PHP ou TypeScript: uma comparação de duas linguagens para web pelas suas características

Trabalho de Conclusão do Curso Superior de Tecnologia em Sistemas Para Internet

Jean de Oliveira Lopes
Orientador(a): Fabio Okuyama

[1]Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Campus Porto Alegre
Av Cel Vicente, 281, Porto Alegre – RS – Brasil

jeanoliveiralopes@gmail.com, fabio.okuyama@poa.ifrs.edu.br

Resumo. *O Artigo pretende trazer análises de duas linguagens, sendo estas PHP e TypeScript, e fazer uma breve comparação entre elas, trazendo pontos de vistas de diversos autores da literatura e abordando seu uso em um mesmo cenário. Comparação essa foca na produtividade, propensão a erros, bem como os paradigmas de cada uma e seus sistemas de tipos.*

1. Introdução

O propósito deste artigo foi comparar duas linguagens de programação em um mesmo cenário através de suas características para evidenciar elementos que favoreçam a escolha mais adequada. A primeira, o PHP, representando uma linguagem mais tradicional, enquanto a segunda, o TypeScript, ocupa o lugar de uma linguagem mais emergente. Segundo (Reghunadh e Jain, 2011) nenhuma linguagem de programação é a "melhor escolha" e não existe apenas um fator para se pensar ao fazer tal escolha.

De acordo com (Britton, 2008), existem características técnicas em abundância para se comparar linguagens de programação. Ao longo do artigo estas linguagens são brevemente apresentadas, ainda existe a possibilidade de conhecer características de ambas e essas características são confrontadas, com intuito de verificar o quanto cada linguagem pode ser útil para um mesmo cenário. Para confrontar as mesmas, o artigo traz da literatura alguns critérios de comparação. A partir destes critérios, são trazidas comparações já alçadas pela literatura para tais linguagens escolhidas. Ao final do artigo é exposto o protótipo de uma API do módulo de um sistema hipotético de venda de passagens, para servir de exemplo de cenário onde ambas estão sendo usadas. A partir dessa API, o artigo cria um vínculo entre as características abordadas até então e o código desenvolvido pelas duas diferentes linguagens com o intuito de corroborar com as conclusões já existentes de outros autores.

2. Contexto

O artigo traz duas linguagens de programação e considera uma tradicional e outra emergente. PHP é dita neste artigo como exemplo de uma linguagem popular. A figura 1 mostra os resultados do Google trends para as linguagens de programação PHP e TypeScript. Em vermelho está o PHP e em azul o TypeScript. Os números representam

o interesse de pesquisa relativo ao ponto mais alto no gráfico de uma determinada região em um dado período. Um valor de 100 é o pico de popularidade de um termo. Um valor de 50 significa que o termo teve metade da popularidade. Da mesma forma, uma pontuação de 0 significa que o termo teve menos de 1% da popularidade que o pico.

Figura 1. A classificação do Google Trends.

2.1 A escolha de uma linguagem de programação

Há milhares de linguagens de programação, e novas são criadas sucessivamente. Por isso, pode ser difícil determinar a melhor linguagem para escrever um programa (Lee, 2012). (Zapalowski, 2011) afirma que existem mais de 8000 linguagens de programação, e em pouquíssimas delas são realizados estudos para obtenção de métricas que forneçam dados relevantes sobre as implementações que nelas foram feitas. (Reghunadh e Jain, 2011) diz que durante o planejamento de uma solução de software, a escolha de uma linguagem de programação pode depender de muitos fatores. Alguns desses fatores são a plataforma direcionada, a extensibilidade da linguagem, o tempo de produção, a performance, o suporte e a comunidade. (Briton, 2008) chama alguns desses fatores de características técnicas. Ele também sugere que o foco da escolha dos critérios de comparação durante a busca pela linguagem mais adequada sejam aqueles critérios de resultado, como facilidade de aprendizado, performance, portabilidade, entre outros.

2.2 PHP

Segundo (ZEND, 2016), o próprio site que mantém a linguagem, o PHP é uma linguagem de *script open-source* de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento web e que pode ser embutida dentro do HTML. A primeira versão do PHP veio a público em 1995, pelo seu criador, Rasmus Lerdorf, e a partir daí foi continuamente aprimorada. A linguagem foi amplamente utilizada para web, sendo uma das pioneiras, é usada até hoje. Interpretada, com suporte a orientação a objetos, e hoje, com até suporte a aspectos do paradigma funcional, caracteriza-se por ser manipulada no servidor e gerar código HTML ou similares para o lado cliente. O PHP herda a mesma sintaxe de C, e suas variáveis são prefixadas com $ (cifrão). Ele possui arrays associativos, estruturas de controle como if, foreach e while. A linguagem também não é sensível ao caso. Uma das principais características do PHP é que sua

tipagem é dinâmica e fraca, ou seja, possibilita a mudança do tipo das variáveis em tempo de execução. Como linguagem já consolidada, possui uma vasta comunidade e documentação em abundância na internet.

