Relatório Técnico do Projeto

Este documento contem todas as informações do projeto, onde abordaremos os seguintes tópicos:

- Estrutura do Projeto
- Arquitetura do Backend
- Arquitetura do Frontend
- Como Executar
- Como Acessar

Estrutura do Projeto

Para fins de entrega simplificada, o projeto foi consolidado em um único repositório. Em um cenário de produção, no entanto, a separação em repositórios distintos para backend e frontend é altamente recomendada para melhor organização e escalabilidade.

Na raiz do repositório, temos nosso arquivo docker-compose.yaml que ficará responsável por colocar nossa aplicação no **ar**. O **docker** é de extrema importância quando queremos garantir que nossa aplicação funcione, independente do servidor que estaremos usando.

O arquivo docker-compose.yaml, presente na raiz do repositório, orquestra a execução da aplicação em contêineres Docker. A utilização do Docker garante a portabilidade e consistência da aplicação, permitindo que ela seja executada de forma confiável em qualquer ambiente.

O **frontend** foi desenvolvido com **react.js** devido à sua capacidade de criar interfaces de usuário dinâmicas e reativas, proporcionando uma experiência moderna e responsiva. Além disso, a arquitetura baseada em componentes do React facilita a manutenção e escalabilidade do código.

O **backend** foi implementado em **node.js** com **express**, uma combinação que oferece alta performance e escalabilidade. **node.js**, com seu modelo de E/S não bloqueante, é ideal para aplicações em tempo real, enquanto o Express simplifica a criação de APIs RESTful.

Arquitetura do Backend

A arquitetura do backend é composta por três principais componentes: api-gateway, game-service, e user-service. Cada um desses componentes é um microsserviço independente, responsável por uma parte específica da aplicação. Abaixo está a explicação detalhada da estrutura de diretórios e arquivos de cada microsserviço e do API Gateway.

```
backend/
    api-gateway/
        Dockerfile
        package.json
```

```
        src/
            app.ts
            server.ts
        temp.env
        tsconfig.json
    game-service/
        .vscode/
            settings.json
        db/
            Dockerfile
            init.sql
            temp.env
        Dockerfile
        jest.config.ts
        package.json
        src/
            app.ts
            config/
                db.ts
            controllers/
                AccessController.ts
                PlayerController.ts
            middlewares/
                authMiddleware.ts
                authorizationMiddleware.ts
                errorMiddleware.ts
                validationMiddleware.ts
            models/
                AccessModel.ts
                playerModel.ts
                userModel.ts
            repositories/
                AccessRepository.ts
                PlayerRepository.ts
            routes/
                acessRoutes.ts
                playerRoutes.ts
            services/
                AccessService.ts
                PlayerService.ts
            utils/
                idGenerator.ts
                jwtUtils.ts
                passwordUtils.ts
                scoresCalculator.ts
        temp.env
        tsconfig.json
    user-service/
        .vscode/
            settings.json
        db/
```

```
            Dockerfile
            init.sql
            temp.env
        Dockerfile
        jest.config.ts
        package.json
        src/
            app.ts
            config/
                db.ts
            controllers/
                UserController.ts
            middlewares/
                authMiddleware.ts
                authorizationMiddleware.ts
                errorMiddleware.ts
                validationMiddleware.ts
            models/
                userModel.ts
            repositories/
                UserRepository.ts
            routes/
                userRoutes.ts
            services/
                UserService.ts
            utils/
                idGenerator.ts
                jwtUtils.ts
                passwordUtils.ts
        temp.env
        tsconfig.json
docker-compose.yaml
```

Vale salientar que devido a limitações de tempo, a realização de testes foi impactada, sendo possível realizar apenas alguns testes unitários. Entretanto caso haja tempo, toda a aplicação deve ser testada unitariamente e em sua integração.

API Gateway

O **api-gateway** é responsável por rotear as requisições para os microsserviços apropriados `(user-service e game-service)`. Ele utiliza o `http-proxy-middleware` para criar proxies para os microsserviços.

- Dockerfile: Define a imagem Docker para o API Gateway.
- package.json: Contém as dependências e scripts do projeto.
- src/app.ts: Configura as rotas de proxy para os microsserviços.
- src/server.ts: Inicia o servidor do API Gateway.

- temp.env: Arquivo de variáveis de ambiente. Copie o conteúdo dele em um arquivo `.env` ao lado do `temp.env`
- tsconfig.json: Configurações do TypeScript para o projeto. O Typescript colabora no desenvolvimento, ajudando a prefinir erros e garantindo consistência nos dados.

Game Service

O `game-service` é responsável por gerenciar os jogadores e seus acessos. Ele utiliza o `express` para criar uma API RESTful.

- .vscode/settings.json: Configurações do VSCode.
- db/Dockerfile: Define a imagem Docker para o banco de dados PostgreSQL.
- db/init.sql: Script SQL para inicializar o banco de dados com as tabelas que serão usadas.
- db/temp.env: Arquivo de variáveis de ambiente para o banco de dados. Crie o arquivo `.env` ao lado deste arquivo e copie o conteúdo de `temp.env` para o arquivo `.env`
- Dockerfile: Define a imagem Docker para o game-service.
- jest.config.ts: Configurações do Jest para testes.
- package.json: Contém as dependências e scripts do projeto.
- src/app.ts: Configura a aplicação Express.
- src/config/db.ts: Configura a conexão com o banco de dados.
- src/controllers/: Contém os controladores que lidam com as requisições.
- src/middlewares/: Contém os middlewares para autenticação, autorização, validação e tratamento de erros.
- src/models/: Define os modelos de dados.
- src/repositories/: Contém os repositórios que interagem com o banco de dados.
- src/routes/: Define as rotas da API.
- src/services/: Contém a lógica de negócios.
- src/utils/: Contém utilitários como gerador de IDs, utilitários JWT, etc.
- temp.env: Arquivo de variáveis de ambiente. Crie o arquivo `.env` ao lado deste arquivo e copie o conteúdo de `temp.env` para o arquivo `.env`
- tsconfig.json: Configurações do TypeScript.

