🚀 reactrack

O projeto Open Source em ReactJS + TypeScript

📌 Sobre o Projeto

O reactrack é um ecossistema Open Source desenvolvido com a stack MERN, criado para funcionar como uma vitrine de projetos em ReactJS + TypeScript.

Ele atende tanto:

- Desenvolvedores iniciantes, que desejam aprender com projetos reais
- Desenvolvedores experientes, que querem contribuir via Pull Requests

Cada sistema é independente, mas integrado a um grande hub — formando um verdadeiro “sistema de sistemas”.

🎯 Objetivos

- Criar uma vitrine real de projetos React + TypeScript
- Estimular contribuições Open Source via PRs
- Servir como material de estudo prático
- Centralizar múltiplos sistemas em um único ecossistema
- Compartilhar boas práticas de frontend e backend

🏗️ Arquitetura Geral (MERN)

- Frontend: ReactJS + TypeScript + Vite
- Backend: Node.js + Express
- Banco de Dados: MongoDB (Mongoose) + JSON DBs

- Estado & Cache: TanStack Query
- Autenticação: JWT + Cookies HTTP Only
- Infra: Firebase, Cloudinary, Nodemailer

📦 Estrutura do Projeto

O reactrack é organizado de forma modular, separando claramente responsabilidades entre frontend (client), backend (server) e recursos auxiliares.

🎨 Frontend — client

Frontend desenvolvido com Vite + React + TypeScript, focado em performance, componentização e escalabilidade.

📁 Estrutura de Pastas

ui-ux

- Imagens e referências visuais dos sistemas desenvolvidos no reactrack.

public

- Imagens públicas do projeto
- Arquivos CSS utilizados no SPA

src

Pasta principal com todo o setup e funcionamento da aplicação.

- **main.tsx**
  Importação do CSS global, criação da árvore de componentes, configuração do:
  - Context Provider principal
  - TanStack Query Provider

    - React Router DOM

- **index.css**
  Estilos globais do projeto.

- **routes/index.tsx**
  Definição das rotas usando createBrowserRouter, com paths e componentes funcionais.

- **App.tsx**
  Renderiza o container de toasts e o Outlet, responsável por carregar os componentes de cada rota.

- **App.css**
  Estilizações específicas do sistema Fit.

assets

- Recursos visuais como imagens e ícones do projeto.

🌐 Camada de API — api

Responsável por toda a comunicação com o backend e serviços externos.

authState

- Função que verifica se o usuário está autenticado para acessar o sistema.

interfaces

- **requests**
  Interfaces das requisições:
    - get: backendUrl
    - post: login, register, resetPassword, verify, verifyEmail
- **responses**
  Interfaces de resposta:
    - get: userData, message

    - post: message

Funções principais

- login – autenticação com email e password
- logout – desloga o usuário via POST
- register – cadastro com name, email e password
- resetOtp – envio de email para reset de senha
- resetPassword – redefinição de senha com OTP
- verifyAccount – validação do usuário via OTP
- verifyEmail – confirmação de email
- userData – retorno dos dados do usuário logado

projects

- Interfaces do retorno da API do GitHub para o sistema Projects.

urls

- Endereços dos serviços utilizados pelo reactrack.

talkive

- **config**
  Configurações do Cloudinary e Firebase Firestore (signup, login, logout e resetPassword).

- **lib**
  Upload de imagens usando Cloudinary.

🧰 Utilitários — utils

- **systems**
  Funções utilitárias globais, incluindo o queryClient do TanStack Query.

- **commonAxios.ts**
  Instância comum do Axios para tratamento de status HTTP e exibição de toasts.

🧩 Estrutura Interna

interfaces

- Interfaces de respostas de API e componentes HTML customizados.

models

- Models de sistemas, como o Convene, utilizando classes.

components

- Componentes reutilizáveis e componentes específicos de cada página.
- Inclui elementos como Navbar.

hooks

- **backend –** obtém o backendUrl via AppContext
- **routes –** abstração do useNavigate
- **states –** gerenciamento de estados de login, signup e reset de senha

context

- Contextos dos sistemas:
    - Coin
    - Opinly

    - Talkive / TalkNet

- Inclui também o AppContext.

📄 Páginas — pages

- **Auth**
  Home, Login, Signup, EmailVerify, ResetPassword

- **Convene**
  Sistema com JSON DB e TanStack Query

- **Crypto**
  Consumo da API CoinGecko com detalhes de moedas

- **Fit**
  Integração com ExerciseDB e MuscleWiki

- **Investments**
  Entrada de dados e exibição com Google Charts e tabelas customizadas

- **Movies**
  Consumo da API TMDB

- **Opinly**
  Sistema com JSON DB e hooks modernos do React

- **Projects**
  Integração com a API do GitHub

- **Talkive**
  Chat em tempo real com Firebase e Cloudinary

- **SystemsLayout**
  Layout padrão com Navbar, Header e Outlet

🛠️ Backend — server

Backend estruturado em Node.js + Express + TypeScript, seguindo o padrão MVC.

