UNIVERSIDADE DO OESTE DE SANTA CATARINA
UNOESC – CAMPUS SÃO MIGUEL DO OESTE
CURSO DE CIÊNCIA DA COMPUTAÇÃO

TASK MANAGER

TRABALHO REFERENTE AO COMPONENTE
PROGRAMAÇÃO III

Fernando Camilo

Schneider

Jean Carlo Toral

Joao Ulisses Porto Alegre Ciriaco Teixeira

São Miguel do Oeste, Santa Catarina,
Brasil 2025

Resumo

Este artigo detalha o desenvolvimento de uma aplicação de lista de tarefas, motivada pela necessidade de organizar ideias de forma independente e ágil. O projeto explora a aplicação de frameworks modernos, utilizando o **CodeIgniter** por sua leveza e agilidade, e seguindo o padrão arquitetural **MVC (Model-View-Controller)** para estruturação do código. A interação com o banco de dados **PostgreSQL** foi simplificada pelo **Mapeamento Objeto-Relacional (ORM)**, enquanto o uso de **Docker e Docker Compose** garantiu um ambiente de desenvolvimento consistente e portátil. O processo de desenvolvimento abordou desde a modelagem do banco de dados e a implementação das funcionalidades (cadastro e login de usuários, registro, visualização, exclusão e marcação de status de tarefas) até a configuração do ambiente e a execução de testes com PHPUnit. Desafios como a inicialização do projeto, problemas de conexão com o banco de dados, estruturação de arquivos, execução de queries e questões de cache foram enfrentados e superados, enriquecendo o aprendizado. Em suma, o trabalho demonstra os benefícios da utilização de frameworks para a construção de software eficiente e de fácil manutenção.

1. Introdução

No cenário atual de desenvolvimento web, a utilização de frameworks tornou-se indispensável para a criação de aplicações escaláveis, seguras e eficientes. Eles fornecem uma estrutura organizada, bibliotecas de código reutilizáveis e padrões de design que aceleram o processo de desenvolvimento e promovem a boa prática. Este artigo detalha uma aplicação de lista de tarefas, escolhida como um projeto prático para a organização de ideias de forma fácil, independente e rápida. Para isso, exploramos conceitos como o padrão arquitetural **MVC (Model-View-Controller)** para organizar o código, o **Mapeamento Objeto-Relacional (ORM)** para simplificar a interação com o banco de dados, e a **Containerização com Docker** para garantir um ambiente de desenvolvimento consistente. Durante o processo de desenvolvimento, enfrentamos desafios significativos, como o entendimento aprofundado do framework e a aplicação correta das lógicas de programação, além da dificuldade em resolver problemas específicos, como as marcações de tarefas completas. Apesar dessas adversidades, este projeto prático demonstra como um framework pode ser empregado para construir uma ferramenta útil, seguindo as orientações para artigos científicos.

2. A Aplicação de Lista de Tarefas

A aplicação de lista de tarefas, desenvolvida como um projeto de estudo de

frameworks, permite aos usuários gerenciar suas atividades diárias. As funcionalidades principais incluem:

- **Cadastro de Usuários:** Implementação de um módulo de registro que permite aos novos usuários criar credenciais de acesso, gerenciando a persistência de dados de autenticação e a integridade do banco de dados de usuários.
- **Login de Usuários:** Módulo de autenticação que valida as credenciais do usuário e estabelece uma sessão segura, garantindo o acesso autorizado às funcionalidades personalizadas e aos dados específicos do usuário.
- **Registro de Tarefas:** Funcionalidade para a persistência de novas entidades de tarefa no sistema, capturando dados como descrição e status inicial, e associando-as ao usuário autenticado.
- **Visualização de Tarefas:** Exibição dinâmica da coleção de tarefas persistidas no banco de dados, apresentando-as ao usuário de forma estruturada para facilitar o gerenciamento.
- **Exclusão de Tarefas:** Operação para a remoção permanente de registros de tarefas do banco de dados, mantendo a integridade referencial e o controle de acesso baseado no usuário.
- **Marcação de Status:** Funcionalidade para a atualização do estado de uma tarefa (e.g., de "pendente" para "concluída"), implicando a modificação de atributos específicos no registro persistido.

Esta aplicação serve como um excelente ponto de partida para entender como os frameworks facilitam a implementação de funcionalidades CRUD (Create, Read, Update, Delete) em um ambiente web.

