

# 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 níveis 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 **Continuous 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 recomendamos 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

CHUCK NORRIS FACTS

The Chuck Norris integration existed even before Slack existed

TECNOLOGY

Chuck Norris, Jesus, and Barack Obama were standing by a lake. Jesus walked out on the water and was shortly followed by Chuck. Obama tried to follow, but fell in the water. After muck kicking and splashing Jesus said: Do you think we should tell him about the "stepping stones"? Chuck then said: "What stepping stone?"

POLITICAL

# 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 preferencialmente em nível funcional de aceitação via testes de UI



# Parte 02 - SEARCH (I)

- A segunda tela da aplicação permite ao usuário a possibilidade de digitar um termo de busca (que será enviado na consulta da API).
- Adicionalmente, essa tela também oferece uma pequena **nuvem de tags** com sugestões de termos de pesquisa, bem como lista os últimos termos de busca pesquisados pelo usuários
- Utilize a abordagem de UI mais aderente à plataforma para essa tela, **no tocante ao input de texto e listagem de termos**. Considere a proposta ao lado como um *wireframe* nesse sentido

## SEARCH FACTS

Enter your search term

Suggestions

GAMES

SPORTS

DEV

SCIENCE

TECHNOLOGY

MUSIC

TRAVEL

CARRER

Past Searches

Star Wars

Github

Trump

Dumb

# Parte 02 - SEARCH (II)

- A nuvem de tags deve ser populada a partir das categorias que a REST API expõe. **Elas devem ser carregadas na primeira vez que o app é iniciado e persistidas offline, de forma assíncrona.** Essa lógica de front-end deve ser testável.
- Dentre as categorias pré-carregadas disponíveis, **8 delas são escolhidas ao acaso para formar as sugestões de busca.** Essa lógica de front-end deve ser testável.
- Cada entrada nas sugestões de busca é clicável e deve acionar a busca por termo usando a própria categoria como parâmetro.

## SEARCH FACTS

Enter your search term

### Suggestions

GAMES

SPORTS

DEV

SCIENCE

TECHNOLOGY

MUSIC

TRAVEL

CARRER

### Past Searches

Star Wars

Github

Trump

Dumb

# Parte 02 - SEARCH (III)

- Os termos de busca já utilizados pelo usuário estão exposto abaixo das sugestões e são **persistidos offline**. Esses termos são clicáveis e disparam uma nova busca na API usando o termo como parâmetro.
- A listagem de termos já usados não apresenta **entradas repetidas**. Essa lógica de front-end deve ser testável.
- A listagem de termos já usados está ordenada de forma que o **termo mais recentemente utilizado aparece como primeiro na lista**. Essa lógica de front-end deve ser testável.

## SEARCH FACTS

Enter your search term

### Suggestions

GAMES

SPORTS

DEV

SCIENCE

TECHNOLOGY

MUSIC

TRAVEL

CARRER

### Past Searches

Star Wars

Github

Trump

Dumb

# Parte 02 - SEARCH (IV)

- Quando o usuário optar por digitar um novo termo de busca, você deve oferecer a opção configuração de teclado / input de texto adequada(s) pra que a busca seja disparada no Enter do teclado

SEARCH FACTS

Donald Trump |

Suggestions

GAMES

SPORTS

DEV

SCIENCE

TECHNOLOGY

MUSIC

TRAVEL

CARRER

QWERTYUIOP

ASDFGHJKL

↑ZXCVBNM↵

?123🎤[ ]...

# EXTRAS DESSE DESAFIO

- Nós recomendamos que você escolha um dos extras a seguir para complementar a entrega desse desafio
- As funcionalidades adicionais propostas apresentam um novo grau de dificuldade em relação às anteriores e contam pontos adicionais para a avaliação do processo.
- Você ganha (muito) mais pontos se fizer os dois extras

# EXTRA 01 - Auto-retry

- Adicione suporte a retry automático via **Reactive Programming** para chamadas REST que a aplicação executa.
- Para isso, você deve implementar uma lógica tal que, enquanto o usuário vê algum estado de tela referente a "Carregando" ou "Processando", a aplicação tenta fazer uma nova chamada da API REST caso um erro de networking ou erro 5xx seja detectado na operação



# EXTRA 01 - Auto-retry

- Como critério de aceitação desse extra, você deve
  - ▶ Decidir como identificar erros de conexão e quais deles serão tratados
  - ▶ Associar à lógica de auto-retry em todos os pontos da aplicação que executam chamadas REST
  - ▶ Assumir que serão feitas no máximo 2 tentativas, com intervalos de tempo de 4 e 8 segundos entre cada uma delas.
- **Esses comportamentos e definições devem estar testados em níveis de unidade e integração**

# EXTRA 02 - Offline Facts

- Adicione **suporte offline-first** na aplicação. Para isso, você deve persistir por algum mecanismo os resultados de pesquisas bem sucedidas e correlacionar os mesmos com termos já pesquisados. O *storage* local deve ser a fonte única de verdade para a aplicação
- Como critérios de aceitação desse extra, o usuário consegue
  - ▶ Visualizar offline todos os fatos associados a termos que ele já pesquisou anteriormente
  - ▶ Ver novos novos fatos associados a um termo já pesquisado **quando online**, caso existam, **independente desses virem do banco de dados**
  - ▶ Ter uma **amostra aleatória de até 10 fatos escolhidos ao acaso na tela inicial**, quando offline ou online, com base no que estiver localmente persistido no device
- **Esses comportamentos e definições devem estar testados em níveis de unidade e integração**

# BOA PROVA

stone<sup>®</sup>