TECH CHALLENGE

MOBILE ENGINEER

CRITÉRIOS DE AVALIAÇÃO (I)

Para a posição de **Mobile Engineer** em Banking, nós esperamos que **seu código**

- Respeite as práticas de **SOLID** e **Clean Code**
- Faça uso de **Dependency Injection** (assistida ou não por frameworks)
- Exercite o código em suas múltiplas formas de uso/fluxo com **testes automatizados,** nos níves de unidade, integração e aceitação
- Trate o código de teste como first-class citizen, maximizando intenção e legibilidade (assistido ou não por frameworks)
- Faça o uso de **programação reativa funcional** (RxSwift / RxJava)
- Faça o uso de uma **arquitetura de responsabilidades** bem definida (MVC, MVVM, VIPER, etc)
- Faça o uso correto de um setup de **Continous Integration** e está pronto para Continuous Delivery
- Entregue uma **user interface otimizada**, e que faz uso das técnicas mais modernas relacionadas aos toolkits de UI nativos

CRITÉRIOS DE AVALIAÇÃO (II)

Para a posição de **Mobile Engineer** em Banking, nós esperamos que **você**

- Ofereça mecanismos para que nós identifiquemos sua lógica de divisão de tarefas, bem como de tracking de execução das mesmas (sugestão : Github projects)
- Faça o uso apropriado de Git, com commit messages significativas, commits com contribuições de tamanho adequado, uso compreensivo de branching e afins
- Documente algumas das principais decisões feitas no projeto
- Nos procure caso alguma coisa esteja esquisita e/ou não faça sentido

SUBMISSÃO

- Nós recomendados que sua solução seja **submetida como um projeto open-source no Github**. Caso você opte por manter seu projeto em um **repositório privado, nós forneceremos os usernames do Github** que serão os revisores do seu projeto

- Você deve enviar o link do Github e/ou solicitar os revisores apenas quando o trabalho estiver **100% concluído do seu lado**.

- **Não há prazo para a submissão** (*mas a vaga não estará aberta para sempre, rs*)

- Nós esperamos encontrar uma documentação mínima que descreva o básico que um desenvolvedor precisa saber para
  (a) instalar o projeto nas condições corretas e tê-lo *up-and-running*
  (b) rodar eventuais builds e/ou gerar artefatos pela CLI

DESCRIÇÃO

- Nesse desafio, você irá construir uma pequena aplicação que consome dados da API dos fatos do Chuck Norris