3. Fundamentação Teórica

Esta seção deve apresentar os conceitos teóricos que embasam o projeto. Inclua aqui uma discussão aprofundada sobre:

- **Justificativa do Framework:** Optamos pelo **CodeIgniter** devido à sua natureza **leve e rápida**, ideal para projetos de médio porte que precisam ser desenvolvidos e implantados rapidamente. Ele oferece simplicidade e bom desempenho, sendo uma alternativa eficiente para quem busca agilidade sem a complexidade de frameworks maiores. Seu sistema ORM nativo é suficiente para as necessidades básicas de manipulação de dados da aplicação.
- **Conceitos de MVC (Model-View-Controller):** O MVC é um padrão de arquitetura que divide a aplicação em três partes para organizar o código.
    - O **Model** cuida dos dados e da lógica de negócios, interagindo com o banco de dados (ex: uma classe Tarefa).
    - A **View** é o que o usuário vê, ou seja, a interface gráfica (ex: arquivos HTML para exibir a lista de tarefas).

    - O **Controller** age como um intermediário, recebendo as requisições do usuário, processando-as e decidindo qual Visão mostrar.
- **Mapeamento Objeto-Relacional (ORM):** No projeto, o ORM nativo do CodeIgniter foi fundamental para simplificar a interação com o banco de dados. Em vez de escrever comandos SQL complexos para cada operação, utilizamos os métodos do ORM para realizar as ações de criar, ler, atualizar e deletar tarefas. Isso agilizou o desenvolvimento e tornou o código de manipulação de dados mais limpo e fácil de manter dentro da aplicação de lista de tarefas.
- **Containerização (Docker):** Docker e Docker Compose foram essenciais para criar um ambiente de desenvolvimento padronizado e sem conflitos para a aplicação de lista de tarefas. Utilizamos o **Docker** para empacotar a aplicação PHP, o servidor web e o banco de dados PostgreSQL em "contêineres" separados. O **Docker Compose** nos permitiu definir e gerenciar facilmente esses múltiplos contêineres juntos, garantindo que o ambiente de desenvolvimento fosse consistente e facilmente replicável para todos os membros da equipe, sem problemas de compatibilidade ou configuração demorada.

4. Arquitetura e Tecnologias

O projeto baseou-se em uma arquitetura robusta, utilizando as seguintes tecnologias:

4.1. Framework PHP

O núcleo da aplicação foi desenvolvido utilizando um framework PHP que serviu como a espinha dorsal do projeto. Ele ajudou a organizar o código, separando as preocupações em Modelos, Views e Controladores (MVC). Isso permitiu gerenciar o que acontece quando o usuário acessa uma página (roteamento), como os dados são processados e como a aplicação se comunica com o banco de dados. A escolha de um framework trouxe recursos de segurança e um padrão de trabalho que facilitou a criação e a manutenção do código.

4.2. Docker e Docker Compose

Para garantir que a aplicação funcionasse da mesma forma em qualquer computador, sem problemas de compatibilidade, utilizamos o Docker e o Docker Compose. O **Docker** nos permitiu "empacotar" a aplicação PHP junto com todas as suas dependências, como o interpretador PHP e o servidor web, em um contêiner isolado. Já o **Docker Compose** foi usado para orchestrar e conectar diferentes partes do ambiente. Por exemplo, configuramos um arquivo docker-compose.yml para iniciar tanto o contêiner da aplicação PHP quanto o do banco de dados PostgreSQL simultaneamente, simplificando muito a configuração e o gerenciamento de todo o ambiente de desenvolvimento. Isso significa que, ao invés de instalar o PHP, o servidor

web e o banco de dados individualmente, bastou rodar um comando para ter tudo pronto.

4.3. Banco de Dados PostgreSQL

O PostgreSQL foi escolhido como o banco de dados para armazenar todas as informações da aplicação, como os dados de usuários e as tarefas. Ele é um sistema de gerenciamento de banco de dados robusto e confiável, que lida bem com várias operações ao mesmo tempo e é compatível com os padrões de consulta de dados (SQL). Sua capacidade de lidar com dados estruturados de forma segura e eficiente o tornou a escolha ideal para gerenciar as tarefas e os usuários da nossa aplicação.