2.3 TypeScript

A despeito do seu sucesso, JavaScript tornou-se uma linguagem fraca para o desenvolvimento e manutenção de grandes aplicações. TypeScript é uma extensão do JavaScript com intenção de suprir sua deficiência. Sintaticamente, TypeScript é um superset[1] do Ecma-script5[2], então todo programa JavaScript é um programa TypeScript. TypeScript agrega ao JavaScript um sistema de módulos, classes, interfaces, e um sistema de tipagem estática. Como o TypeScript visa prover uma leve assistência aos programadores, o sistema de módulos e sistema de tipos é fácil de usar. Em particular, ele suporta muitas práticas comuns ao JavaScript. Ele também possibilita o uso de código assistido por parte das IDEs[3], o que sempre foi muito comum em linguagens como o C# e o Java. (Bierman, Gavin, Martín Abadi, Torgersen, 2014).

2.4 Conceito de Sistema de Tipos e Checagem de Tipos

Um dos pilares fundamentais das linguagens de programação pode ser o sistema de tipos que nela está embutido. Quando lidamos com variáveis, precisamos que ela assuma, num determinado instante, um tipo definido, para que possamos interagir com ela, assim trabalhando com outras variáveis de tipos compatíveis. Segundo PIERCE (2002), um sistema de tipos é um método sintático tratável para provar a isenção de certos comportamentos num programa através da classificação de frases de acordo com as espécies de valores que elas suportam. Ainda, segundo o autor, este, que pode ser estático ou dinâmico, define em que momento o tipo de uma declaração deve ser definido. Como aponta (SOUSA, FIGUEIREDO e VALENTE, 2013), Declarações em linguagens com tipagem estática, como Java e C#, devem ser acompanhadas pela definição de um tipo, que pode ser usado pelo compilador para checar a corretude do código. Já em linguagens dinamicamente tipadas, como Ruby e JavaScript, a definição do tipo só e realizada em tempo de execução. Mas como indicar a diferença da performance de desenvolvimento que o programador pode ter usando uma linguagem com tipagem estática ou dinâmica. O grande problema a ser exposto aqui é que um sistema de tipos ajuda a indicar, com auxílio da IDE, o tipo a ser passado como parâmetro, retorno, ou instanciado. Com tipagem estática, podemos usar interfaces e definir famílias de objetos estaticamente tipados em tempo de compilação, para que assim possamos programar de forma mais ágil, facilitando a leitura e a manutenção e diminuindo a quantidade de documentação necessária para se escrever o código.

---

1 Um conjunto que inclui outro conjunto ou conjuntos

2 Uma linguagem de programação baseada em scripts, padronizada pela Ecma International na especificação ECMA262. A linguagem é bastante usada em tecnologias para Internet, sendo esta base para a criação do JavaScript/JScript e também do ActionScript.

3 Ambiente de desenvolvimento integrado.

Segundo (MEIJER e DRAYTON, 2004), "os problemas em torno de uma linguagem híbrida entre tipagem dinâmica e estática são largamente mal compreendidos, e ambos lados geralmente usam argumentos fracos para justificar o seu lado". Linguagens de tipagem estática possuem recursos numerosos, como inferência de tipos[4], subtipos coercivos[5], tipos genéricos[6] e covariância[7]. Linguagens com tipagem dinâmica, por serem mais simples, permitem que programadores executem suas tarefas de desenvolvimento mais rapidamente (PIERCE, 2002). De acordo com (TRATT, 2009), ao removerem o trabalho de declarar os tipos das variáveis, estas linguagens permitem que seus usuários foquem no problema a ser resolvido, ao invés de se preocuparem com as regras da linguagem. Dentre os benefícios de uma tipagem dinâmica, ainda podemos encontrar recursos como *Lazy Evaluation*[8], funções de primeira classe[9], serialização[10] e *code literals*[11]. (MEIJER e DRAYTON, 2004) ressalta o uso de linguagens de tipagem estática sempre que possível, e de linguagens de tipagem dinâmica apenas quando necessário.

2.5 Paradigmas de Programação

O paradigma de programação é uma forma de utilizar a linguagem de programação, regida por certas regras que delimitam e organizam seu uso, bem como disposição do código, seguindo uma linha de raciocínio que é norteada pelo próprio conceito do paradigma em si. Um paradigma de programação pode combinar dois ou mais paradigmas para potencializar as análises e soluções (Jungthon e Goulart, 2008).