📁 Estrutura

- **package.json**
  Scripts do servidor, dependências e devDependencies (Type Module).

- **tsconfig.json**
  Configuração do compilador TypeScript.

- **.env.local**
  Variáveis de ambiente:
    - Porta
    - MongoDB URL
    - JWT Token
    - SMTP credentials
    - Sender email

- **server.ts**
  Inicialização do Express, conexão com o banco e middlewares globais.

🔐 **Middlewares — middleware**

- **userAuth.ts**
  Gerencia autenticação JWT e injeta userId no request.

routes

- **auth.ts**
  Endpoints de:
    - register, login, logout
    - verifyEmail, sendVerifyOtp
    - sendResetOtp, resetPassword
    - isAuthenticated

- **home.ts –** retorna HTML com status 200

- **error.ts –** retorna HTML com status 404
- **user.ts –** retorna dados do usuário autenticado

🧠 Controllers

- Gerenciam cookies JWT
- Validações
- Transporte de emails
- Retorno de status e dados da API

🗄️ Outras Pastas

- **config –** conexão MongoDB com Mongoose
- **dist –** arquivos transpilados para JS
- **interfaces –** padronização dos dados do server
- **json –** bancos JSON (Convene e Opinly)
- **models –** schema de usuário com Mongoose
- **public –** imagens, CSS e JS públicos
- **systems –** scripts JS para sincronização via Nodemon
- **templates –** templates de email (Nodemailer + Brevo)
- **views –** páginas HTML de home e error

🤝 Contribuições

O **reactrack** é **Open Source** e está aberto para contribuições da comunidade!
Você pode criar novos sistemas, melhorar os existentes ou aprender explorando o código.

🚀 Como contribuir

1. Fork o repositório
2. Crie uma branch para sua feature
3. Implemente melhorias ou novos sistemas
4. Abra um Pull Request

📣 Conclusão

reactrack não é apenas um projeto.

É um **hub de aprendizado, colaboração e crescimento** para desenvolvedores **React + TypeScript** 💙⚛️

Explicação mais detalhada para programadores

Folders

ui-ux: imagens dos sistemas feitos no reactrack.

client:

configuração de inicialização Vite com React + TS.

public: imagens públicas do projeto para o SPA e CSSs.

src: pasta main com o setup do projeto e seu funcionamento.

main.tsx: arquivo com importação do CSS global, criação da árvore de componentes, uso do provedor de contexto principal, provedor de query com react-query e provedor de rotas com react-router-dom.

index.css: arquivo CSS global do projeto com propriedades gerais.

routes/index.tsx: arquivo das rotas com createBrowserRouter, contendo paths e elements para FC.

App.tsx: componente que renderiza o container de toasts e Outlet para lidar com os componentes de cada path.

App.css: arquivo CSS para lidar com as estilizações do sistema ‘Fit’.

assets: recursos de imagens e ícones para o projeto.

api:

authState: função para verificar se o usuário está autenticado a acessar o sistema.

interfaces:

requests: interfaces de requisições com get (backendUrl) e post (login, register, resetPassword, verify e verifyEmail).

responses: interfaces de respostas com get (userData e message) e post (message).

login: função que recebe o backendUrl e um payload de objeto com email e password, logando no sistema.

logout: função que recebe o backendUrl e desloga da conta com POST.

projects: interfaces com o retorno da API do GitHub acerca do ‘Projects’ system.

register: função que recebe o backendUrl e um payload de objeto com name, email e password, cadastrando no sistema.

resetOtp: função que recebe o backendUrl e um payload de objeto com email, passando para uma próxima tela do sistema.

resetPassword: função que recebe o backendUrl e um payload de objeto com email, otp e newPassword, resetando a nova senha do usuário.

userData: função que recebe o backendUrl e retorna os dados do usuário, usados na página Home.

verifyAccount: função que recebe o backendUrl e um payload de objeto com otp, autenticando o usuário para acessar os sistemas do ‘reactrack’.

verifyEmail: função que recebe o backendUrl e um payload de objeto com email, mudando para uma tela que ele insere o código otp recebido no email.

urls: arquivos com os endereços dos serviços usados pelo ‘reactrack’.

