



## Trabalho Integrador

Este trabalho tem como objetivo a prática de integração entre frontend e backend do CCR GEX613 - Programação II utilizando conceitos desenvolvidos em paralelo com os componentes de Banco de Dados e Engenharia de Software.



### I Instruções

Desenvolver uma aplicação web completa, utilizando as tecnologias HTML5, CSS3, JavaScript e integração com banco de dados relacional, que permita:

- A. a realização de operações CRUD (Create, Read, Update, Delete);
- B. a exibição de informações em um dashboard dinâmico; e
- C. o gerenciamento de acesso com autenticação e autorização de usuários.

O projeto integra competências de Programação II, Engenharia de Software e Banco de Dados, promovendo o desenvolvimento de uma solução que reflita práticas profissionais de engenharia de sistemas.

Funcionalidades obrigatórias

1. **Sistema de Autenticação e Autorização:** Implementação de sistema de login. Controle de sessão do usuário (pode ser via cookies, tokens ou sessões). Diferenciação de acesso para pelo menos dois perfis de usuários (ex.: Administrador e Usuário Comum). Restrição de funcionalidades de acordo com o perfil de acesso.
2. **Operações CRUD:** O sistema deve possuir, no mínimo, três entidades distintas com operações CRUD completas (Create, Read, Update, Delete). Cada CRUD deverá permitir: Criação de novos registros. Listagem dos registros. Edição e atualização de registros existentes. Exclusão de registros. Exemplos possíveis de entidades: Usuários, Produtos,



Pedidos. Estudantes, Disciplinas, Matrículas.

3. **Dashboard:** Implementação de um dashboard acessível após autenticação. O dashboard deve apresentar dados analíticos, como: Quantidade de registros em cada tabela. Informações resumidas (ex.: número de usuários ativos, pedidos realizados, etc.). Deve ser implementada a atualização dinâmica dos dados exibidos.
4. **Visualização, Ordenação e Filtragem:** Exibição organizada dos dados Possibilidade de ordenação por pelo menos dois campos (ex.: nome e data). Possibilidade de filtragem textual ou por categoria.
5. **Responsividade:** A aplicação deverá ser responsiva, adaptando-se adequadamente a dispositivos móveis, tablets e desktops.
6. **Tratamento de erros:** Mensagens claras para o usuário em caso de falhas (ex.: falha na autenticação, falha na conexão com a API, etc.).



## II Entrega

**Github:** Código fonte e um README com as instruções para executar o projeto, bem como as bibliotecas que são necessárias para o funcionamento do código e demais instruções necessárias. Para os estudantes cursando Banco de Dados ou que utilizarem algum banco de dados no projeto, deve ser entregue script de criação das tabelas e quaisquer outros comandos para inserir dados necessários para a operação inicial do projeto.


**SIGAA:** Além do código, cada estudante deverá entregar um relatório escrito, em formato PDF, contendo a descrição da lógica utilizada na implementação, comentando sobre as decisões tomadas e as facilidades/dificuldades durante este tempo. Este relatório deve conter o link para o repositório do Github onde está a entrega do código fonte.

**Data limite para entrega (previsão): 01/07/2025, 23h59m.**



### III Avaliação

Os seguintes critérios serão avaliados

1. **Organização e Estruturação do Código-Fonte:** Avaliar a disposição lógica e modular do código HTML, CSS e JavaScript, considerando a separação adequada de responsabilidades, a presença de comentários explicativos e a legibilidade geral do projeto.
2. **Utilização Correta de Arquivos CSS e JS Externos:** Verificar se o projeto adota as boas práticas de desenvolvimento web, como a separação do estilo (CSS) e da lógica de interação (JavaScript) em arquivos externos, evitando a utilização excessiva de código embutido no HTML (<style> e <script> inline).
3. **Padrão de Nomenclatura de Variáveis e Funções:** Avaliar a utilização consistente de padrões de nomenclatura, como camelCase para variáveis e funções em JavaScript, além da escolha de nomes significativos, autoexplicativos e aderentes ao domínio da aplicação. 
4. **Boas Práticas na Escrita de HTML5:** Analisar a conformidade do código com os padrões HTML5, incluindo o uso semântico de tags e a correta estruturação da página com <!DOCTYPE html>, <html>, <head>, <body>, entre outros.
5. **Estilização Responsiva com CSS:** Avaliar se o CSS foi desenvolvido levando em conta a responsividade da aplicação, utilizando técnicas como media queries (@media) para adaptação a diferentes tamanhos de tela, garantindo acessibilidade em dispositivos móveis e desktop.
6. **Estrutura e Modularização do Código JavaScript:** Analisar se o código JavaScript foi estruturado de forma modular (uso de funções reutilizáveis, classes ou módulos, se aplicável), evitando duplicação de código e respeitando os princípios de baixo acoplamento e alta coesão.
7. **Tratamento de Erros e Validação de Formulários:** Avaliar a implementação de mecanismos de validação de dados em formulários HTML, tanto no lado do cliente (JavaScript) quanto no lado do servidor, quando aplicável, com mensagens de erro claras e feedback ao usuário.
8. **Integração e Consumo de APIs (REST ou Similares):** Avaliar a capacidade de consumir APIs RESTful utilizando JavaScript (por exemplo, com fetch ou axios), realizando requisições assíncronas (AJAX) e tratamento adequado de respostas e erros.
9. **Qualidade Estética e Usabilidade da Interface:** Avaliar a preocupação com a experiência do usuário (UX) e a apresentação visual da aplicação (UI), incluindo a escolha adequada de fontes, cores, espaçamento e hierarquia visual para facilitar a navegação e a compreensão.
10. **Coerência do relatório apresentado.** A exposição deve apresentar sequência lógica,



clareza nas justificativas das decisões técnicas, correspondência entre a descrição da lógica e a implementação efetiva, e alinhamento entre as dificuldades relatadas e as soluções adotadas.

