**Consumo de APIs REST em Desenvolvimento Mobile**

**Nome do aluno:**Arthur Amorim Alves

**Tema do projeto:**

Programa de consumo de API Rest para um minigame de Pokémon

1. **Pesquisa Teórica**

- O que é uma API REST?

- Métodos HTTP (GET, POST, PUT, DELETE) e seus usos.

- Diferenças entre Fetch API e Axios.

- Como o Expo facilita o consumo de APIs (ex: suporte a fetch, bibliotecas de rede).

- Gerenciamento de estados assíncronos (loading, error, dados).

R:

- Uma API REST é uma forma de um site ou app conversar com outro sistema usando regras simples. Ela usa URLs e métodos HTTP (como GET e POST) para enviar e receber dados.

- GET: pega dados (ex: buscar lista de usuários).

POST: envia dados (ex: cadastrar novo usuário).

PUT: atualiza dados (ex: editar um usuário).

DELETE: apaga dados (ex: excluir um usuário).

- Em resumo o método Fetch API já é mais inclusa em navegadores, ele infelizmente não tem um retorno imediato em JSON sendo necessário o tratamento dos dados. Com .json o tratamento dos erros acaba se tornando mais problemático dependendo da situação sendo menos adaptativo pra isso. Já o Axios ele não está incluso diretamente nos navegadores sendo necessário uma instalação. Ele retorna automático em JSON e é muito mais prático o tratamento de erros em sua usabilidade.

- Expo já possui um suporte ao Fetch, além de permitir bibliotecas como Axios e React Query. Ao contrário de outros emuladores ele possui uma facilidade muito maior para iniciar e testar os projetos, fora o fato de ser prático a conexão é muito boa para o celular e etc.

- Loading, error, dados são os 3 estados mais importantes ao chamar uma API o loading refere-se ao carregamento do conteúdo da API, error é referente a caso de algum erro de requisição, conexão mal-sucedida e tals. Dados: e naus referente à o estado final de uma requisição de API, seria quando por exemplo os dados foram recebidos e mostrados na tela.

**Relatório Técnico**

- O processo de desenvolvimento foi algo meio novo, usado de ferramentas como a documentação das diversas ferramentas usadas foi algo lento devido alguns erros e por ser algo novo foi meio difícil aprender a lidar com algo desconhecido. Mas no geral a parte do front-end das telas, react em relação a estruturação visual foi bem simples e fácil de entender, sua semelhança com o css facilitou bastante no processo da construção do aplicativo e na criação da interface. Uma facilidade também foi no uso dos Links e de alguns elementos prontos como o radio. Praticamente foram bem simples e criação do index e do help

No quesito da parte de funções a questão da seleção foi o padrão de programador, onde foi necessário planejar como cada botão iria adquirir seu próprio estado e como cada um iria representar uma geração de pokemon de acordo como quando pressionasse-o. Algo também novo foi a utilização do router no caso useRoute() servindo como um redirecionamento de tela com a possibilidade de enviar um parâmetro (dados) para outra tela.

No play onde houve a maior parte das ações e construções do aplicativo, obviamente foi a tela mais demorada a ser feita e consequentemente a maior trabalhosa, as importações da inicialmente foram simples e estava tudo facilitado, percebido depois, a API se dividia muito dentro dela para um único pokemon, usando fetch e com a quantidade de JSON a ser trabalhado revelou uma estrutura um pouco poluída de certa forma e com muitas requisições ao mesmo tempo, o que pode até deixar um pouco lenta para se trabalhar, tendo em vista também a API ser um pouco grande no quesito das informações do pokemon, não servindo diretamente ou apropriadamente para tratamento de informações simples de cada pokemon.

**Desafios Encontrados**

O primeiro desafio encontrado foi na questão das importações iniciais do projeto e com alguns tutoriais e inconsistências nas documentações. Inicialmente acabei importante incorretamente e a estrutura estava bem diferente da qual as pessoas mostravam nos tutoriais e nas documentações e então para lidar com as routes acabou sendo bem confuso, mas depois que acabei “resetando” o projeto e troquei a estrutura para a importação correta ficou tudo mais facilitado. A dificuldade em relação aos botões de radio, foi algo bem bobo, no máximo o tempo para pensar em como fazer as atribuições e como funciona para a usabilidade do botão de quando fosse pressionado.

Na tela play foram onde as maiores problemáticas foram encontradas. Inicialmente tive alguns problemas para lidar com a situação de loading e para pegar dados com a API. Foi de certa forma simples, mas o problema foram com as outras informações e de como eu ia trata-las. A API se dividia muito dentro de si, o id do pokemon atribuído em uma tela não servia para o outro url, então meio que dentro de uma página como pokemon\_species um parâmetro de pokemon\_evolves ou algo relacionado tinha como objeto o url e dentro do url o endereço para a outra página da API que continua as evoluções porém ela não usava o id do pokemon definido em species e sim um outro id próprio então de certa forma acabou sendo muito confuso para mim. No código é possivel perceber eu usar muito também de console.log na parte da geração escolhida e etc, pois eu estava tendo um problema onde mesmo com a geração estando certa a API retornava um pokemon de uma geração que não fazia parte, e mais tarde acabei percebendo que apenas não havia atribuído a variável correta a o campo da url da API... Bom, para a questão do url eu precisei bastante passar horas estudando a estrutura e entender como funcionaria a parte de acessar cada propriedade, no meio do código no arquivo: parametros.txt lá eu coloquei todos os tipos de atributos que eu precisaria do pokemon e como chegar neles devido a imensa estrutura da API, fora isso também adicionei uma parte da API de evolução devido a muitos objetos dentro de cada um, aí usando as linhas do vscode eu podia perceber melhor qual elemento estava dentro de qual objeto e etc. Assim ficou mais fácil de lidar com a questão da evolução, visto que alguns pokemons podem ter ou não evoluções.

O problema mais chato que tive foi perto da finalização onde eu achei que tal parâmetro da API se tratava dos tipos, porém depois de um tempo analisei que se tratava dos egg\_group que é uma outra propriedade que em alguns aspectos se assemelhava bastante, porém ela estava em outra tela de importação da API e para piorar era conjunta em um array, o que me causou muita dor de cabeça no final. Na função descrita de comparar os tipos de pokemon eu usei o console.log diversas vezes para entender como estava sendo levado os valores para as funções já que não apresentavam êxito. Logo fiquei fazendo vários testes e vistorias para entender o porque, o motivo, razão ou circunstância aquilo não funciona. Depois de um tempo usando adoidadamente o console, usei a propriedade do typeof, afinal no console mostrava bem semelhante a string, porém ao usar o typeof percebi que estava em tipo objeto, foi quando entendi como eu estava tratando errado o objeto, estava tentando tratar ele como array, como string, até como numérico porém não havia percebido que havia deixado ele como tipo objeto, então fiz o tratamento dele usando o object.values tirando assim os valores de dentro do objeto e os convertendo em um array para percorrer eles de forma adequada, por fim assim finalizando o protótipo.