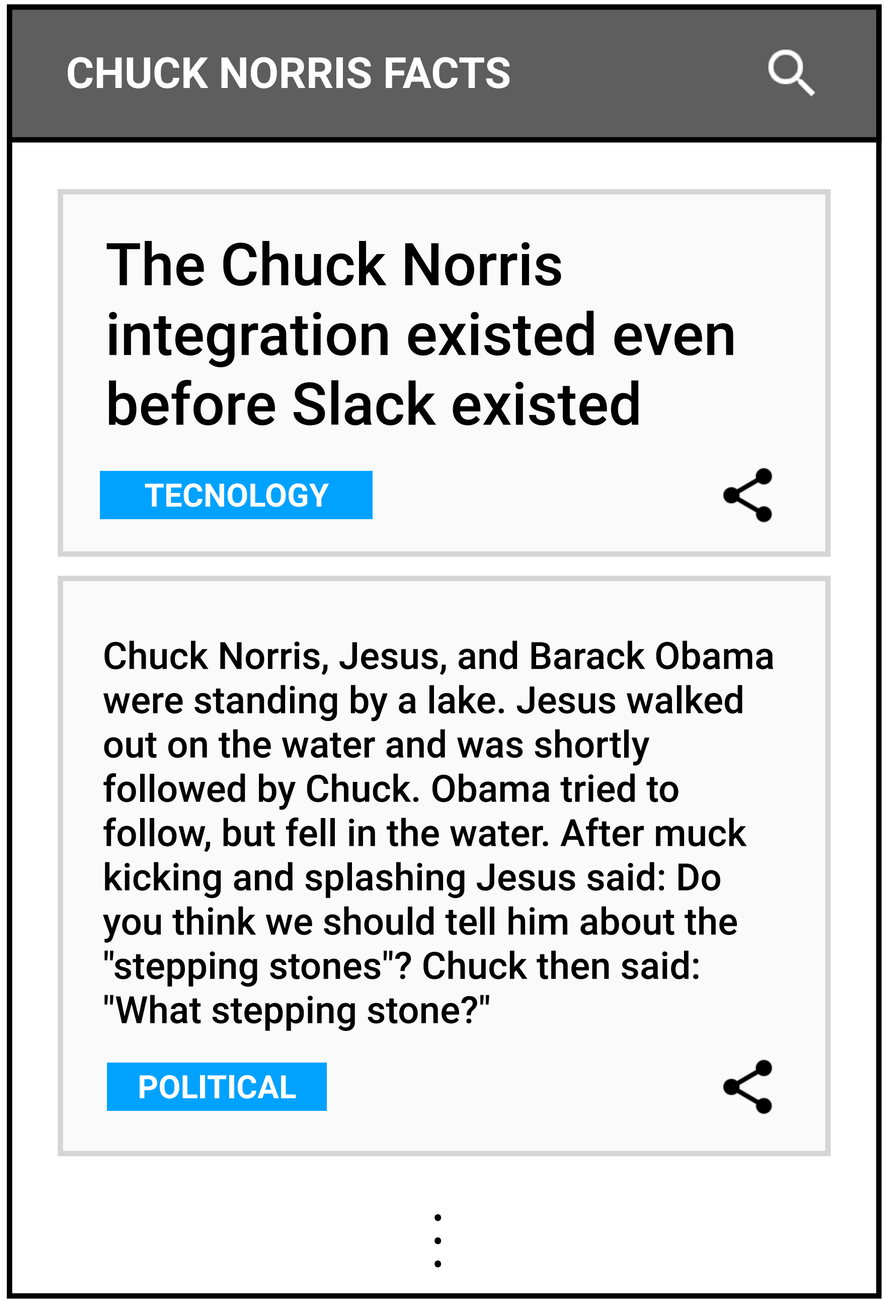
- A aplicação deve permitir que o usuário pesquise por Chuck Norris Facts via API e compartilhe eles com amigos

