

*Situação Problema–* ***CheckList***

*Curso:* ***ADS – Análise e Desenvolvimento de Sistema***

*Ano atualização:* ***2023/1***

*Nome da Trilha: Projeto Individual*

Blogg Twitr

**Tema**

Link para Trello: <*após criá-lo, adicione o link aqui*>

**CHECKLIST DE ATIVIDADES - Projeto Individual****[1](#_bookmark1)**

Escolher proposta\*

projeto Criar Desenho Rico\*

Justificativa para o projeto\*

Criar Diagrama de Caso de Uso

Criar Digrama de Entidade e Relacionamento \*

Descrever proposta de sistema \*

Descrever Regras de negócio\*

Descrever Requisitos Funcionais\*

❏

Descrever Requisitos Não Funcionais \*

[1](#_bookmark0)A ordem acima pode ser alterada de caso para caso, assim a palavra que vale é a de quem lhe orientar!

## O sistema deve conter no mínimo:

❏ *Página padrão;*

❏ *Página de Contato;*

❏ *Página de About (quem somos):*

❏ *Página de Login;*

❏ *Página de gerenciamento (Para administrar as informações mostradas na pagina padrão e também de usuários);*

* *Consultar;*
* *Adicionar;*
* *Alterar; ou*
* *Excluir.*

Cabe ressaltar que cada projeto terá suas especificidades ais quais serão adaptativas na analise de software podendo ter itens complementares.

Ressaltamos que não se deve pensar em algo muito grande para que não demande muito tempo e recursos no desenvolvimento desse primeiro projeto.

# CRIAR QUADRO NO TRELLO

## RESULTADO FINAL:

link do quadro do Trello na página inicial do projeto:

https://trello.com/b/8wDznO8V/kanban-projeto-individual

# 

# DESENHO RICO

## RESULTADO FINAL:

# 

# MAPA MENTAL

## RESULTADO FINAL:

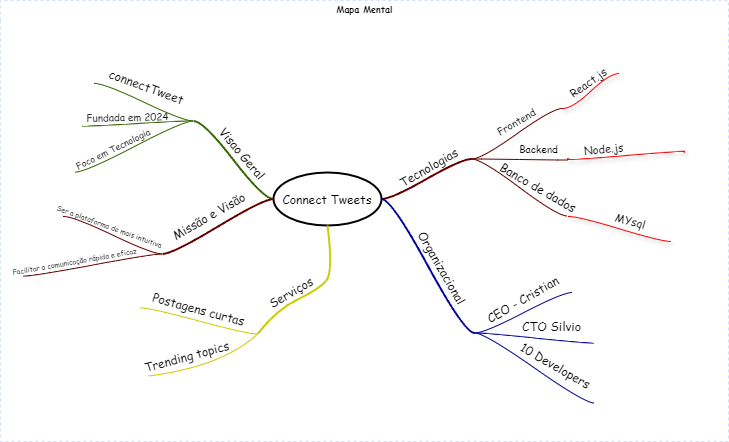
****

Fig1 Mapa mental.

# REGRAS DE NEGÓCIO

## RESULTADO FINAL:

1. **Deve haver um cadastro de usuário para ver os tweets**
2. **Um usuário não pode editar os tweets de outros usuários**
3. **Um usuário não pode eliminar os tweets de outros usuários**
4. O tweet deve conter no máximo 280 caracteres.
5. O Administrador pode eliminar as contas existentes.
6. O usuário pode postar tweets.
7. O usuário pode editar e eliminar seus tweets
8. Só pode existir um administrador