4.4. Ferramentas Adicionais

- **Composer:** Essencial para o projeto, o Composer foi utilizado para gerenciar as dependências do PHP. Isso significa que ele automatizou a instalação e a atualização de todas as bibliotecas de código que o framework e outras partes do projeto precisavam para funcionar corretamente. Foi crucial para manter o ambiente de desenvolvimento organizado e íntegro.
- **PHPUnit:** Para garantir a qualidade e a confiabilidade da aplicação, utilizamos o PHPUnit. Ele é uma ferramenta de testes automatizados que nos permitiu criar e executar testes para verificar se as diferentes partes do código estavam funcionando como esperado. Isso ajudou a identificar e corrigir erros (bugs) logo no início do desenvolvimento, contribuindo para um software mais estável.

5. Desenvolvimento

Esta seção descreve o processo de desenvolvimento da aplicação de lista de tarefas, desde a concepção do esquema de banco de dados até a execução do ambiente.

- **Modelagem do Banco de Dados:** A fase inicial envolveu a definição das estruturas de dados necessárias para a aplicação. Foram projetadas duas tabelas principais: users para armazenar informações de cadastro e autenticação de usuários (e.g., id, email, password_hash, created_at, updated_at) e tasks para as tarefas individuais (e.g., id, user_id - chave estrangeira para users, title, description, status - booleano para "concluída" ou "pendente", created_at, updated_at). A gestão dessas estruturas foi realizada através do sistema de **migrations** do CodeIgniter, permitindo o versionamento do esquema do banco de dados e facilitando atualizações incrementais.
- **Implementação ORM:** Para a interação com o banco de dados, o framework CodeIgniter forneceu seu sistema de Mapeamento Objeto-Relacional (ORM). Isso nos permitiu manipular os dados como objetos PHP, evitando a necessidade de escrever consultas SQL diretamente. Por exemplo, para registrar um novo usuário,

um objeto User era criado e persistido via um método como $user->save(). De forma similar, para listar as tarefas de um usuário, métodos do ORM eram utilizados para buscar objetos Task associados ao user_id logado. A exclusão de tarefas e a alteração de seu status (marcando como completa/pendente) também foram realizadas através de chamadas aos métodos ORM correspondentes, simplificando as operações CRUD e tornando o código mais legível e padronizado.

- **Estrutura do Código:** O projeto seguiu rigorosamente o padrão arquitetural MVC (Model-View-Controller) proposto pelo CodeIgniter. Os arquivos foram organizados em diretórios específicos:
  - **app/Controllers**: Contém as classes que recebem as requisições HTTP, processam a lógica de negócio (interagindo com os Modelos) e preparam os dados para as Views. Por exemplo, AuthController.php para login e cadastro, e TasksController.php para gerenciar as tarefas.
  - **app/Models**: Abriga as classes que representam as entidades do banco de dados (User.php, Task.php) e encapsulam a lógica de interação com o banco de dados através do ORM.
  - app/Views: Contém os templates HTML que são renderizados e enviados ao navegador do usuário. Isso inclui formulários de login/cadastro e a interface principal para exibir a lista de tarefas.

    Essa separação clara de responsabilidades contribuiu para a modularidade e a facilidade de manutenção do código.
- **Configuração e Execução:** A configuração do ambiente de desenvolvimento foi simplificada pelo uso de Docker. Os passos para colocar a aplicação em funcionamento incluíram:
  1. **Criação do arquivo .env:** Um arquivo de variáveis de ambiente foi configurado para armazenar credenciais de banco de dados (e.g., DB_HOSTNAME, DB_DATABASE, DB_USERNAME, DB_PASSWORD) e outras configurações sensíveis, mantendo-as separadas do controle de versão.
  2. **Inicialização dos contêineres Docker:** Utilizou-se o comando docker-compose up -d (executado a partir da raiz do projeto, onde o docker-compose.yml está localizado) para levantar os serviços da aplicação (servidor web com PHP e banco de dados PostgreSQL).
  3. **Instalação de dependências:** O Composer foi utilizado para instalar as bibliotecas PHP necessárias ao framework e à aplicação com o comando composer install.
  4. Execução das Migrações de Banco de Dados: As tabelas do banco de dados foram criadas e atualizadas através do sistema de migrações do CodeIgniter, executando php spark migrate dentro do contêiner da aplicação.

     Após esses passos, a aplicação estava acessível e pronta para uso, demonstrando a eficiência da containerização para a padronização do

ambiente.

6. Resultados

Nesta seção, apresentamos os resultados obtidos com o desenvolvimento da aplicação, com exemplos de como o projeto é executado para demonstrar as funcionalidades implementadas.

