php.com.br

7 ERROS COMUNS EM

INICIANTES PHP

PHP.COM.BR

INTRODUÇÃO

Você quer implementar e colocar em produção sua aplicação PHP sem dores de cabeça?

Aposto que você não gosta de perder horas e horas tentando encontrar um problema sem saber ao certo por onde começar a procurar. Tempo é dinheiro! Não esbarre em erros que poderiam ser facilmente evitados.

Nesta cartilha vamos apresentar 7 erros muito comuns que desenvolvedores PHP cometem nos primeiros contatos com a lingugem. Erros relacionados à configuração, gestão de ambientes, banco de dados, apresentação, lógica. Tenho certeza de que ao ler este material, você poupará tempo evitando cair nas mesmas armadilhas.

1. NÃO OLHAR OS LOGS DO JEITO CERTO

A maioria dos desenvolvedores de primeira viagem em PHP não dominam a configuração do ambiente, o que faz muita diferença. Para acelerar o processo, a maioria utiliza algum instalador (Wamp, Xampp), que já traz um PHP pré-configurado. Mas nem sempre a configuração de fábrica é a melhor, você precisa conhecer os detalhes, caso contrário poderá perder tempo. Caso a configuração esteja errada, você poderá deixar de ver erros durante o desenvolvimento, ou até vê-los em produção (quando não é recomendado).

O primeiro passo é localizar o PHP.INI correto. Incrivelmente, alguns instaladores trazem mais de um, em pastas diferentes, o que atrapalha bastante. Use a função phpinfo(); para saber a localização exata do PHP.INI. Verifique como estão as diretivas, e para que server cada uma. Configure as diretivas error_reporting, display_errors, error_log, e log_errors. Elas são muito importante para você ver erros em desenvolvimento, mas não em produção.

Em desenvolvimento use E_ALL no error_reporting, e ligue o display_errors. Em produção desligue NOTICE'S e DEPRECATED'S, deslique o display_errors, ligue os logs (error_log, log_errors). Está ocorrendo um erro que você não está localizando nem nos logs do PHP e está usando o Apache como servidor de páginas? Localize o error.log do Apache, lá tem muita informação útil (ex. Fatal errors).

Confira nossos artigo com instruções de instalação do PHP:

- https://php.com.br/instalacao-php-linux
- https://php.com.br/instalacao-php-windows
- https://php.com.br/instalacao-php-macos

Artigo sobre como encontrar os erros:
- https://www.adianti.com.br/forum/pt/view_5093).

2. IGNORAR O SERVIDOR DE APLICAÇÃO

Muitos desenvolvedores de primeira viagem ignoram o ambiente que está por baixo de onde a aplicação vai rodar, preocupando-se apenas com características da aplicação. Apesar de muitos desenvolverem em Windows, e a instalação de ambiente lá ser bastante facilitada, a maioria das aplicações PHP em produção hoje rodam em nuvem Linux.

O custo de hosting é muito mais vantajoso em Linux, as opções são muitas, e o PHP foi feito para rodar em ambientes Unix, onde é mais seguro e robusto, além de oferecer muitas ferramentas de virtualização e containers relacionadas. É importante você dominar a infraestrutura que está abaixo de sua aplicação. Saiba como instalar pacotes, pois quase todos módulos do PHP são representados por pacotes. Aprenda a habilitar e desabilitar pacotes do PHP no servidor.

O PHP roda em conjunto com um servidor de páginas (Ex: Apache, nginx). Aprenda as configurações do servidor de páginas. O apache, por exemplo, possui três modelos diferentes: prefork, event, worker. Além disso, possui módulos de segurança, anti-ddos (evasive), que são opcionais. Dependendo do tipo de aplicação, um modelo de execução é mais indicado que outro. Neste post, as diferenças entre eles (prefor, event, worker) são bem explicadas https://serverfault.com/a/383634.