User Service

O `user-service` é responsável por gerenciar os usuários e a autenticação. Ele também utiliza o `express` para criar uma API RESTful.

- .vscode/settings.json: Configurações do VSCode.
- db/Dockerfile: Define a imagem Docker para o banco de dados PostgreSQL.
- db/init.sql: Script SQL para inicializar o banco de dados.
- db/temp.env: Arquivo de variáveis de ambiente para o banco de dados. Crie o arquivo `.env` ao lado deste arquivo e copie o conteúdo de `temp.env` para o arquivo `.env`
- Dockerfile: Define a imagem Docker para o user-service.
- jest.config.ts: Configurações do Jest para testes.
- package.json: Contém as dependências e scripts do projeto.

- src/app.ts: Configura o aplicativo Express.
- src/config/db.ts: Configura a conexão com o banco de dados.
- src/controllers/: Contém os controladores que lidam com as requisições.
- src/middlewares/: Contém os middlewares para autenticação, - autorização, validação e tratamento de erros.
- src/models/: Define os modelos de dados.
- src/repositories/: Contém os repositórios que interagem com o banco de dados.
- src/routes/: Define as rotas da API.
- src/services/: Contém a lógica de negócios.
- src/utils/: Contém utilitários como gerador de IDs, utilitários JWT, etc.
- temp.env: Arquivo de variáveis de ambiente. Crie o arquivo `.env` ao lado deste arquivo e copie o conteúdo de `temp.env` para o arquivo `.env`
- tsconfig.json: Configurações do TypeScript.

Arquitetura do Frontend

O `front` do projeto é construído utilizando React, uma biblioteca JavaScript para construção de interfaces de usuário. O projeto está configurado para usar TypeScript, proporcionando tipagem estática e melhorando a qualidade do código. A estrutura do projeto é organizada de forma modular, facilitando a manutenção e escalabilidade.

Foram usadas algumas ferramentas para o desenvolvimento, como:

- **Vite**: Utilizado para build e desenvolvimento rápido do projeto.
- **ESLint**: Configurado para garantir a qualidade do código através de linting.
- **TypeScript**: Proporciona tipagem estática, ajudando a evitar erros comuns e melhorar a manutenção do código.

Arquivos de Configuração e Metadados

- `.env.txt`: Arquivo de configuração de variáveis de ambiente.
- `.eslintrc.cjs`: Configuração do ESLint para linting do código.
- `.gitignore`: Arquivo para especificar quais arquivos e diretórios devem ser ignorados pelo Git.
- `Dockerfile`: Arquivo de configuração para criar uma imagem Docker do projeto.
- `Makefile`: Arquivo de automação de tarefas usando Make.
- `package.json`: Metadados do projeto e lista de dependências.
- `README.md`: Documentação do projeto.
- `tsconfig.json`: Configuração do TypeScript.
- `tsconfig.node.json`: Configuração adicional do TypeScript para Node.js.
- `vite.config.ts`: Configuração do Vite para build e desenvolvimento.

Diretórios

- `public/`: Arquivos públicos acessíveis diretamente.

    - `logo.png`: Imagem do logo do projeto.

- `src/`: Código-fonte do projeto.
    - `@types/`: Definições de tipos TypeScript.
        - `admin.d.ts`: Definições de tipos para administração.
        - `category.d.ts`: Definições de tipos para categorias.
        - `charge.d.ts`: Definições de tipos para cobranças.
        - `coin.d.ts`: Definições de tipos para moedas.
        - `cookies.d.ts`: Definições de tipos para cookies.
        - `order.d.ts`: Definições de tipos para pedidos.
        - `...`: Outros arquivos de definições de tipos.
    - `App.css`: Estilos globais do aplicativo.
    - `App.tsx`: Componente principal do aplicativo React.
    - `assets/`: Recursos estáticos como imagens e fontes.
    - `components/`: Componentes reutilizáveis do React.
    - `constants/`: Constantes usadas no projeto.
    - `data/`: Dados estáticos ou mockados.
    - `index.css`: Estilos globais adicionais.
    - `main.tsx`: Ponto de entrada principal do aplicativo React.
    - `mocks/`: Dados mockados para testes.
    - `pages/`: Componentes de páginas do React.
    - `redux/`: Configuração e estados do Redux.
    - `routers/`: Configurações de rotas do React Router.
    - `services/`: Serviços e APIs.
    - `utils/`: Funções utilitárias.
    - `vite-env.d.ts`: Definições de tipos específicas do Vite.

Outros Arquivos

- `index.html`: Arquivo HTML principal do projeto.

Como Executar

Para iniciar a aplicação, você deve inicialmente ir em cada microsserviço do backend e criar o arquivo `.env` tanto na raiz do microsserviço quanto na pasta `db`. Nestes diretórios teremos arquivos `temp.env` contendo os valores que devem ficar nos arquivos .env para que nossas aplicações funcionem corretamente.

Vale salientar que os arquivos `temp.env` nao existiriam, onde eles deveriam ficar restrito a zona de acesso apenas dos envolvidos no desenvolvimento.

Com o procedimento a cima feito, vamos usar o docker para colocar nossa aplicação no **ar**. Tenha o docker instalado em sua maquina.

Use o seguinte comando para iniciar a aplicação

```
docker compose up -d --build
```

Prontinho, basta aguardar um pouco e o projeto estará pronto para uso.

Como Acessar

Para acessar o `front` acesse waffle.alofan.com.br