- **Demonstração das Funcionalidades:** A aplicação pode ser acessada através de um navegador web após a execução dos comandos de inicialização Docker.
  - **Fluxo de Usuário:** Ao abrir o navegador e acessar a URL configurada, o usuário é direcionado para a página inicial.
  - **Cadastro de Usuários:** Caso não possua conta, o usuário pode clicar em "Registrar", preencher o formulário com email e senha, e submeter. Uma mensagem de sucesso ou erro é exibida.
  - **Login de Usuários:** Após o registro ou se já possui uma conta, o usuário insere suas credenciais. Um login bem-sucedido redireciona para a página principal da lista de tarefas.
  - **Registro de Tarefas:** Na página principal, um campo de texto e um botão "Adicionar Tarefa" permitem inserir novas tarefas. Ao submeter, a tarefa aparece na lista abaixo.
  - **Visualização de Tarefas:** Todas as tarefas cadastradas pelo usuário logado são exibidas em uma lista. Cada item de tarefa mostra seu título/descrição e o checkbox de status atual.
  - **Exclusão de Tarefas:** Ao lado de cada tarefa na lista, um ícone de lixeira ou botão "Excluir" permite remover a tarefa. Uma confirmação é exibida antes da exclusão definitiva.
  - **Marcação de Status:** Um checkbox ao lado de cada tarefa permite alternar seu status. A alteração é refletida visualmente e no banco de dados.

7. Reflexão sobre Desafios

Durante o desenvolvimento do projeto, diversos desafios foram encontrados e superados, contribuindo significativamente para o aprendizado e a robustez da solução.

- **Dificuldade na Inicialização do Projeto:** Problemas ao configurar o ambiente de desenvolvimento, especialmente com o Docker.
- **Problemas em Conexões com o Banco de Dados:** Falhas na comunicação entre a aplicação PHP e o banco de dados PostgreSQL.
- **Problemas com a Estruturação do Projeto:** Dificuldade para o framework localizar ou carregar arquivos corretamente devido a erros de nomenclatura ou configuração.

- **Problemas com a Execução de Algumas Queries:** Erros em operações de banco de dados, mesmo utilizando o ORM.
- **Problemas com o Cache ao Executar em um Computador Fora do Ambiente de Desenvolvimento:** Comportamento inesperado da aplicação em ambientes diferentes do de desenvolvimento devido a problemas de cache.

A superação desses obstáculos não apenas resolveu as questões técnicas, mas também aprofundou a compreensão sobre as ferramentas utilizadas e as boas práticas de desenvolvimento web.

7. Link para o vídeo explicando sobre o projeto

https://youtu.be/SZGj055Ch3k

8. Conclusão

Este projeto demonstrou a construção de uma aplicação de lista de tarefas, um esforço prático para aplicar e compreender os benefícios dos frameworks modernos no desenvolvimento web. A escolha do **CodeIgniter** pela sua agilidade, aliada à organização proporcionada pelo padrão **MVC**, otimizou o ciclo de desenvolvimento. A utilização do **ORM** simplificou a interação com o **PostgreSQL**, enquanto a **containerização com Docker e Docker Compose** assegurou a consistência do ambiente. Ao longo do processo, foram enfrentados e superados desafios significantes, desde a inicialização do projeto e problemas de conexão com o banco de dados, até a estruturação de arquivos e questões de cache em diferentes ambientes. O sucesso na implementação das funcionalidades de cadastro, login, registro, visualização, exclusão e marcação de status de tarefas, reforça a eficácia da abordagem baseada em frameworks. Em suma, este trabalho não apenas resultou em uma ferramenta funcional, mas também proporcionou um aprendizado valioso sobre os fluxos de desenvolvimento web e a superação de obstáculos práticos, consolidando o conhecimento sobre a construção de software robusto e de fácil manutenção.

9. Referências

- CODEIGNITER. **Documentação Oficial do CodeIgniter**. Disponível em: https://codeigniter.com/user_guide. Acesso em: 29 abr. 2025.
- MVC. **Model-View-Controller**. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2023. Disponível em: https://pt.wikipedia.org/wiki/Model-View-Controller. Acesso em: 04 mai. 2025.
- DOCKER. **Docker Documentation**. Disponível em: https://docs.docker.com/. Acesso em: 04 mai. 2025.
- GOMES, I. **Entendendo o Padrão MVC em Aplicações Web**. Medium, 2021. Disponível em:

https://medium.com/@ivan.n.gomes/entendendo-o-padr%C3%A3o-mvc-em-aplica%C3%A7%C3%B5es-web-a0e28f3a8b41. Acesso em: 04 mai. 2025.