Quer um tutorial bem completo sobre instalação de um servidor PHP em Linux: https://www.adianti.com.br/forum/pt/view_4402

Quer um tutorial bem completo sobre instalação de um servidor PHP em Windows: http://www.adianti.com.br/forum/pt/view_5094

3. NÃO GERENCIAR OS AMBIENTES

A maioria dos desenvolvedores de primeira viagem optam por desenvolver em Windows. É até compreensível, devido à facilidade de instalação de ferramentas, e uso do ambiente.No entanto, como já vimos, a maioria das aplicações em produção acaba rodando em Linux, devido à robustez superior, custo inferior, e maior quantidade de ferramentas. Desenvolver sobre o sistema de arquivos do Windows e publicar em Linux não é uma boa opção. Os sistemas de arquivos e de permissão nos dois sistemas é diferente.

Essas diferenças influenciam a aplicação, pois muitas delas precisarão gerenciar arquivos (upload, mover arquivos, criar diretório). Assim, sua implementação também possuirá diferenças. Como exemplo, podemos citar operações como conceder permissões de escrita em um arquivo ou pasta, que diferem de um sistema operacional para outro. Isso sem falar que o Linux é case-sensitive (diferencia maiúsculas e minúsculas), e o windows não. Isso é muito importante, e pode fazer toda a diferença entre um arquivo ser localizado ou não.

Caso a aplicação precise acessar caminhos, temos outros agravantes, como a diferenciação de barras (/, \) entre um sistema e outro. E se sua aplicação precisar executar um comando? A forma de executar comandos externos também varia. Ao utilizar um exec() por exemplo, os comandos no Linux são diferentes do Windows e residem em locais diferentes.

Mas qual a solução então? Seu ambiente de desenvolvimento deve ter a mesma configuração que o ambiente de produção. E para tal, existem ferramentas que facilitam muito essa aproximação. Mas como fazer isso? Virtualização (VirtualBox, vagrant), containers (docker). Assim, você poderá rodar a aplicação já no desenvolvimento um ambiente Unix-like, e não terá surpresas ao colocar a aplicação em produção.

4. IGNORAR O BANCO DE DADOS.

Outro fator ignorado por desenvolvedores de primeira viagem em PHP é o banco de dados. Muitos o abstraem de tal maneira a ignorar questões mais técnicas de performance. Se você trabalha em uma empresa maior, talvez você possui um DBA a disposição para cuidar do banco de dados para você. Mas se você está começando ou trabalha sozinho, precisa cuidar de tudo. Cuidar de tudo significa que você precisará aprender algumas coisas essenciais. A primeira delas é a configuração do banco de dados.

O PostgreSQL, por exemplo, o banco de dados open source mais robusto disponível atualmente, possui os arquivos de configuração postgresql.conf, e o pg_hba.conf. Enquanto que o postgresql.conf define a configuração do banco, o pg_hba.conf define regras de acesso (permissão) local ou remoto. É no postgresql.conf que você define o quanto de memória o banco de dados vai utilizar para operações. Então, não adianta você ter uma máquina com 16 Gb de RAM, e o banco estar configurado para ocupar 512Mb.

Outro fator muito esquecido, é a criação de índices. Muitos desenvolvedores criam a aplicação, e o banco de dados, mas esquecem de criar índices. Enquanto que a aplicação possui poucos registros, a performance é razoável, mas a medida em que ela vai crescendo, vai degradando a performance. Na dúvida, crie índices para os campos de chave estrangeira, e qualquer campo buscável em WHERE. Isso fará a performance não degradar com uma quantidade maior de registros. Evite criar índices em tabelas de logs, pois como elas sofrem muito insert e não devem degradar a perforamnce da operação principal, o índice deixa lenta a inserção.

5. MISTURAR LÓGICA DE NEGÓCIOS COM APRESENTAÇÃO

Quando estamos começando, não sabemos que precisamos separar as coisas. Então é comum criarmos tudo misturado (PHP, HTML, SQL). Um exemplo é misturar lógica com apresentação. Quem nunca criou aquela regra de negócio que verifica por exemplo, se o cliente está inadimplente, e caso esteja, emite um HTML com uma mensagem de erro?

