Relatório do Projeto: Triolingo

Triolingo - Aprenda idiomas de forma divertida

Integrantes

Alexandre Medeiros Cavalcante

João Tiago Lima Carvalho

João Lucas Martins Silva Carvalho

Warney Rego Ferreira Filho

Marcus Vinicius Moura Lima

Objetivos da Aplicação

O objetivo principal do Triolingo é oferecer uma plataforma de aprendizado de idiomas interativa e gamificada, inspirada em funcionalidades de aplicativos populares como o Duolingo. A aplicação visa permitir que os usuários aprendam novos idiomas através de lições estruturadas, exercícios variados e acompanhem seu progresso, além de adicionar elementos de engajamento como ranking e um jogo de roleta.

Funcionalidades Implementadas

Com base nos arquivos fornecidos, as seguintes funcionalidades foram implementadas:

- **Autenticação de Usuários:**
  - Registro de novos usuários com nome, e-mail e senha.
  - Login de usuários existentes com e-mail e senha.
  - Logout de usuários.
  - Gerenciamento de token de autenticação via localStorage.
- **Gerenciamento de Perfil de Usuário:**
  - Obtenção de perfil do usuário (UID, e-mail, nome).
  - Seleção e atualização de idiomas alvo para aprendizado.
  - Garantia da existência de um documento do usuário no Firestore (verificação).
- **Sistema de Lições:**
  - Carregamento de dados de lições a partir de arquivos JSON locais.

    - Estrutura para diferentes tipos de exercícios:
        - Múltipla Escolha
        - Tradução
        - Preencher Lacunas
    - Contêiner de lição para gerenciar o fluxo dos exercícios, pontuação e progresso.
    - Comparação de respostas para exercícios de tradução ignorando acentos, maiúsculas/minúsculas e caracteres não alfabéticos.
- **Acompanhamento de Progresso:**
    - Salvamento do progresso da lição (ID da lição, idioma, pontuação, total de exercícios, percentual de conclusão, status de completude).
    - Obtenção do progresso de todas as lições do usuário ou filtrado por idioma.
    - Obtenção do progresso de uma lição específica.
    - Visualização do progresso recente no dashboard.
- **Gamificação:**
    - Sistema de Pontos: Obtenção de pontos do usuário.
    - Rankings:
        - Ranking Geral de usuários.
        - Ranking por Idioma.
    - Jogo de Roleta:
        - Permite ao usuário apostar pontos selecionando uma cor (vermelho ou preto).
        - Interface visual da roleta com animação de giro.
        - Histórico das últimas apostas.
- **Interface do Usuário (UI):**
    - Tema Claro/Escuro com persistência no localStorage e opção de recarregar a página ao trocar.
    - Componentes de UI reutilizáveis e estilizados (Badge, Button, Card, etc.).
    - Animações e transições de página utilizando Framer Motion.
    - Componente de Navbar responsivo com informações do usuário e botão de logout.
    - Layout global da aplicação configurado em `app/layout.tsx`.
    - Dashboard principal para visualização de idiomas selecionados, progresso recente e acesso aos rankings e jogos.
- **Configuração e Estrutura do Projeto:**
    - Utilização de Next.js com App Router.
    - TypeScript para tipagem estática.
    - Tailwind CSS para estilização, com configuração customizada de cores e temas.

- ESLint para linting de código.

Explicação Técnica do Código

- **`package.json` e `package-lock.json`**: Definem as dependências do projeto, scripts e metadados. As principais dependências incluem Next.js, React, Tailwind CSS, Firebase (cliente), Framer Motion e ESLint.

- **`next.config.js`**: Configurações específicas do Next.js, como a desativação da otimização de imagens e o gerenciamento de ESLint durante o build.

- **`tailwind.config.ts` e `postcss.config.js`**: Configurações do Tailwind CSS e PostCSS, definindo temas, cores customizadas, plugins e o uso do Autoprefixer.

- **`app/globals.css`**: Estilos globais da aplicação, incluindo a definição das variáveis de cores para os temas claro e escuro, e animações CSS.

- **`app/layout.tsx`**: Componente raiz do layout da aplicação. Envolve todos os filhos com `AuthProvider` (para gerenciamento de autenticação) e `ThemeProvider` (para gerenciamento de temas).

- **`app/page.tsx`**: Página inicial que redireciona o usuário para o `/dashboard` se autenticado, ou para `/login` caso contrário.

- **`app/register/page.tsx`, `app/login/page.tsx`, `app/language-selection/page.tsx`**: Páginas de registro, login e seleção de idioma, respectivamente. Utilizam os componentes de autenticação e seleção de idioma.

