não vai ser necessário.

arquivo de rotas:



Exemplos de uso:

OBS: Sempre colocar um await antes.

Buscar um usuário:

```
const response = await requestData(`/user/${user.id}`, 'GET', {}, true)
```

Registrar um usuário:

```
const response = await requestData("/user/register", "POST", userData, true)
```

Edição de usuário:

```
const response = await requestData(`/user/${user_id}`, "PATCH", formData, true)
```

Buscar uma sessão de usuário:

```
const response = await requestData("/user/session", "GET", {}, true)
```

Tratamentos de retornos:

Existem dois tipos de retornos, quando ocorre tudo certo ou quando acontece algum problema.

A função retorna alguns parâmetros:

```
    return {
      success: true,
      status: response.status,
      data: response.data,
      message: response.data.message
    }
  } catch (err) {
    if (err.response?.status === 401) {
      // dispara evento global
      window.dispatchEvent(new Event('SESSION_EXPIRED'))
    }
    return {
      success: false,
      status: err.response?.status || 500,
      message: err.response?.data?.message || err.message,
    }
  }
```

quando ocorre tudo certo ela retorna:
success: true ou false indicando se foi bem sucedido ou não.
status: são os status da requisição que pode ser 200, 500, 404, 502
data: é o retorno da informação vinda do banco (ex.: um usuário ou todos os pets)
message: é a mensagem de informativo vinda do banco para ser retornada para o usuário

quando dá algum erro:
success: sempre retorna false
status: são os status da requisição que pode ser 500, 404, 502 nunca retorna 200
obs: toda vez que o status é 401 significa que o usuário não está autenticado.
message: é a mensagem de informativo vinda do banco (ex.: erro ao encontrar usuário)

Exemplos de uso:

para uso só precisa usar um **if** e **else**

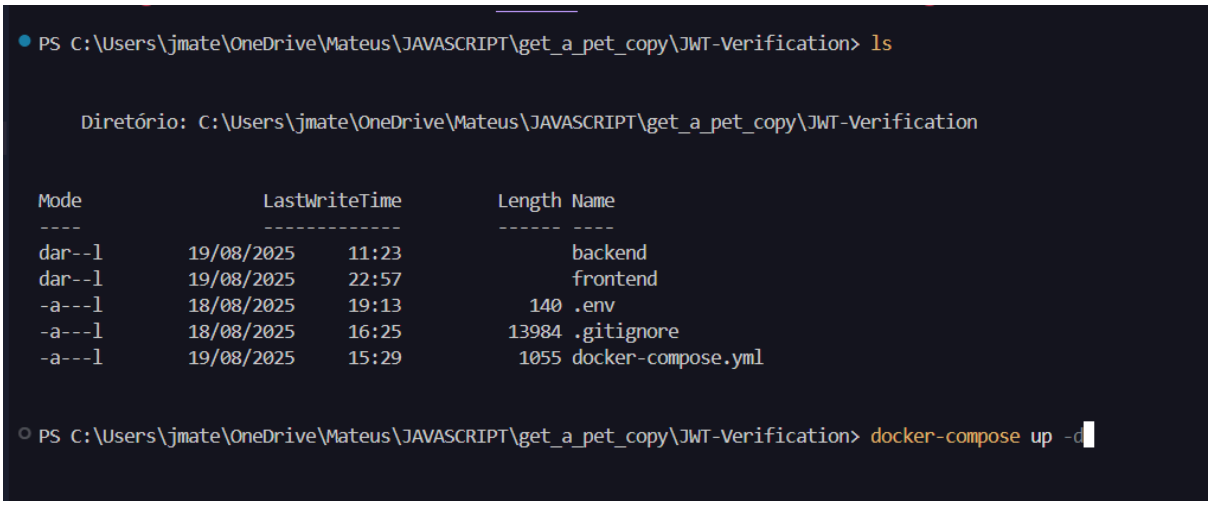
```
async function login(credentials) {
  const response = await requestData("/user/login", "POST", credentials, true)
  if (response.success) {
    setAuthenticated(true)
    setUser(response.data.user)
    setFlashMessage(response.data.message, "success")
    navigate("/")
  } else {
    setFlashMessage(response.message, "error")
  }
  return response
}

async function register(userData) {
  const response = await requestData("/user/register", "POST", userData, true)
  if (response.success) {
    setAuthenticated(true)
    setUser(response.data.user)
    setFlashMessage(response.data.message, "success")
    navigate("/")
  } else {
    setFlashMessage(response.message, "error")
  }
  return response
}
```

```
// busca sessão do usuário
useEffect(() => {
  async function fetchSession() {
    const response = await requestData("/user/session", "GET", {}, true)
    if (response.success) {
      setUser(response.data.user)
    } else {
      setUser(null)
    }
  }
  fetchSession()
}, [])
```

Docker:

OBS: Todos os comandos devem ser iniciados na raiz da aplicação.



Rodar o projeto primeira vez:

- docker-compose up --build

Parar os containers:

- docker-compose stop

Iniciar sem fazer build:

- docker-compose start

Iniciar com build:

- docker compose up -d

Derrubar os containers sem apagar o banco:

- docker-compose down

Apagar Todo o banco:

- docker-compose down -v

Logs da aplicação:

- docker compose logs -f backend
- docker compose logs -f frontend
- docker compose logs -f db

Consultar o banco de dados:

- docker compose exec db psql -U postgres -d nome_do_banco
- \dt

Formatação de tabelas no banco:

- **\x auto** (as tabelas organizar automaticamente)
- **\pset border 2** (colocar bordas nas tabelas)

Acessar a aplicação:

- http://localhost:5173/