DAS

Propósito

O ponto central do problema é que a falta de informação de modo eficaz, preciso e facilitado, diminui o interesse geral dos leigos e interessados em excursões. A solução encontrada para o descrito problema é o desenvolvimento de um sistema acessível para o público por meio de um site que promove, avalia e informa a respeito das excursões.

Objetivos e Filosofias Arquiteturais

O sistema conterá uma integração contínua de software através das pipelines e serviços fornecidos pela plataforma GitHub. É muito importante que a configuração das pipelines estejam em ordem. Caso contrário, poderá trazer problemas de performance durante a entrega de uma release.

Como o sistema pode sofrer impactos de performance do servidor, é necessário um amplo tratamento de erro para toda a aplicação. Além disso, os tratamentos devem ser consistentes e padronizados.

Suposições e Dependências

O time de desenvolvimento deve conter programadores experientes e especialistas em UX. Os recursos usados pela aplicação devem ter alta disponibilidade para que o sistema não fique offline facilmente e nem frequentemente, portanto, serão usados servidores na nuvem. O sistema é totalmente novo, portanto não existem dependências de interfaces legadas.

Requisitos Arquiteturalmente Significantes

Serão criados dois ambientes para a aplicação, um ambiente para homologação e o outro, para produção. Os dois ambientes terão as mesmas configurações (sistema operacional, hardware, gerenciamento de dados e armazenamento).

Como os ambientes de produção e homologação devem ser padronizados, ou seja, vão performar de forma idêntica, será utilizado o Docker, uma ferramenta auxiliar capaz de executar instâncias da mesma aplicação em diferentes ambientes.

Decisões, Restrições e Justificativas

- O sistema deve ser performático e, portanto precisará de uma boa infraestrutura e ferramentas que sejam conhecidas pela sua performance boa;
- O sistema não deve conter muitas funcionalidades, apenas o necessário. Isso faz com que o uso seja mais simples para o usuário final, deixando mais inclusivo, não necessitando de grande custo de treinamento;
- As excursões não devem ter um botão para se inscrever, as pessoas podem decidir sobre ir ou não por conta própria, independente do sistema;
- A interface do usuário deve ser o mais simples possível.

Mecanismos Arquiteturais

Persistência

O projeto deve manter uma boa normalização quanto à persistência de dados. É importante conter diretórios específicos para o relacionamento e mapeamento dos dados. Além disso, um schema bem definido para cada tabela, juntamente com seus respectivos tratamentos.

Tratamento de Erros

Os tratamentos de erro devem ser padronizados em todo o projeto. As aplicações serão desenvolvidas sob o protocolo HTTP, portanto é necessário conter os códigos de status padronizados para cada modelo de requisição. Além disso, as respostas das requisições devem conter um objeto JSON normalizado.

Autenticação

A autenticação da aplicação será feita através de “bearer tokens” JWT (JSON Web Token). O token JWT provê todas as informações necessárias para autenticação e

reconhecimento dos usuários de forma performática e criptografada. O token é passado por meio do server para o client, onde será armazenado.

Abstrações Chaves

- CI/CD para uma entrega mais eficiente.
- SOLID para melhorar a qualidade do código.
- Clean Code para melhorar a qualidade do código.
- Servidores in Cloud visando a escalabilidade do produto.
- Git Flow para padronizar entre o time o controle de versionamento.
- Ambientes de Lab, STG e Prod.

Camadas ou Estruturas Arquiteturais

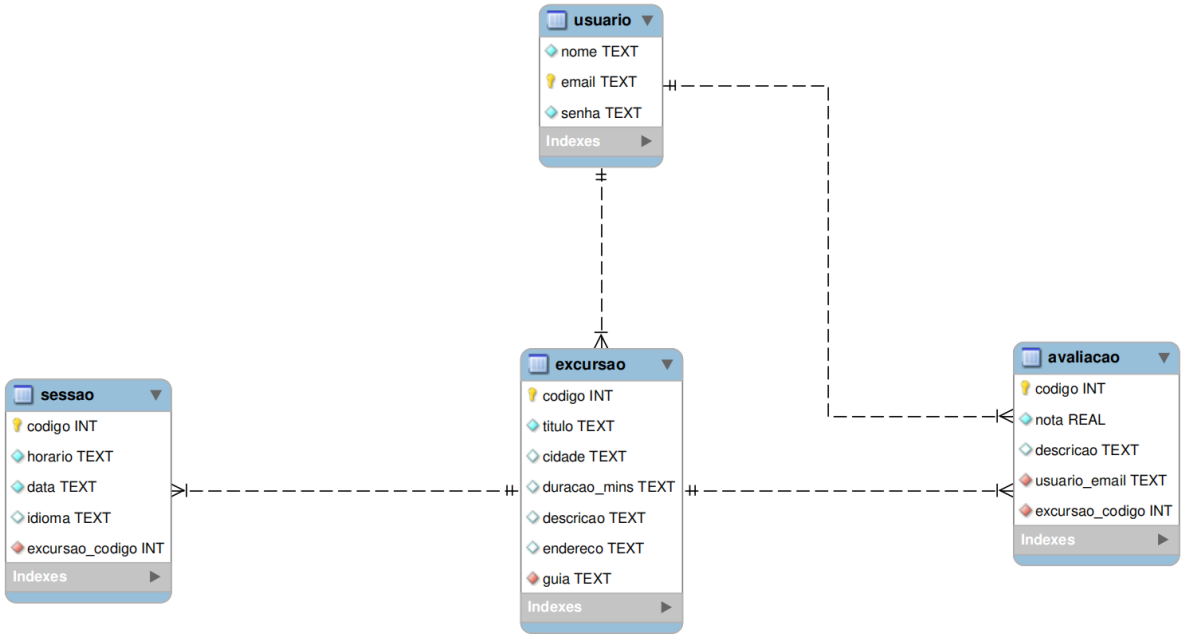
O sistema contará com um back-end desenvolvido utilizando arquitetura REST com Layer Framework. Já o front-end será desenvolvido utilizando o Atomic Design, Redux e Redux Thunk.

Visão Arquitetural

O sistema deverá contar com uma arquitetura que promova a manutenibilidade do software, com um padrão de fácil entendimento para novos desenvolvedores que entraram no projeto, sem replicação de código e que promova um código limpo.

Visões Recomendadas

- **Lógica:** O sistema contém apenas 4 classes principais, a de Usuário, Excursão, Sessão e Avaliação (de usuário). Cada uma terá uma tabela no banco de dados SQLite3 com relacionamento entre elas. Os relacionamentos entre as tabelas se darão seguindo o diagrama a seguir:



- **Operacional:** O sistema é composto por apenas um node, porém de forma que possa facilmente ser escalável, replicando os nodes e incluindo um load balancer.
- **Casos de uso:**
  1. O sistema deve garantir que os serviços não apresentem inconsistências;
  2. O sistema deverá tratar todos os formulários;
  3. O sistema deve emitir feedback aos usuários todas as vezes que um registro for
     alterado, editado ou descadastrado.