VideoShare Mobile

Aplicativo desenvolvido com Expo e React Native, utilizando o roteador expo-router e a navegação do ecossistema React Navigation.

Link para esse arquivo no github

Link para apresentação do projeto no youtube

Apresentação

Este repositório contém um trabalho acadêmico da disciplina 28743 - DESENVOLVIMENTO MOBILE.

- Aluno(a): Vitor Mateus Weirich (weirichvitor@gmail.com)
- Turma: EAD54-12
- Professor(a): Alysson Oliveira
- Disciplina: 28743 - DESENVOLVIMENTO MOBILE

Documento de requisitos e protótipos (wireframes/mockups):

- Caminho no repositório: `layout (Protótipo)/Requisitos e Propósta de Layout (Protótipo).md`
- Link direto (GitHub): https://github.com/vitorweirich/video-share-mobile/blob/master/layout%20(Protótipo)/Requisitos%20e%20Propósta%20de%20Layout%20(Protótipo).md

Principais requisitos funcionais definidos no documento:

- Autenticação (Login e Cadastro)
- Gerenciamento de vídeos: Meus Vídeos (listagem), Visualização de Vídeo e Envio de Vídeo

Estrutura básica do projeto

Pastas e arquivos principais:

- `app/` — Páginas (file-based routing via expo-router)
  - `login.tsx` — Tela de Login
  - `cadastro.tsx` — Tela de Cadastro
  - `(tabs)/_layout.tsx` — Layout das abas
  - `(tabs)/videos/index.tsx` — Tela Meus Vídeos (listagem)
  - `(tabs)/upload/index.tsx` — Tela de Envio de Vídeo
  - `video/[id].tsx` — Tela de Visualização de Vídeo (detalhes/reprodução)
  - `_layout.tsx`, `+not-found.tsx` — Arquivos de layout e fallback
- `components/` — Componentes reutilizáveis de UI
- `contexts/AuthContext.tsx` — Contexto de autenticação
- `store/videos.tsx` — Estado/armazenamento local de vídeos
- `constants/api.ts` — Configurações auxiliares (endpoints, etc.)
- `assets/` — Imagens, fontes e ícones

- `layout (Protótipo)/` — Protótipos e documento de requisitos

Telas previstas/implementadas:

- Login (`app/login.tsx`)
- Cadastro (`app/cadastro.tsx`)
- Meus Vídeos / Listagem (`app/(tabs)/videos/index.tsx`)
- Envio de Vídeo (`app/(tabs)/upload/index.tsx`)
- Visualização de Vídeo (`app/video/[id].tsx`)

Dependências principais (parcial):

- Expo e ecossistema: `expo`, `expo-router`, `expo-splash-screen`, `expo-status-bar`, `expo-constants`, `expo-font`, `expo-blur`, `expo-haptics`, `expo-linking`, `expo-symbols`
- Navegação: `@react-navigation/native`, `@react-navigation/bottom-tabs`, `@react-navigation/elements`, `react-native-screens`, `react-native-safe-area-context`
- Gestos/Animações: `react-native-gesture-handler`, `react-native-reanimated`
- Vídeo e arquivos: `expo-av` (reprodução de vídeo), `expo-document-picker` (seleção de arquivos)
- Utilidades: `@react-native-async-storage/async-storage`, `nanoid`
- UI/Ícones: `@expo/vector-icons`

Scripts úteis (package.json):

- `yarn start` — Inicia o servidor do Expo
- `yarn android` — Executa no Android (build nativo via Expo)
- `yarn ios` — Executa no iOS (requer macOS/Xcode)
- `yarn web` — Executa no navegador
- `yarn lint` — Linting com ESLint/Expo

Como executar

1. Instale as dependências: `yarn` (ou `npm install`)
2. Inicie o app: `yarn start` (ou `npx expo start`)
3. Abra no emulador Android, iOS ou no Expo Go conforme sua preferência.

Este projeto usa file-based routing: as rotas são definidas pelos arquivos dentro da pasta `app/`.