Existem muitos paradigmas de programação, dentre eles, o imperativo, estruturado, funcional, lógico e orientado a objetos. Cada um possui soluções para tipos de metodologias de desenvolvimento diferentes, focadas em problemas diferentes. Nem sempre um único paradigma vai resolver o problema de uma aplicação em si. Nem sempre uma única linguagem terá todos os paradigmas suficientes para se desenvolver a aplicação no nível de complexidade desejado.

A escolha de um paradigma influencia de maneira crucial o desenvolvimento da aplicação web. Muitos critérios podem ser levados em conta na escolha do paradigma.

---

4 Permite omitir o tipo da variável no momento da declaração.

5 É uma técnica de polimorfismo de tipos na qual um subtipo é um tipo que é relacionado a outro tipo (o supertipo) pela mesma noção de substituibilidade, significando que elementos do programa, tipicamente subrotinas ou funções, escritos para operar em elementos do supertipo podem também operar em elementos do subtipo (Pierce, 2002).

6 Classe ou interface que é parametrizada com tipos.

7 Técnica de programação que permite passar um tipo mais específico quando um tipo mais genérico é esperado, quando instanciando uma variável.

8 Conceito da programação em que o runtime da aplicação adia a avaliação de um determinado trecho de código até quando esse código realmente precisa ser executado.

9 Funções que operam sobre outras funções, ou sendo passadas como argumento ou como retorno.

10 Processo de transformação de objetos ou estruturas de dados em um formato que pode ser estaticamente armazenado.

11 Técnica em que o código que pode ser passado e processado em tempo de execução.

Um deles, são os recursos humanos. É necessário muitas vezes, que previamente a equipe de programadores tenha um conhecimento prévio do paradigma escolhido e saiba usá-lo de maneira adequada. Um paradigma de programação pode possuir uma curva de aprendizado lenta. Muitas vezes maior que a própria linguagem de programação que se utiliza dele. Certos paradigmas como lógico e funcional são mais voltados para resolver problemas científicos e matemáticos, fornecendo alta disponibilidade e um bom nível de tolerância a falhas. Outros, como o orientado a objeto, não tem a maior performance, conforme (WSEAS, 2005), mas são úteis para compreensão humana e facilitam a manutenção do sistema, uma vez que exprimem de maneira quase totalmente satisfatória uma abordagem do mundo real.

Os paradigmas tratados aqui são, principalmente o orientado a objetos e o estruturado. O paradigma funcional será levemente abordado. É um objetivo claro trazer vantagens do paradigma orientado a objetos e compará-lo ao sistema que não faz uso dele.

3. Metodologia

A metodologia de desenvolvimento começou com um estudo bibliográficos de artigos e manuais sobre linguagens de programação. Aliado a este, seguiu-se com um estudo sobre os artigos que fazem comparações entre as linguagens. Resolveu-se destacar como critérios de avaliação para a comparação das linguagens de programação os mesmos trazidos pela literatura referenciada no artigo que comparou as linguagens aqui apresentadas, esses critérios são discutidos na seção 3.1. Em seguida é apresentado um artigo que traz uma comparação feita com diversas linguagens, entre elas, o PHP e o TypeScript. O artigo de (Ray e Baishakhi, 2014) tem como objetivo apontar a linguagem mais propensa a erros, associando suas propriedades e as separando em categorias. Além deste artigo, é discutido um critério trazido na comparação, que é a checagem de tipos. Em seguida foi definido um cenário de aplicação que originou uma API desenvolvida duas vezes, em PHP e TypeScript, respectivamente. São destacados alguns erros que ocorreram durante a programação das duas APIs. A causa e impacto destes erros é discutida e procura-se evidenciar se a checagem de tipos pode influenciar a ocorrência de tais erros. Assim a implementação serve para verificar as principais diferenças apontadas, utilizando assim os critérios levantados e indicando assim o que seria supostamente mais adequado para o cenário.

3. 1 Critérios de comparação escolhidos

Os critérios escolhidos foram os seguintes: paradigma, tipagem forte ou fraca, tipagem dinâmica ou estática, gerenciamento da memória, propensão a erros e tamanho do código. Estes critérios foram escolhidos porque segundo (Ray e Baishakhi, 2014) eles influenciam na qualidade do código. Destes, o paradigma é conceituado na seção 2.5, enquanto a tipagem forte ou fraca, e dinâmica ou estática é conceituada na seção 2.4. O gerenciamento de memória pode ser automático ou manual em algumas linguagens de programação, (Knuth, 1997) explica que este processo trata-se de alocar memória durante o tempo de execução do programa, usando-a, e a liberando após o seu uso. Esse processo é controlado em algumas linguagens de programação e outras não. Trazendo algumas vantagens e desvantagens para ambas situações. Há mais controle dos recursos

de hardware disponibilizados quando a memória é manualmente gerenciada, entretanto, quando a memória é automaticamente gerenciada o programador pode focar mais seus esforços no domínio do problema. A propensão a erros e o tamanho do código são critérios que podem ser comparados após o experimento ser efetivado, e como aponta (Ray, Posnett, Filkov, Devanbu, 2014), são indicativos da qualidade do código desenvolvido. Segundo (Britton, 2008), esses são critérios de resultado. Também são critérios trazidos pela literatura aqui apresentada e medidos nos experimentos abordados por ela.