# REQUISITOS FUNCIONAIS

## RESULTADO FINAL:

1. **Tela de Registro**
   * O sistema deve permitir que novos usuários criem uma conta fornecendo um nome de usuário, e-mail e senha.
   * O sistema deve validar se o e-mail é único e se a senha atende aos critérios de segurança.
2. **Tela de Login**
   * O sistema deve permitir que usuários registrados façam login com seu e-mail e senha.
   * O sistema deve autenticar o usuário e redirecioná-lo para a página principal.
3. **Página Principal**
   * A página deve exibir uma lista de tweets de todos os usuários, incluindo o texto do tweet e autor.
   * O usuário deve poder escrever um novo tweet e publicá-lo.
   * O usuário deve poder editar seus próprios tweets.
   * O usuário deve poder excluir seus próprios tweets.
4. **Tela de Administração**
   * O administrador deve poder desativar ou excluir contas de usuários e com isso os tweets gerados.
5. **Notificações e Mensagens**
   * O sistema deve fornecer mensagens de erro e sucesso apropriadas para ações como registro, login, postagem, edição e exclusão de tweets.
6. **Segurança**
   * O sistema deve proteger as senhas dos usuários utilizando técnicas de hashing.
   * O sistema deve garantir que apenas usuários autenticados possam postar, editar e excluir tweets.

# REQUISITOS NÃO FUNCIONAIS

1. RESULTADO FINAL:  
     
   **Desempenho**
   * O sistema deve ser capaz de processar e responder a requisições de usuários em menos de 2 segundos na maioria das operações.
2. **Escalabilidade**
   * O sistema deve ser projetado para suportar o crescimento do número de usuários e tweets sem necessidade de grandes alterações estruturais.
3. **Segurança**
   * A aplicação deve implementar medidas de segurança para proteger contra ataques comuns, como SQL Injection, Cross-Site Scripting (XSS) e Cross-Site Request Forgery (CSRF).
   * As conexões devem ser criptografadas utilizando HTTPS.
4. **Usabilidade**
   * A interface deve ser intuitiva e fácil de usar para os usuários, com uma navegação clara e responsiva.
   * O design deve ser adaptável para diferentes dispositivos, incluindo desktops e dispositivos móveis.
5. **Manutenibilidade**
   * O código-fonte deve ser bem documentado e organizado para facilitar a manutenção e futuras melhorias.
   * O sistema deve seguir boas práticas de desenvolvimento e padrões de código para garantir a clareza e a qualidade.
6. **Disponibilidade**
   * O sistema deve ter alta disponibilidade, com uma meta de tempo de inatividade não superior a 1% ao mês.
7. **Backup e Recuperação**
   * O sistema deve implementar estratégias de backup regulares para garantir a recuperação de dados em caso de falha.

**3. Arquitetura do Sistema**

**3.1 Estrutura Geral**

* **Frontend**: Interface do usuário desenvolvida com HTML, CSS e JavaScript. Utilização de frameworks como.
* **Backend**: Servidor implementado em Node.js com Express para gerenciar as rotas e lógica de aplicação.
* **Banco de Dados**: Armazenamento dos dados dos usuários e tweets em um banco de dados relacional como MySQL.

**3.2 Diagrama de Arquitetura**

* **Frontend**: Comunica-se com o backend através de APIs RESTful.
* **Backend**: Processa requisições do frontend, realiza operações CRUD no banco de dados e aplica regras de negócio.
* **Banco de Dados**: Armazena usuários, tweets e informações relacionadas.

**4. Tecnologias Utilizadas**

* **Node.js**: Ambiente de execução JavaScript no lado do servidor.
* **Express.js**: Framework para construção de APIs e gerenciamento de rotas.
* **MySQL**: Sistema de gerenciamento de banco de dados relacional.
* **HTML/CSS/JavaScript**: Tecnologias para o desenvolvimento do frontend.
* ***Boostrap*** (opcional): Frameworks para construção de interfaces dinâmicas.

**5. Implementação**

**5.1 Requisitos**

* **Node.js e npm**: Instalados no ambiente de desenvolvimento.
* **Banco de Dados**: Configurado e acessível.
* **Ferramentas de Desenvolvimento**: IDE e ferramentas de controle de versão (Git).

**5.2 Passos para Implementação**

1. **Configuração do Ambiente**: Instalação do Node.js e configuração do banco de dados.
2. **Desenvolvimento do Backend**: Criação das APIs para autenticação, gerenciamento de tweets e contas.
3. **Desenvolvimento do Frontend**: Implementação das interfaces de usuário e integração com o backend.
4. **Testes**: Testes unitários e funcionais para garantir a qualidade do sistema.
5. **Deploy**: Implementação em um ambiente de produção, se aplicável.