- **`app/dashboard/page.tsx`**: Página principal do dashboard do usuário. Exibe os idiomas que o usuário está aprendendo, progresso recente, e links para lições e rankings.

- **`app/dashboard/lesson/page.tsx`**: Página que renderiza uma lição específica, utilizando o `LessonContainer` para gerenciar a lógica da lição.

- **`app/dashboard/roulette/page.tsx`**: Página do jogo de roleta.

- **`app/dashboard/rankings/page.tsx`**: Página para exibir os rankings de usuários.

- **`hooks/useAuth.tsx`**: Hook customizado para gerenciar o estado de autenticação (usuário, status de carregamento, erros) e prover funções como `login`, `register`, `logout`, `updateLanguages`. Interage com as funções em `lib/firebase.ts`.

- **`hooks/use-toast.ts`**: Hook customizado para sistema de notificações (toasts).

- **`lib/firebase.ts`**: Módulo central para interações com a API backend. Contém funções para registro, login, obtenção de perfil, gerenciamento de idiomas, conclusão de lições, obtenção de progresso, rankings e apostas na roleta. Define a URL base da API.

- **`lib/lessons.ts`**: Funções para carregar dados de lições (de arquivos JSON em `public/data/`) e para salvar/obter o progresso das lições através da API.

- **`lib/utils.ts`**: Funções utilitárias, como `cn` (para mesclar classes Tailwind), formatação de datas e comparação de strings ignorando acentos e formatação.

- **`public/data/*.json` (e.g., `lesson1-en.json`)**: Arquivos JSON que contêm a estrutura e o conteúdo das lições para diferentes idiomas.

- **`components/ui/`**: Diretório contendo componentes de UI reutilizáveis como `Button`, `Card`, `Navbar`, `ThemeSwitcher`, `AnimatedBackground`, `MotionButton`, `MotionCard`, `MotionText`, `PageTransition`, `Rankings`, `Roulette`. Muitos destes componentes utilizam `framer-motion` para animações.

- **`components/lessons/`**: Componentes relacionados à estrutura das lições:
    - `LessonTypes.ts`: Define as interfaces TypeScript para lições e exercícios.
    - `LessonContainer.tsx`: Componente que orquestra a apresentação dos exercícios de uma lição, gerencia o estado da lição (exercício atual, pontuação) e lida com a finalização da lição.
    - `MultipleChoiceExercise.tsx`, `TranslateExercise.tsx`, `FillTheBlankExercise.tsx`: Componentes que renderizam os

diferentes tipos de exercícios, lidam com a interação do usuário e validam as respostas.

API Utilizada

A aplicação Triolingo utiliza uma **API backend customizada**, desenvolvida especificamente para o projeto, para gerenciar a lógica de negócios, autenticação de usuários, persistência de dados e funcionalidades interativas como o jogo de roleta e o sistema de ranking.

Tecnologias e Frameworks Principais da API:

- **Node.js com Express.js**: A API é construída sobre a plataforma Node.js, utilizando o framework Express.js para o roteamento e gerenciamento de requisições HTTP.
- **Firebase (Admin SDK e Client SDK)**:
    - **Firebase Authentication**: Utilizado para o registro, login e gerenciamento de sessões de usuários via e-mail e senha.
    - **Firestore**: Empregado como o banco de dados NoSQL principal para armazenar informações dos usuários (perfis, progresso, idiomas, pontos, estatísticas) e dados das lições.
    - **Firebase Realtime Database**: Usado para gerenciar o status online dos usuários em tempo real, embora a lógica principal de atualização de status no Firestore também exista.
- **dotenv**: Para gerenciar variáveis de ambiente, como credenciais do Firebase e a porta do servidor.
- **cors**: Middleware para habilitar o Cross-Origin Resource Sharing, permitindo que o frontend acesse a API.

Principais Funcionalidades e Endpoints da API:

A API é estruturada em torno de algumas rotas principais, cada uma lidando com um conjunto específico de funcionalidades:

1. **Autenticação (`/api/auth`):**
    a. `POST /register`: Registra um novo usuário, criando uma entrada no Firebase Authentication e um documento correspondente no Firestore com informações iniciais (nome, e-mail, UID, pontos, estatísticas).
    b. `POST /login`: Autentica um usuário existente, verificando as credenciais com o Firebase Authentication e retornando um token de ID.

Também verifica e, se necessário, cria ou atualiza o documento do usuário no Firestore.

c. `POST /logout`: Desconecta o usuário.