3.1 A API desenvolvida

O modelo de sistema implementado foi somente um módulo de gerência de itinerários para a venda de passagens de uma rodoviária. Com ele pode-se cadastrar, visualizar, editar ou remover um itinerário de uma lista de itinerários. A API guarda no banco de dados em mysql os itinerários cadastrados. A API foi desenvolvida duas vezes, uma em TypeScript e a outra em PHP. Os erros que surgiram durante o desenvolvimento das duas APIs foram coletados para poder-se analisar os mesmos.

4. Resultados trazidos da literatura

Foi realizada uma pesquisa no github intitulada *A Large Scale Study of Programming Languages and Code Quality in Github* (Ray, Posnett, Filkov, Devanbu, 2014) que envolveu algumas linguagens de programação, dentre elas, o PHP e o TypeScript. Segundo os autores, a pesquisa não é um experimento controlado, isto é, os desenvolvedores não foram monitorados enquanto programavam nas diferentes linguagens de programação. Os pesquisadores então não puderam comparar resultados como o esforço do desenvolvimento e qualidade do programa.

O estudo começou com o a escolha das 17 linguagens de programação mais populares do github, analisando um total de 850 projetos. Linguagens que não foram consideradas de propósito geral foram descartadas. Para a coleta desses projetos, foi usado o GitHub Archive, um banco de dados que armazena todas as atividades de projetos públicos do Github. A próxima etapa foi escolher os critérios de comparação. Esses critérios foram os mesmos escolhidos no presente artigo: paradigma de programação, tipagem estática ou dinâmica, tipagem forte ou fraca, e memória gerenciada ou não gerenciada. Esses critérios são os mesmos abordados no presente artigo. Com esses critérios, foi possível dividir as 17 linguagens em classes diferentes: *Functional-Static-Strong-Managed*, *Functional-Dynamic-Strong-Managed*, *Procedural-Static-Strong-Managed*, *Script-Dynamic-Strong-Managed*, *Script-Dynamic-Weak-Managed*, *Procedural-Static-Weak-Unmanaged*. Em seguida, os autores identificaram os domínios dos projetos, como: aplicação, banco de dados, analisador de código, middleware, biblioteca, framework, e outros. O quadro 1 explica detalhadamente as classes das linguagens definidas por (Ray, Posnett, Filkov, Devanbu, 2014). O PHP pertence à classe *Script-Dynamic-Weak-Managed,* enquanto o TypeScript não pertence a nenhuma delas. Os motivos são explicados na seção 4.4.

Quadro 1. Classes de linguagens de programação, segundo (Ray, Posnett, Filkov, Devanbu, 2014)

Functional-Static-Strong-Managed	Paradigma funcional, tipagem estática em tempo de compilação, tipagem forte e memória auto gerenciada.
Functional-Dynamic-Strong-Managed	Paradigma funcional, tipagem dinâmica em tempo de compilação, tipagem forte, memória auto gerenciada.
Procedural-Static-Strong-Managed	Paradigma procedural, tipagem estática em tempo de compilação, fortemente tipada, memória auto gerenciada.
Script-Dynamic-Strong-Managed	Paradigma de *script*, tipagem dinâmica em tempo de compilação, fortemente tipada, memória auto gerenciada
Script-Dynamic-Weak-Managed	Paradigma de *script*, tipagem dinâmica em tempo de compilação, tipagem fraca, memória auto gerenciada.
Procedural-Static-Weak-Unmanaged	Paradigma procedural, tipagem estática em tempo de compilação, tipagem fraca, memória gerenciada manualmente.

Após dividir os projetos pelos seus domínios, os autores categorizaram os bugs que estavam relacionados a eles, atribuindo-lhes uma causa e um impacto. Da causa foram elencadas sub-categorias: algoritmos, concorrência, memória, erro genérico de programação e desconhecido. Para equiparar os projetos, foi utilizado o método de regressão binomial negativa. A estatística se deu por rastrear e quantificar os *commits*[12] defeituosos, assim associando o impacto das linguagens de programação para medir quais menos propensas a erros. Para isso foram usadas técnicas de processamento de linguagem natural nos logs dos commits.