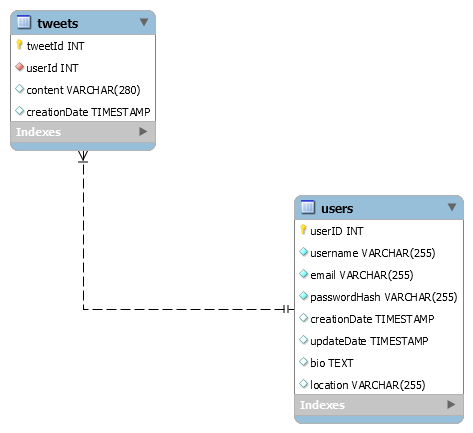
# DIAGRAMA DE CASOS DE USO

## RESULTADO FINAL:

# DIAGRAMA ENTIDADE RELACIONAMENTO

## RESULTADO FINAL:

.



* Tela de autenticação;
* Tela inicial de cada perfil: por exemplo, se tiver os perfis de administrador e cliente, serão 2 telas. Nessas telas, desejamos ver o padrão de cabeçalho, de menu (se for do estilo dropdown, exiba-o aberto) e rodapé - esses 3 devem ser iguais em todas as páginas daquele perfil - e também o que será carregado na parte “central” assim que o usuário for autenticado;
* Telas do padrão visual para os Requisitos de Entrada (CRUDs): para não ter que fazer todos os wireframes desses Requisitos, sugerimos que represente apenas o mais complexo (com

mais dados) ou então represente de um modo genérico como será o padrão deles. Dessa maneira já conseguiremos compreender como será feita cada uma das 4 operações de cada CRUD, afinal todos seguirão este padrão. Caso haja modais ou telas diferentes para alguma das operações, também as represente;

* Todos os Requisitos Funcionais de Processo e Saída (todas as telas/modais envolvidas): como cada RF desses deve ter suas especificidades, é importante representá-las com clareza. Considere também fazer paginação nos registros e consultas (não fizemos nas OTs, mas é básico em um software), e se pensar em outras coisas, contanto que não sejam funcionalidades novas, podemos considerar, mas ao mesmo tempo, cuidado para não dizer que fará coisas que nem sabe ainda, mas “já viu antes, então dá pra fazer”, sem nem pesquisar como se faz! Se quiser

implementar algo que não viu nas OTs, primeiro pesquise como fazer, e então pense o seguinte:

* Realmente é algo necessário para meu PI ou posso ver isso depois, no Consolidado?
* Levará muito tempo para ser feito?
* Precisarei fazer “na mão” ou há algum plugin/biblioteca que faz?
* É de uma dificuldade aceitável?

Aí sim, poderemos discutir sobre a viabilidade de acrescentar um novo conhecimento no projeto. Assim que terminar, antes de validar, use as telas criadas, principalmente nos Processos e Saídas, para simular o fluxo de execução de cada funcionalidade. Isso vai te ajudar a perceber se não falta nenhuma tela/botão.

Feito isso, aí sim, procure seus orientadores para a validação.

Ah, é MUITO IMPORTANTE que defina a ideia final de TODAS as telas, pois isso não deve ser alterado no futuro (na hora de programar), exceto em caso de algum erro. O QUE VOCÊ PROJETAR E FOR APROVADO, VOCÊ IRÁ REALIZAR!

**P.S.:** Capriche na divisão das atividades do checklist desse cartão no Trello, tem bastante coisa pra fazer...

# CRIAR ESTRUTURA DE ARQUIVOS DE SISTEMA

## RESULTADO FINAL:

A organização por diretórios é o modo como o Sistema organiza logicamente os diversos arquivos contidos em um dispositivo físico de armazenamento.

## CONTEXTUALIZANDO:

Diretórios

O diretório contém entradas associadas aos arquivos onde são armazenadas informações como localização física, nome, organização e demais atributos.

Ao abrir um arquivo, o Sistema Operacional procura a sua entrada na estrutura de diretórios em uma tabela mantida na memória principal, contendo todos os arquivos. É necessário fechar o arquivo ao término de seu uso.

## Nível Único:

* Organização mais simples de uma estrutura de diretórios.
* Existe apenas um único diretório contendo todos os arquivos do disco.
* O nível único é bastante limitado, não permitindo que usuários criem arquivos com mesmo nome.