2. **Usuário (`/api/user`):**
    a. `GET /profile`: Retorna o perfil do usuário autenticado, incluindo UID, e-mail, nome e idiomas de interesse, buscando dados do Firestore.
    b. `GET /learning-languages`: Retorna a lista de idiomas que o usuário está aprendendo.
    c. `POST /languages`: Permite ao usuário atualizar a lista de idiomas alvo que deseja aprender, salvando essa informação no Firestore.
    d. `GET /check-firestore`: Verifica se um usuário autenticado possui um documento no Firestore e o cria/atualiza se necessário, garantindo a consistência dos dados do perfil.
    e. `GET /rankings/general`: Retorna um ranking geral dos usuários baseado em seus pontos totais.
    f. `GET /rankings/language/:language`: Retorna um ranking de usuários filtrado por um idioma específico, baseado nas estatísticas de desempenho naquele idioma.
    g. `POST /online-status`: Atualiza o status online do usuário e o último horário de atividade no Firestore.
    h. `GET /points`: Retorna a quantidade atual de pontos do usuário.
3. **Lições (`/api/lessons`):**
    a. `POST /progress`: Salva o progresso de uma lição específica para o usuário (pontuação, total de exercícios, percentual de conclusão). Atualiza o registro se a nova pontuação for maior.
    b. `POST /complete`: Marca uma lição como finalizada, calcula os pontos ganhos e atualiza as estatísticas do usuário e o progresso da lição no Firestore. A conclusão é considerada se o percentual for de 60% ou mais. Utiliza transações Firestore para garantir a atomicidade das atualizações de progresso e pontos do usuário.
    c. `GET /progress`: Retorna o progresso de todas as lições do usuário, podendo ser filtrado por idioma.
    d. `GET /recent-progress`: Obtém o progresso mais recente das lições do usuário, ordenado por data de atualização.
    e. `GET /progress/:lessonId`: Retorna o progresso de uma lição específica para um determinado idioma.
4. **Roleta (`/api/roulette`):**
    a. `POST /bet`: Permite ao usuário autenticado apostar uma quantidade de seus pontos em uma cor (vermelho ou preto). O resultado é gerado aleatoriamente, e os pontos do usuário são atualizados no Firestore, juntamente com estatísticas do jogo de roleta.

b. `GET /history`: Retorna as estatísticas do jogo de roleta para o usuário (total de apostas, vitórias, perdas, pontos ganhos/perdidos) e seus pontos atuais.

Autenticação e Autorização na API:

- A API utiliza tokens **JWT (JSON Web Tokens)** gerados pelo Firebase Authentication para proteger suas rotas.
- O frontend envia o token no cabeçalho `Authorization` como um "Bearer token" em requisições para endpoints protegidos.
- Um **middleware de autenticação** (`middleware/auth.js`) na API intercepta essas requisições, verifica a validade do token usando o Firebase Admin SDK e, em caso de sucesso, anexa as informações do usuário decodificadas (`req.user`) ao objeto da requisição, tornando-as disponíveis para as rotas subsequentes.
- Em ambiente de desenvolvimento, há uma contingência para decodificar manualmente o token ou usar um usuário padrão caso a verificação com o Admin SDK falhe, facilitando testes locais.

Configuração e Credenciais da API:

- A API é configurada para usar o Firebase Admin SDK para interações privilegiadas com os serviços Firebase, como a verificação de tokens e acesso direto ao Firestore/Realtime Database.
- As credenciais do Firebase Admin SDK são obtidas a partir de um arquivo JSON (`service-account.json`) localizado na pasta `api/` do projeto. Este arquivo contém informações sensíveis como `project_id`, `private_key_id`, `private_key`, e `client_email`, permitindo acesso administrativo ao projeto Firebase "triolingo-59713".
- O arquivo `config/firebase.js` é responsável por inicializar tanto o Firebase Client SDK (para operações como `createUserWithEmailAndPassword`) quanto o Firebase Admin SDK. Ele também inclui um adaptador para operações Firestore, permitindo alternar entre o SDK do cliente e o SDK Admin para essas operações, dependendo da variável de ambiente `USE_ADMIN_SDK`.
- Para hospedagem em plataformas como Vercel, é sugerido configurar as credenciais do Firebase como variáveis de ambiente (e.g., `FIREBASE_SERVICE_ACCOUNT`) em vez de usar o arquivo `service-account.json` diretamente. O `README.md` da API também orienta a configuração de variáveis de ambiente como `FIREBASE_CLIENT_EMAIL` e `FIREBASE_PRIVATE_KEY` em um arquivo `.env`.

A API, portanto, atua como uma camada intermediária crucial entre o frontend e os serviços do Firebase, encapsulando a lógica de negócios e garantindo a segurança e integridade dos dados.