Como resultado, os autores chegaram à conclusão de que algumas linguagens têm uma maior associação com defeitos do que outras, contudo, o efeito é pequeno.

12 Quando se utiliza a técnica de controle de versão, um commit é o passo de tornar permanente uma adição ou alteração no código fonte tido como referência no desenvolviemento de um dado projeto.

Dentre as médias atingidas e seus coeficientes de variação, as linguagens Clojure, Haskell, Ruby, Scala e TypeScript, todas tiveram coeficiente negativo, implicando que estas linguagens têm médias menos prováveis para os resultados dos commits defeituosos. Destas, o TypeScript obteve o coeficiente de variação mais baixo (-0.43), tornando-se a média menos provável. Ainda assim, PHP e TypeScript tiveram médias próximas (0.06 e 0.05).

Outro resultado observado foi que há uma pequena porém significante relação entre a classe da linguagem e seus defeitos. Linguagens funcionais têm uma menor relação com os defeitos do que linguagens procedurais ou de script. Também foi verificado que não há relação geral entre domínio e propensão a erro da linguagem. Outro fato observado foi que os tipos de defeitos estão fortemente associados com as linguagens. A linguagem é mais importante para categorias específicas do que para defeitos em geral.

(Hanenberg, 2010) comparou tipagem estática e tipagem dinâmica monitorando 48 programadores durante 27 horas enquanto desenvolviam um parser[13]. Não foi encontrada diferença significante na qualidade do código de ambas linguagens. Contudo, a linguagem de tipagem dinâmica foi desenvolvida em menos tempo. Seu estudo foi conduzido com estudantes de graduação em um laboratório com linguagens e IDE pré estabelecidos.

4.1 Resultados obtidos com a API.

A API foi desenvolvida como experimento para este artigo. Ela foi desenvolvida primeiro em PHP, após em TypeScript. Para infraestrutura no PHP, utilizou-se seu servidor web embutido. Para infraestrutura no TypeScript, utilizou-se nodejs para rodar o JavaScript transpilado como resultado. O banco de dados utilizado foi *MySql*, em um sistema operacional Ubuntu.

4.1.1 O desenvolvimento do servidor em PHP

Segundo (ZEND, 2016), o PHP é uma linguagem interpretada, que pode ser manipulada pelo servidor Apache[14], através de um módulo, que pode ser instalado e habilitado no mesmo. PHP também possui um servidor embutido em seu executável, o qual foi usado neste experimento. O servidor ficou por conta de dois arquivos com responsabilidades diferentes: itinerario.php e respository.php. Foi usada a versão mais recente do PHP (7.0.15) enquanto eram desenvolvidos os protótipos.

O arquivo itinerario.php contém os *endpoints*[15] que recebem pedidos e enviam respostas. O arquivo precisa incluir repository.php e executar suas funções, que são

13 Processo de analisar uma sequência de entrada (lida de um arquivo de computador ou do teclado, por exemplo) para determinar sua estrutura gramatical segundo uma determinada gramática formal.

14 Servidor HTTP open-source para sistemas operacionais como UNIX e Windows

15 URL onde seu serviço pode ser acessado por uma aplicação cliente.

responsáveis por consultar o banco de dados e manejar sua conexão. O código tem pouco menos de 50 linhas. O PHP provê funções suficientes embutidas na linguagem para fazer a maior parte das tarefas. Existem funções de conversão para string e JSON[16] embutidas na linguagem que auxiliam muito na hora de converter os dados recebidos pelo cliente e enviados para o banco. O formato trafegado do lado cliente para o lado servidor foi o x-www-form-urlencoded. Esse tipo de dado é um *string* padronizado no seguinte formato: "nomedavariavel=valor&nomedavariavel=valor…". O PHP recebe esses valores como array em variáveis globais de servidor. Esse array pode ser facilmente manipulável e tratável. É possível passar o array como argumento para funções que vão inserir esses dados no banco. Os arrays contém os nomes das variáveis como índice assim como os seus conteúdos como valores do array. Por outro lado, pode ser um pouco mais complexo lidar com certos tipos de variáveis como buffers ou arquivos fragmentados. Como observado durante o desenvolvimento e teste do programa, o PHP possui em si um fluxo fácil de manipular dentro do seu contexto de execução. Uma página pode ser invocada e seu script é lido do início ao fim. As variáveis globais como $_GET e $_POST contém o pedido e as funções de saída como print ou echo fazem com que o conteúdo impresso siga para o fluxo de resposta.

Com um circuito simplificado, não existe uma forma tão organizada de manipular o pedido e a resposta. Torna-se mais complicado manipular o código de status da resposta, cache ou cookies. O php não é orientado a eventos e a resposta sempre espera pelo resultado do processamento do código no servidor. Por não ser orientado a eventos, o ciclo de vida do PHP começa com o pedido e termina com a impressão da resposta.