talkive:

config: configurações do serviço Cloudinary e Firebase Firestore com módulos de signup, login, logout e resetPass para sistema ‘Talkive/TalkNet’.

lib: funcionalidade de upload de imagens no sistema ‘Talkive/TalkNet’ usando Cloudinary.

utils:

systems: funções utilitárias para sistemas do ‘reactrack’, incluindo ‘queryClient’ do ‘tanstack-query’.

commonAxios.ts: função common com axios para validar ‘status’ sem ter erro do server, mostrando os toasts.

interfaces: pastas com interfaces de respostas de API e componentes HTML customizados em componentes.

models: pasta com models dos sistemas, como o sistema ‘Convene’ com suas classes.

components: pastas com os componentes de cada página do sistema e elementos HTML customizáveis, como Navbar.

hooks:

backend: hook customizado para get de backendUrl a partir do context do AppContext.

routes: hook customizado para uso do hook do ‘react-router-dom’ (useNavigate), navegando entre as páginas.

states: hook customizado para administrar states de login e signUp, bem como, hook personalizado de email e newPassword para resetar senha.

context: contextos de sistemas (Coin, Opinly e Talkive/TalkNet) com Context e ContextProvider, tal qual com AppContext, para retorno dos dados do usuário.

pages:

Auth: páginas que se comunicam com o server: Home, Login, Signup, EmailVerify, ResetPassword.

Convene: sistema com banco de dados JSON usando ‘tanstack-query’ para otimização de requisições. Nesta pasta, contém o Provider e as Sections.

Crypto: sistema com chamada da API coingecko gratuitamente, analisando os índices de criptomoedas. Nesta pasta, contém o Provider, bem como, a Crypto e a CoinDetail.

Fit: sistema com chamada da API exerciseDB gratuitamente e recursos do muscleWiki. Nesta pasta, contém o Fit e o ExerciseDetail.

Investments: sistema com componentização para entrada de dados e saída em gráfico do googleCharts, tal qual, tabela customizada. Nesta pasta, contém o Provider e o Investments.

Movies: sistema com chamada da API tmdb com chave de acesso. Nesta pasta, contém o Provider e as Sections.

Opinly: sistema com banco de dados JSON usando fs e hooks novos do reactJS. Nesta pasta, contém o Provider e o Opinly.

Projects: sistema com chamada da API do GitHub gratuitamente, analisando o perfil de usuário. Nesta pasta, contém o Provider e o HomeProjects.

Talkive: sistema com integração Firebase e Cloudinary para chat em tempo real. Nesta pasta, contém o Provider e as Sections.

SystemsLayout: layout configurado para navbar, header e Outlet dos sistemas do ‘reactrack’.

server:

package.json: JSON com script para rodar servidor e serviços de banco de dados JSON. Type Module, dependências e dev dependências.

tsconfig.json: JSON com configurações do transpilador, target, rootDir e outDir.

.env.local: configurações de ambiente com porta, url do MongoDB, token JWT, usuário SMTP, senha SMTP e Sender email.

server.ts: arquivo que instancia o express, conecta ao banco de dados, define configurações do framework e seus respectivos middlewares.

middleware:

userAuth.ts: middleware que gerencia o token JWT e fornece a propriedade userId com o id decodificado do token.

routes:

auth.ts: endpoints que garantem o funcionamento das funções de register, login, logout, sendVerifyOtp, verifyEmail, isAuthenticated, sendResetOtp e resetPassword.

error.ts: middleware que garante o envio de um arquivo html e status 404.

home.ts: middleware que garante o envio de um arquivo html e status 200.

user.ts: endpoint que retorna os dados do usuário ativo cadastrado no banco de dados.

controllers: intermediários que realizam o set do token nos cookies, permitindo o login e cadastro, desloga o usuário, limpando o cookie (logout), bem como faz as validações, retornam os status/dados e realiza o transporte de email.

config: configuração mongoDB usando mongoose para conexão ao banco de dados.

dist: arquivos em JS que foram transpilados pelo TS, mantendo a hierarquia de pastas e sendo usados pelos scripts do package.json.

interfaces: padronização dos dados usados no server do ‘reactrack’.

json: bancos de dados JSON para sistemas ‘Convene’ e ‘Opinly’.

models: modelo de user (schema) para ser utilizado pelo mongoose facilitando as querys com estilo de ORM.

public: pasta com imagens, css e js a ser acessado pelo html e JSON.

systems: sistemas definidos em JS para ‘Convene’ e ‘Opinly’, usando scripts para sincronização em nodemoon.

templates: template de email e configuração nodemailer para transporte dos emails usando brevo.

views: páginas HTML de home e error usadas pelos middlewares de index e undefined.