## Master File Directory (MFD):

* Existe um nível de diretório adicional para controlar os diretórios individuais dos usuários.
* indexado pelo nome do usuário e, nele, cada entrada aponta para o diretório (UFD) pessoal.

## Estrutura de diretórios em árvore:

* Existe o diretório MFD que é a raiz, os galhos são os UFD e os arquivos são as folhas.
* Cada subdiretório abaixo do MDF pode conter arquivos e novos subdiretórios e assim por diante.
* Quando se referencia a um arquivo, é necessário especificar seu nome, bem como o diretório onde ele se encontra, referência chamada PATH.
* Mais organizada e adotada pela maioria dos Sistemas Operacionais.
* Na maioria dos sistemas, diretórios também são tratados como arquivos, com identificação de atributos, proteção identificação do criador e data da criação.

# CRIAR INTERFACE DE LOGIN

## RESULTADO FINAL:

Front-End da interface visual da tela de login finalizado

## CONTEXTUALIZANDO:

Neste momento, desenvolva o frontend da tela de autenticação conforme seu wireframe.

Por ser sua primeira tela, você criará seus primeiros arquivos HTML, CSS e JS (esse, por enquanto, apenas para validar o formulário), e talvez ainda precise de alguma biblioteca (talvez jQuery e jQueryUI ou Bootstrap) ou imagem. Assim, organize seus arquivos de uma maneira coerente, pensando em todo o restante do projeto que está por vir e todos os outros arquivos que utilizará.

Se essa tela for a inicial de seu website, lembre-se que ela deve ser a index.html e estar na raiz.

## CRIAR INTERFACE DE LAYOUT GERAL DE CADA PERFIL RESULTADO FINAL:

Frontend das interfaces com o layout de cada perfil finalizado.

## CONTEXTUALIZANDO:

Neste momento, você desenvolverá as telas para os wireframes de tela inicial de cada perfil.

Essas telas deverão conter o cabeçalho, menu e rodapé conforme seu wireframe, e uma parte da tela deve ser “reservada” para possibilitar que sejam carregados dinamicamente os conteúdos, algo que faremos mais à frente.

Pense bem na organização de pastas, separando cada perfil (isso vai lhe ajudar no futuro), porém, levando em consideração o que é igual entre eles, como por exemplo, se todos tiverem configurações CSS em comum, você poderá colocar tudo em apenas um arquivo CSS.

## DESENVOLVER A PROGRAMAÇÃO DO PRIMEIRO CRUD DO PROJETO RESULTADO FINAL:

Front-End e Back-End do primeiro CRUD funcionais e finalizados

## CONTEXTUALIZANDO:

Primeiro, defina o CRUD a ser realizado. Sugerimos que seja o Requisito Funcional de entrada que tenha menos campos e que seja um sem chaves estrangeiras pois, como será seu primeiro, terá várias coisas pra fazer além de simplesmente programá-lo (como criar a classe de conexão ao banco, por exemplo).

Depois, pense quais os arquivos necessários para isso, e onde os criará, tanto do frontend quanto do backend (que, por sinal, será iniciado agora, ou seja, pense também na organização de seus pacotes). Lembre-se de levar em conta que o arquivo HTML dele deve ser carregado dentro da área destinada para isso no layout geral (então já é uma boa hora de fazer a função de carregamento dinâmico das páginas), que o CSS desse CRUD será usado em todos os CRUDs (então use nomes mais genéricos para suas classes visando reaproveitamento), e que será a primeira ação backend de seu projeto (então o web.xml e o Jersey farão sua estreia em seu projeto).

A seguir, crie um cartão no Trello com o mesmo nome do Requisito Funcional escolhido, e o checklist dividindo as ações necessárias para sua finalização.

Por fim, prepare o SQL que criou a partir de seu DER para execução e, seguindo também seus wireframes, faça com que esse CRUD funcione conforme planejado.

## FINALIZAR E APLICAR TRILHA DE SEGURANÇA RESULTADO FINAL:

Conhecimentos adquiridos na trilha aplicados no projeto.

## CONTEXTUALIZANDO:

Sua primeira tarefa nesse momento é realizar o(s) CRUD(s) relacionados a registro dos usuários do software (de todos os níveis), e se houver outro CRUD que precise ser feito antes, por dependência direta entre eles, faça também. Solicitamos apenas que converse sobre isso com seu orientador, indicando quais CRUDs você identificou que devem ser feitos antes dos relativos ao cadastro de usuários.