Resolveu-se no trabalho simular uma API que trata os pedidos de acordo com os verbos HTTP. Esse modelo de aplicação foi desenvolvido seguindo a arquitetura de um web service. Segundo (Papazoglou, 2008), web services são uma forma revolucionária de disponibilizar conteúdo online, onde a informação pode ser acessada de forma automatizada, bem como por seres humanos. Isso serviu de motivação para criar não simples páginas dinâmicas, mas uma aplicação como um serviço. Porém, não existe uma forma padronizada no PHP para descobrir qual o método do pedido. Contudo, foi possível criar uma API que respondesse todos os verbos HTTP requisitados, podendo-se assim, fazer pedidos por GET, POST, PUT ou DELETE.

O arquivo repository.php ficou responsável por manipular o banco de dados. Foi aproveitada uma biblioteca embutida durante a instalação da linguagem: mysqli. Essa biblioteca é chamada no manual do PHP de extensão e provê um driver para a comunicação com o MySQL 4.1 ou superior (ZEND, 2016). O arquivo é pequeno e contém em torno de 100 linhas de código, com funções relativamente pequenas. Não existem muitas regras de negócio e o principal objetivo do código do arquivo é fazer as chamadas da biblioteca *mysqli* e tratar as repostas do banco de dados de forma adequada. Existe uma fronteira entre o código e o banco de dados, onde os tipos das

16 Formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar. Está baseado em um subconjunto da linguagem de programação JavaScript, Standard ECMA-262 3a Edição -Dezembro – 1999.

variáveis que serão inseridas no banco precisam estar de acordo com o tipo nas tabelas do banco. As respostas das consultas ao banco de dados eram arrays fáceis de tratar e manipular.

4.1.2 Erros e dificuldades encontrados no desenvolvimento do PHP

Alguns problemas de sintaxe foram descobertos quando começaram os testes com o lado servidor. O lado servidor foi testado separadamente em ambos os protótipos através de um pequeno script em *Node.js* que fazia os pedidos. Isso foi uma forma de simplificar o desenvolvimento sem precisar da interface pronta para testar o funcionamento do lado servidor.

Quadro 2. Erros encontrados durante o desenvolvimento do código PHP

Invocando GET vazio /itinerario.php - syntax error, unexpected 'return' (T_RETURN) in /home/jean/workspace/TCC/prjphp/server/repository.php on line 14 - Erro de sintaxe
Uncaught Error: Class 'mysqli_connect' not found in /home/jean/workspace/TCC/prjphp/server/itinerario.php:56 - Erro instanciando uma função;
Uncaught Error: Call to a member function query() on null in /home/jean/workspace/TCC/prjphp/server/repository.php:9 - Problema de escopo de variável; A variável mysqli foi declarada e iniciada no mesmo arquivo, mas não é reconhecida dentro das funções do mesmo arquivo; http://php.net/manual/pt_BR/language.variables.scope.php
Invocando o POST - É complicado prever o retorno de $mysqli->query(). O retorno da variável é misto, tornando assim, difícil o tratamento. Pode ser um valor booleano, como também um objeto mysqli_result. Neste caso, é complicado até mesmo navegar pela documentação do php para conhecer as propriedades do objeto. http://php.net/manual/pt_BR/mysqli.query.php
Invocando o GET com Id - Uncaught Error: Call to undefined method mysqli_result::fectch_assoc() in /home/jean/workspace/TCC/prjphp/server/repository.php:40

é necessário cuidado redobrado com a pronúncia do nome dos métodos, pela falta do código assistido.
Invocando o PUT Uncaught Error: Call to a member function bind_param() on boolean in /home/jean/workspace/TCC/prjphp/server/repository.php:75 É necessário checar o manual para descobrir o uso correto do método stmt->bind_param()
Invocando o Delete Não existe uma padronização para o verbo delete Call to a member function bind_param() on null in /home/jean/workspace/TCC/prjphp/server/repository.php:97
Não foi declarada a variável stmt no retorno de $mysqli->prepare
Era necessário usar o método fetch_all ao invés de fetch_assoc

Esses erros descritos no quadro 2 só foram descobertos quando o arquivo PHP foi invocado, manifestando-se assim, somente em tempo de execução. O uso do manual do PHP foi usado de forma extensiva, mesmo apesar de ter colhido exemplos de trechos de código na web para criar os arquivos.