- A documentação da API pode ser encontrada em

https://api.chucknorris.io/

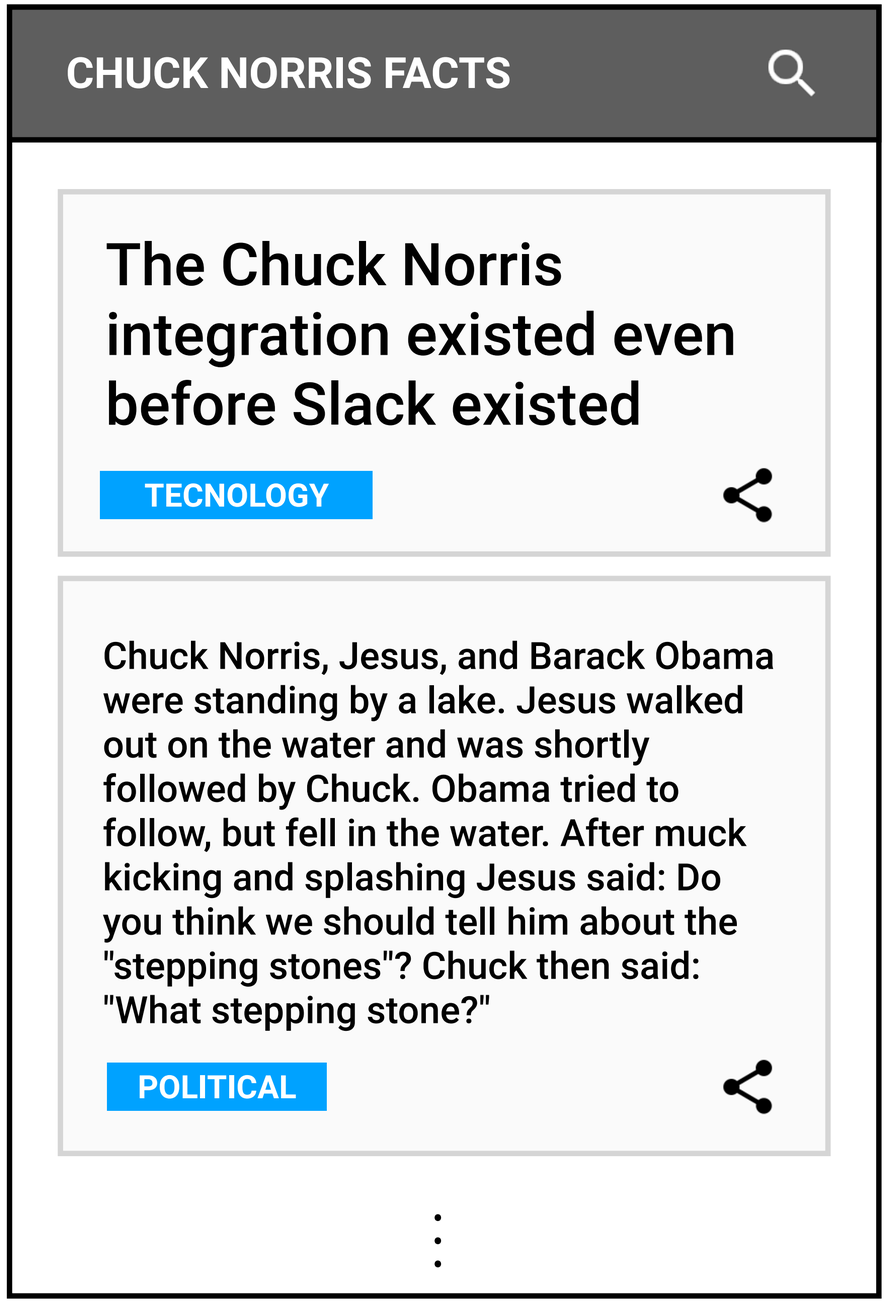
Parte 01 -FACTS (I)

- A tela principal da aplicação mostra a lista com os fatos do Chuck Norris. **No primeiro acesso do app, essa tela está vazia**
- Essa tela oferece um **link acionável** que direciona o usuário para a tela onde ele pode pesquisar por fatos (mais detalhes adiante).
- Ao pesquisar com sucesso um fato, os resultados são exibidos na tela principal da aplicação, como mostrado ao lado
- Utilize a abordagem de UI mais aderente à plataforma para **(a) exibir a listagem e (b) chamar a tela de pesquisa**. Considere a proposta ao lado como um *wireframe* nesse sentido



Parte 01 -FACTS (II)

- Cada entrada na lista deve ser acionável, no sentido de permitir que o usuário compartilhe a URL do Chuck Norris Fact. Use o mecanismo mais simples do sistema
- As entradas da lista podem variar muito com o tamanho do fato em si : para isso, **você deve adotar dois tamanhos de fonte diferentes de acordo com o número de caracteres do Chuck Norris Fact. Use 80 caracteres como critério de decisão**. Essa lógica de front-end deve ser testável.
- **Nem todos os fatos podem ter uma categoria associada**. Nesse caso, um label **UNCATEGORIZED** deve ser associado à entrada na lista. Essa lógica de front-end deve ser testável.
- Essa lista não é paginada e nem atualizável por *pull-to-refresh* ou similares



Parte 01 -FACTS (III)

- Sua lógica deve ser robusta o suficiente para informar ao usuário não apenas os resultados de pesquisa bem sucedida, mas também estados de erro de interesse
- Em particular, você deve oferecer estados e implementações de UI e ações distintas para erros de conexão e erros de REST
- Todas as possíveis saídas da tela devem estar testadas de forma automatizada em nível de integração, e prefencialmente em nível funcional de aceitação via testes de UI

BOA PROVA