#### IV Rúbrica de Avaliação

Critério	Nota	Descrição
Organização e Estruturação do Código-Fonte	0	Código desorganizado, sem estrutura, mistura de camadas (HTML, CSS, JS).
	1-2	Organização muito ruim; mistura frequente de responsabilidades; ausência de identificação.
	3-4	Organização inadequada; separação parcial das camadas; identificação inconsistente.
	5-6	Organização aceitável; separação feita, mas com vários problemas de estrutura e identificação.
	7-8	Boa organização geral; separação clara e lógica, pequenos ajustes pendentes.
	9-10	Código exemplarmente organizado, separação perfeita entre camadas e identificação correta.
Utilização Correta de Arquivos CSS e JS Externos	0	Todos os estilos e scripts embutidos no HTML.
	1-2	Uso majoritário de código embutido; arquivos externos mal referenciados.
	3-4	Separação parcial; parte externa, parte embutida.
	5-6	Separação feita, mas com estrutura de chamadas confusa.
	7-8	Boa separação; arquivos bem referenciados; pequenos erros de organização.
	9-10	Separação correta e organizada, seguindo as melhores práticas.
Padrão de Nomenclatura de Variáveis e Funções	0	Nomes aleatórios; ausência total de padrão.
	1-2	Nomes inconsistentes e confusos; padrão ignorado.
	3-4	Padrão aplicado de forma irregular; inconsistências frequentes.
	5-6	Padrão de nomenclatura utilizado, mas com erros pontuais.
	7-8	Boa consistência na nomeação; pequenas falhas.
	9-10	Nomenclatura consistente, significativa e correta em todo o projeto.
Boas Práticas na Escrita de HTML5	0	Estrutura HTML inadequada; semântica ignorada.
	1-2	Estrutura mínima, com uso incorreto de tags.
	3-4	Estrutura básica, mas uso semântico irregular.
	5-6	Semântica aplicada em boa parte, mas com erros.
	7-8	Uso correto de HTML5, pequenos ajustes necessários.
	9-10	Uso exemplar de HTML5, semântica e hierarquia correta.
Estilização Responsiva com CSS	0	Sem responsividade.
	1-2	Responsividade muito ruim; aplicação inutilizável em mobile.
	3-4	Responsividade parcial; adaptações incompletas.
	5-6	Responsividade aceitável, mas falhas evidentes em layouts.
	7-8	Boa responsividade geral; pequenos ajustes necessários.



	9–10	Responsividade perfeita; adaptação fluida a diferentes telas.
Estrutura e Modularização do Código JavaScript	0	Código desorganizado e repetitivo.
	1–2	Estrutura inadequada; funções desnecessariamente longas.
	3–4	Modularização básica, mas ainda com repetições.
	5–6	Estrutura modular razoável, mas com falhas de escopo ou dependências.
	7–8	Boa modularização; funções reutilizáveis.
	9–10	Código modularizado de forma excelente; alta reutilização e clareza.
Tratamento de Erros e Validação de Formulários	0	Sem tratamento de erros ou validações.
	1–2	Tratamento mínimo e mensagens confusas.
	3–4	Validação parcial; falhas frequentes.
	5–6	Tratamento razoável, mas cobertura de erros limitada.
	7–8	Boa validação e tratamento de erros; pequenas melhorias possíveis.
	9–10	Validação robusta e tratamento amigável de todos os erros.
Integração e Consumo de APIs	0	Nenhuma integração realizada.
	1–2	Tentativa de integração falha ou não funcional.
	3–4	Integração parcial ou maltratada; erros sem tratamento.
	5–6	Integração funcional, mas com problemas de estabilidade.
	7–8	Boa integração; fluxo de dados controlado, tratamento básico de erros.
	9–10	Integração impecável; tratamento robusto, requisições fluídas.
Qualidade Estética e Usabilidade da Interface	0	Interface confusa e desorganizada.
	1–2	Interface funcional, mas muito deficiente esteticamente.
	3–4	Interface compreensível, porém desleixada visualmente.
	5–6	Interface adequada, porém sem refinamento visual.
	7–8	Interface agradável e funcional, pequenos pontos de melhoria.
	9–10	Interface excelente, limpa, intuitiva e responsiva.
Coerência do relatório apresentado	0	Não apresentou ou o texto é incompreensível
	1–2	Descrição muito confusa, sem relação clara com o código apresentado.
	3–4	Descrição incompleta; poucos trechos explicados corretamente.
	5–6	Descrição razoável da lógica, mas com lacunas ou falta de profundidade.
	7–8	Boa descrição da lógica, com explicações coerentes e compreensíveis.
	9–10	Descrição completa e detalhada da lógica de implementação, de forma clara e precisa