4.2 O desenvolvimento do servidor em TypeScript

O TypeScript é uma linguagem com tipagem estática e com um grande suporte para a orientação a objetos. Ainda assim não é uma obrigatoriedade escrever em TypeScript usando puramente a orientação a objetos. O TypeScript é uma linguagem traspilada[17] para JavaScript. O programador precisa ter em mente que em tempo de execução a linguagem que estará sendo interpretada será o JavaScript, isto é, TypeScript é só um artifício para se programar com um uma checagem de tipos e uma sintaxe mais moderna. Seus arquivos ficaram ligeiramente menores, se comparados com os arquivos PHP, no entanto, surgiram mais arquivos com classes para lidar. A figura 2 mostra como ficou o diagrama de classes do servidor desenvolvido em TypeScript, utilizando orientação a objeto.

17 Termo especifico para o código escrito em uma linguagem e transformado em outra linguagem que tem um nível similar de abstração.

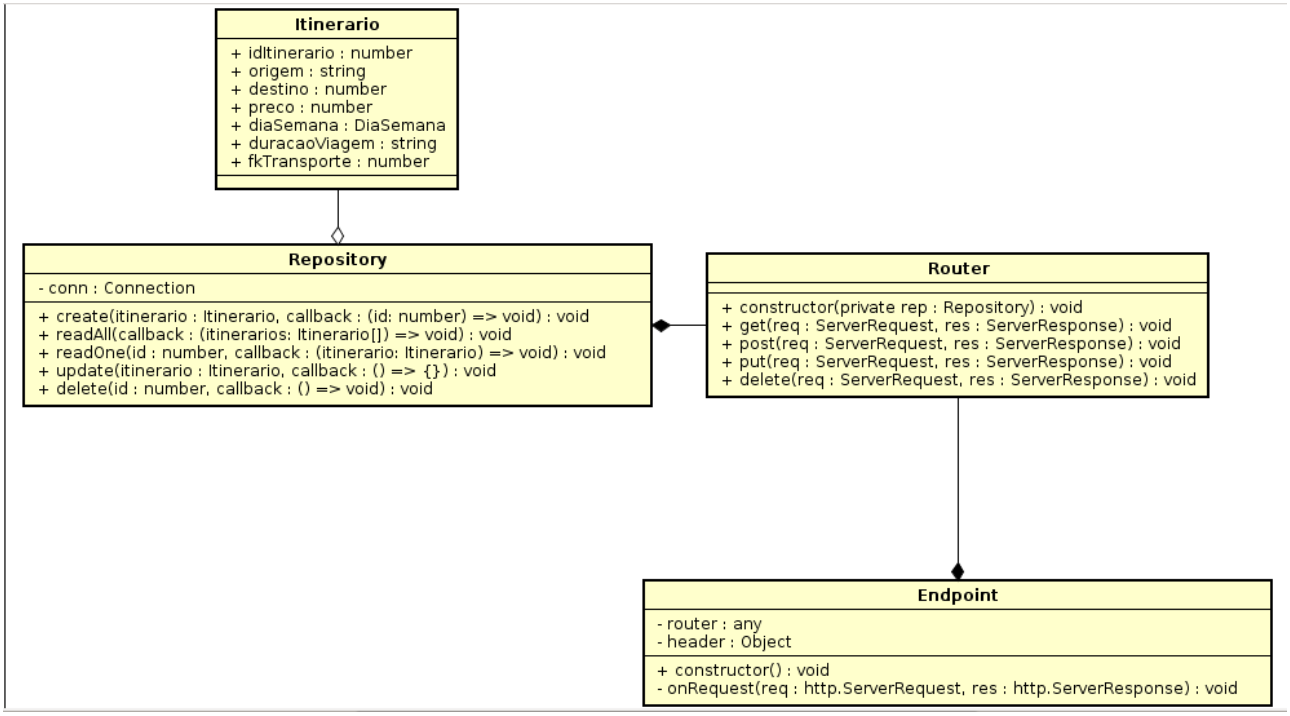

Figura 2. O diagrama de classes do servidor

Foi criado um módulo principal chamado main.ts, que invoca a classe principal responsável por acionar os endpoints, que receberão os pedidos e as respostas. O NodeJS, ao contrário do PHP, é totalmente orientado a eventos, e as funções callback fazem parte da sua cultura (Buna, 2017). O NodeJS recebe os pedidos e as respostas num mesmo contexto e não para o processamento interno antes de enviar a resposta.

Fazendo uso da orientação a objetos, foi criada a classe Rotas.ts, que é responsável por tratar cada verbo HTTP que possa haver. Esses métodos são disparados através de eventos. Foi necessário usar um recurso chamado reflexão, que atualmente não há no TypeScript. Para contornar o problema, foi observado que o TypeScript é livre para se misturar com a sintaxe do JavaScript e seus recursos, sempre que necessário.