A lógica de negócios normalmente reside no domínio, ou core domain. O domínio é a camada onde temos as classes que representam as entidades da aplicação e as regras de negócio mais importantes. Classes que processam lógica de negócio são core domain, e como tal, não devem depender de detalhes técnicos. Interface é um detalhe técnico. Classes que processam lógica devem fazer coisas, ou retornar coisas, e caso ocorram falhas, elas devem lançar exceções. Exceções são a melhor forma de lidar com eventos inesperados.

Exceções devem ser tratadas nas camadas externas ao core domain. Classes de controle podem pegar uma exceção lançada pelo modelo e decidir como enviar isso para a fronteira com o usuário (interface).

Use exceptions! Permita-se utilizar comandos como o die() apenas para debug, jamais permita-se manter um comando como esse em sua aplicação, pois ele quebra o fluxo de execução da aplicação, na versão em produção.

Lembre-se de que você pode utilizar o mesmo core domain para Web, Linha de comando, ou Web Services. Então, ao não utilizar comandos de apresentação do domínio, naturalmente, você fará um modelo que poderá ser reutilizado.

6. IMPLEMENTAR REGRAS EM SQL NO CÓDIGO

Muitos desenvolvedores de primeira viagem conhecem razoavelmente SQL, mas possuem certas dificuldades em realizar as mesmas operações na aplicação usando a Orientação a Objetos. Esses medos acabam fazendo com que o desenvolvedor reproduza na aplicação, as mesmas consultas que faz na base de dados, muitas vezes essas consultas ficam gigantes.

Hoje em dia existem técnicas apuradas que permitem que se escreva muito pouco ou quase nenhum SQL na aplicação. Basta nos reprogramarmos para usar interfaces Orientadas a Objetos para manipulação de dados. Os frameworks implementam técnicas ORM (mapeamento objeto-relacional), que permitem que você interaja com a base de dados de maneira orientada a objetos. O Laravel e o Adianti Framework são exemplos disto.

E o melhor disso tudo, é que o Framework cuida de montar as consultas para você, utilizando as melhores técnicas (ex. prepared statements), evitando problemas como SQL Injection. Se você quiser aprender, aí vão alguns padrões: Data Mapper, Active Record, Row Data Gateway, Repository. Estes padrões te mostram como construir uma API orientada a objetos para interagir com o banco de dados.

7. MISTURAR A APLICAÇÃO COM DETALHES TÉCNICOS

Na pressa do dia a dia, muitos desenvolvedores são tentados a importar bibliotecas e sair utilizando. Bibliotecas para renderização de templates, para envio de e-mails, geração de gráficos, integração com redes sociais, dentre outros. Mas lembre que a melhor biblioteca hoje pode não ser a melhor biblioteca amanhã. Elas evoluem rapidamente, e você desejará substituir ela em breve quando uma nova opção surgir. Lembre-se de que bibliotecas (geração de gráficos, envio de e-mail, autenticação com rede social) são detalhes técnicos para a aplicação, e detalhes devem ser abstraídos e facilmente substituídos em qualquer momento.

Quanto mais o uso biblioteca se espalhar na aplicação de maneira direta, pior. Quanto mais uso direto, mais pontos de manutenção você terá no futuro quando quiser fazer uma manutenção e substituição. Detalhes técnicos são mecanismos de baixo nível, e como tal, devem ser passíveis de substituição a qualquer momento. Utilize padrões de projeto que privilegiem a substituição de partes, tais como: Facade, Wrapper, Injeção de dependência.

Além disse não amarre o núcleo de sua aplicação com detalhes técnicos. O que queremos dizer com isso? Se você fechou uma venda e quer enviar um e-mail automaticamente ao final do processo, não deixe esta chamada fixa a partir do Core Domain. Não acione classes de baixo nível (detalhes técnicos) diretamente a partir de classes do núcleo do negócio (core domain). Não faça o Core depender de detalhes. Essa amarração vai existir em uma camada mais externa (ex. controllers), usando mecanismos como Injeção de dependência. Prefira sempre dependências abstratas no lugar dependências concretas. Prefira não depender de componentes específicos, principalmente de componentes externos. Utilize interfaces, containers, e injeção de dependência para evitar essas "amarras".


php.com.br