Recursos úteis

- Documentação do Expo: https://docs.expo.dev/
- Guia do expo-router: https://docs.expo.dev/router/introduction/
- React Navigation: https://reactnavigation.org/

Integração com o Backend

Esta seção descreve como o app consome as APIs do backend e como a autenticação e o fluxo de upload/reprodução de vídeos funcionam.

Configuração da URL base

- O arquivo `constants/api.ts` define a URL base da API em `API_URL`.

- Por padrão, usa `https://native.videos.vitorweirich.com`.
- É possível sobrescrever em tempo de build/execução definindo a variável de ambiente `EXPO_PUBLIC_API_URL` (qualquer valor `EXPO_PUBLIC_*` fica disponível no bundle do app). Ex.: para apontar para um backend local, defina `EXPO_PUBLIC_API_URL=http://10.0.2.2:8080` (Android Emulator).
- O objeto `jsonHeaders` centraliza os headers JSON e inclui `X-Http-Only: false` para que o backend retorne os tokens no corpo da resposta ao invés de cookies HttpOnly.

Arquivos:

- `constants/api.ts` — `API_URL` e `jsonHeaders`.

Autenticação e tokens

Implementado em `contexts/AuthContext.tsx`.

- Login: `POST /v1/api/auth/login` com `{ email, password }`.
  - Espera receber `{ accessToken, refreshToken }` no corpo da resposta.
  - Tokens são persistidos no AsyncStorage e adicionados como `Authorization: Bearer <accessToken>` em todas as requisições autenticadas.
  - Caso o backend retorne `{ token }` (fluxo de MFA), o app informa o usuário para finalizar a verificação em outro canal.
- Registro: `POST /v1/api/auth/register` com `{ name, email, password }` (envio de e-mail de confirmação).
- Perfil: `GET /v1/api/auth/me` retorna dados do usuário logado; usado na inicialização e após o login para popular `user`.
- Refresh: `POST /v1/api/auth/refresh` com header `X-Refresh-Token`.
  - O app tenta automaticamente o refresh quando recebe `401/403` e repete a chamada original uma vez.
- Logout: `POST /v1/api/auth/logout` com `X-Refresh-Token` para invalidar o refresh token no backend; localmente, os tokens e o perfil são limpos do AsyncStorage.

Headers relevantes:

- `Authorization: Bearer <accessToken>` em endpoints protegidos.
- `X-Refresh-Token: <refreshToken>` para refresh/logout quando aplicável.

Gerenciamento de vídeos

Implementado em `store/videos.tsx` via contexto `VideosProvider`.

- Listar meus vídeos: `GET /v1/videos/me?rows=20`
  - Mapeia a resposta (Page) para `{ id, title, shareUrl, expiresIn }`.
- Upload de vídeo (assinado):
  1. Solicitar URL de upload: `POST /v1/videos/upload` com `{ fileName, fileSize, contentType }`.
     - Resposta esperada: `{ signedUrl, videoId, expirationDate }`.
  2. Enviar o arquivo binário para `signedUrl` via `PUT` (o app usa `expo-file-system` e `createUploadTask` para acompanhar progresso).

3. Registrar upload concluído: `PATCH /v1/videos/upload/{videoId}/register-uploaded` (204 esperado).
4. Atualizar listagem chamando novamente a API de listagem.

- Obter URL de reprodução: `GET /v1/videos/{id}`
  - Retorna um `signedUrl` temporário para reprodução/streaming no player (`expo-av`).

Tratamento de erros e segurança

- Falhas de rede/autorização exibem mensagens amigáveis; quando possível, o app tenta extrair `message` do corpo de erro.
- Tokens ficam em `AsyncStorage` e são usados somente para compor o header `Authorization`.
- O refresh é feito de forma transparente no helper `authFetch` (repetição automática uma vez após 401/403).
- Para uploads a partir de URIs `content://`, o arquivo é copiado para um caminho local `file://` antes do envio para garantir compatibilidade.

Referências de código

- `contexts/AuthContext.tsx` — fluxo de login, refresh, logout, `authFetch`.
- `store/videos.tsx` — listagem, solicitação de URL assinada, upload com progresso, registro de upload e obtenção de URL de playback.