Assim que finalizar, estão disponibilizadas lá no Ava as OTs da trilha de Segurança. Sua missão é fazer a trilha, validando conosco cada OT, e aplicá-la em seu projeto.

Claro, crie os cartões no Trello necessários para registrar suas atividades, tanto dos CRUDs quanto de cada OT da própria trilha.

## DESENVOLVER PROGRAMAÇÃO DOS R.F. DE ENTRADA RESULTADO FINAL:

frontend e backend dos R.F. de entrada finalizados

## CONTEXTUALIZANDO:

Agora, siga o fluxo, finalizando todas as entradas, uma por uma, das mais simples às mais complexas (ou seja, deixe os que envolvem relacionamentos entre tabelas para o final). A cada uma,

faça primeiro o frontend e depois o backend, indicando uma por uma sua finalização para que possamos validar.

Apesar do layout de todos os CRUDs ser o mesmo, dependendo de seu projeto a partir daqui você já terá algumas novidades, como manipular duas tabelas do banco de dados pelo mesmo formulário ou fazer o preenchimento de um campo afetar o carregamento de outro (como por exemplo, numa situação onde escolher um estado carregará em outro campo apenas as cidades daquele estado). Quando chegar em um caso destes, com algo novo, nos procure para que lhe orientemos sobre a melhor maneira de fazer este registro.

No Trello, crie um card para cada R.F., ordenando-os no *backlog* na sequência em que pensa em fazê-los. Ao definir os prazos, pense bem em tudo o que deve ser feito, e se houverem novidades na programação, considere o tempo de aprendizado.

## DESENVOLVER PROGRAMAÇÃO DOS R.F. DE PROCESSO RESULTADO FINAL:

frontend e backend dos R.F. de processo finalizados

## CONTEXTUALIZANDO:

Continue a programação finalizando todos os processos, um por um, primeiro frontend e depois backend, indicando um por um sua finalização para que possamos validar.

Nessa fase, já é mais provável que o layout não seja mais o mesmo. Além disso, é aqui que fará sua função de recuperar senha (que utilizará envio de e-mail), então novos conhecimentos serão necessários! Então, pense bem no que fazer e em como fazer, lembre-se de seus wireframes e conte conosco nas dúvidas que surgirem!

No Trello, como na etapa anterior, crie um card para cada R.F., ordenando-os no *backlog* na sequência em que pensa em fazê-los.

## DESENVOLVER PROGRAMAÇÃO DOS R.F. DE SAÍDA RESULTADO FINAL:

Frontend e Backend dos R.F. de saída finalizados

## CONTEXTUALIZANDO:

Para fechar, é hora de programar as saídas, da mesma maneira que antes.

Aqui já é provável que possa talvez reaproveitar algumas configurações anteriores, como dos formulários (para filtrar as consultas) ou das tabelas (para exibir os dados), mas isso depende de seus wireframes. Talvez até você tenha pensado em gráficos e, se for o caso, deve pesquisar um plugin que atenda suas necessidades. Além disso, conforme definido no projeto, você irá programar uma de suas saídas para possibilitar impressão via PDF ou exportação via CSV, então verifique na internet um plugin para isso, compreenda-o e use-o.

No Trello, siga as orientações das duas etapas anteriores.

## ATUALIZAR DOCUMENTAÇÃO DO SOFTWARE APÓS SEU DESENVOLVIMENTO RESULTADO FINAL:

PI finalizado!!!

## CONTEXTUALIZANDO:

Depois de finalizar a programação de seu PI, vamos fazer uma super hiper ultra mega power validação! Depois dela, pode ser que peçamos para fazer alguns ajustes no projeto, sejam eles anotados por você e negociados conosco durante o andamento do projeto, ou realmente verificados e solicitados por nós.

Crie esse cartão no Trello e mais um cartão chamado “Ajustes finais de programação”, e no momento da validação, alimentaremos juntos os checklists de ambos com os ajustes necessários.

*Uma vez que todas as solicitações sejam atendidas, podemos seguir para novos aprendizados!!*