Armazenamento Local: Tecnologia e Uso

A aplicação utiliza o `localStorage` do navegador para duas finalidades principais:

- **Token de Autenticação**: Após o login ou registro bem-sucedido, um token de autenticação (`auth_token`) é salvo no `localStorage`. Este token é subsequentemente recuperado e enviado no cabeçalho `Authorization` de requisições autenticadas à API. Ao fazer logout, o token é removido do `localStorage`.
- **Preferência de Tema**: A escolha do tema (claro ou escuro) pelo usuário é persistida no `localStorage` com a chave `theme`. Isso permite que a preferência do usuário seja mantida entre as sessões.

Considerações Finais

O projeto Triolingo culmina na entrega de uma aplicação web funcional e promissora para o aprendizado de idiomas, demonstrando a aplicação bem-sucedida de tecnologias modernas e um foco claro na experiência do usuário. A arquitetura, que combina Next.js para o frontend e uma API Node.js/Express com Firebase para o backend, provou ser robusta e escalável para as funcionalidades implementadas.

- **Pontos Fortes Reafirmados:**
    - **Estrutura Modular e Organizada:** A separação de responsabilidades entre o frontend (com seus hooks e componentes reutilizáveis) e a API backend facilitou o desenvolvimento e a manutenção.
    - **Experiência do Usuário (UX) Engajadora:** A gamificação (pontos, rankings, roleta) e os elementos interativos, como o feedback em tempo real nas lições e as animações com Framer Motion, contribuem significativamente para um ambiente de aprendizado mais dinâmico e motivador.
    - **Gerenciamento de Estado Eficaz:** A utilização de React Context e hooks customizados simplificou o gerenciamento do estado da aplicação, tanto global (autenticação, tema) quanto localmente nos componentes.
    - **Abrangência Funcional:** O sistema cobre desde o cadastro inicial do usuário, passando pela seleção de idiomas, realização de lições com

diferentes tipos de exercícios, até o acompanhamento de progresso e participação em rankings.
    - **Flexibilidade de Conteúdo das Lições:** A decisão de carregar o conteúdo das lições a partir de arquivos JSON externos (e.g., `lesson1-en.json`) permite fácil atualização e expansão do material didático sem necessidade de intervenção direta no código-fonte da aplicação.
- **Desafios e Aprendizados:**
    - **Integração Frontend-Backend:** Garantir a comunicação fluida e o tratamento de dados consistente entre o cliente Next.js e a API Node.js foi um desafio constante, mas crucial, que proporcionou aprendizado valioso sobre o design de APIs RESTful e o gerenciamento de estado assíncrono.
    - **Gerenciamento de Dependências e Configurações:** Lidar com as configurações do Firebase (tanto no cliente quanto no Admin SDK), Next.js, Tailwind CSS, e outras dependências exigiu atenção aos detalhes para garantir a compatibilidade e o funcionamento esperado em diferentes ambientes. A configuração do Firebase Admin SDK, especialmente com o `service-account.json`, foi um ponto crítico para a segurança e funcionalidade do backend.
    - **Consistência de Dados no Firestore:** A natureza NoSQL do Firestore e a necessidade de manter a consistência dos dados do usuário, como pontos e progresso em lições, levaram à utilização de transações Firestore na API, o que foi um aprendizado importante sobre operações atômicas.
- **Possíveis Evoluções e Trabalhos Futuros:**
    - **Expansão do Conteúdo:** Adicionar mais idiomas, níveis de dificuldade e uma variedade maior de tipos de exercícios (ex: reconhecimento de voz, formação de frases).
    - **Funcionalidades Sociais:** Implementar recursos como adicionar amigos, comparar progresso diretamente com eles, ou fóruns de discussão por idioma.
    - **Sistema de Notificações Avançado:** Além dos toasts, desenvolver um sistema de notificações para lembrar os usuários de praticar, informar sobre novas lições ou atualizações no ranking.
    - **Melhorias na API:**
        - **Paginação e Filtragem Avançada:** Para endpoints que retornam listas longas.
        - **Monitoramento e Logging:** Implementar ferramentas de monitoramento de performance e logging de erros mais robustas na API.

- **Segurança Aprimorada:** Avaliar a migração do armazenamento de tokens de autenticação de `localStorage` para cookies `HttpOnly` para mitigar riscos de XSS.
- **Testes Automatizados:** Desenvolver uma suíte de testes abrangente (unitários, integração, E2E).
- **Internacionalização (i18n) da Interface:** Permitir que a própria interface do Triolingo seja traduzida.
- **Conteúdo Gerado pelo Usuário:** Explorar a possibilidade de contribuições da comunidade.