A classe Repository foi responsável por fazer as consultas ao banco de dados. Para acessar o banco de dados foi utilizada uma biblioteca externa, o @types/mysql, que é um *wrapper*[18] do pacote MySql para NodeJS, com a interface para o TypeScript. Procurou-se evitar ao máximo usar bibliotecas externas, ou frameworks, para poder ter uma ideia aproximada da produtividade da linguagem individualmente, mesmo assim, algumas foram usadas, tanto no PHP, quanto no TypeScript, para manipular o banco de

18 Adaptador.

dados. A classe Repository manipulou a biblioteca e recebia funções de *callback* para tratar os resultados das consultas obtidas no banco.

4.2.1 Erros e dificuldades encontrados no TypeScript

O TypeScript é uma linguagem que ainda pode ser considerada nova no mercado e sua primeira aparição foi em 1 de outubro de 2012. Há alguns percalços que foram percebidos ao programar em TypeScript, como a falta de documentação para pacote *@types/mysql*, que foi usado para a comunicação com o banco. Outro problema foi a própria importação para o ambiente de desenvolvimento. Foi necessário copiar a pasta diretamente do repositório oficial, para que o código funcionasse. Alguns problemas não foram solucionados lendo a documentação do pacote, que era escassa, e foram resolvidos buscando na internet ou utilizando-se de intuição para resolver o problema. O TypeScript é primariamente considerado um superset do JavaScript. O TypeScript existe apenas em tempo de desenvolvimento, e antes de ser executado ele precisa ser compilado, ou, como a comunidade adotou, transpilado. Em tempo de execução o único comportamento que existe é da linguagem JavaScript.

Pode-se misturar o JavaScript e o TypeScript durante a escrita do código, porém, o código depurado sempre será JavaScript. Existem algumas ferramentas que revertem o código para ser depurado em TypeScript, contudo, essa técnica não foi utilizada no desenvolvimento do trabalho.

Um problema bem comum encontrado foi o do escopo da palavra reservada this. Ela é usada tanto no para se referenciar ao próprio objeto no TypeScript quanto para referenciar a função no JavaScript.

Quadro 3. Os erros encontrados no desenvolvimento do servidor em TypeScript

Repository.ts(2,10): error TS2305: Module '"/home/jean/workspace/TCC/prjtypescript/src/server/Itinerario"' has no exported member 'Itinerario'. Dificuldade para lidar com importações
Pasta mysql teve que ser importada manualmente
Espécie de reflexão do JavaScript Endpoint.js:15 this.router[method](req, res); TypeError: Cannot read property 'GET' of undefined
Error: write after end node.js is a non-blocking async platform. is an Async method, therefore netSocket.end() is called before 'write' is complete.
data.push(chunk.toString());

Dificuldade para descobrir qual o formato dos dados enviados pelo servidor
url.parse(chunk.toString()) ReferenceError: url is not defined Dificuldade pra debugar o código, pois é lidado com uma linguagem dinamicamente tipada em tempo de execução. Apesar, alguns frameworks simulam o TypeScript em tempo de execução também.
Checagem de tipos não funciona em tempo de execução itinerario.duracaoViagem = row['duracaoViagem']; // Aqui duracaoViagem era do tipo Date, mas em tempo de execução ele aceitou um string.
A documentação dos módulos ainda é escassa

4.3 Aplicando os critérios de comparação nas duas linguagens

Quadro 3. A comparação no experimento (Ray, Posnett, Filkov, Devanbu, 2014) de entre PHP e TypeScript a partir dos critérios

	PHP	TypeScript
Paradigma de programação	*Scripting*	*Scripting*
Tipagem em tempo de compilação	Dinâmica	Estática
Tipagem em tempo de execução	Dinâmica	Dinâmica
Gerenciamento de memória	Gerenciada automaticamente	Gerenciada automaticamente
Propensão a erros	6%	5%
Tamanho do código	Indiferente	Indiferente

O quadro 3 mostra como ficou a comparação segundo (Ray, Posnett, Filkov, Devanbu, 2014). O paradigma de programação das duas linguagens é o paradigma de *script.* A

tipagem em tempo de execução de ambas é dinâmica, posto que TypeScript transpila para JavaScript, e este é executado. Tanto o TypeScript quanto o PHP não requerem um gerenciamento manual da memória. Então, dentre os critérios levantados, pode-se observar no quadro 3 que os critérios são iguais nas duas linguagens, exceto a tipagem em tempo de compilação. Como explica (Ray, Posnett, Filkov, Devanbu, 2014), a linguagem TypeScript foi desenvolvida com intuito de ser usado como uma linguagem de tipagem estática e forte. No entanto, na prática, foi verificado que desenvolvedores frequentemente (para 50% das variáveis, e em todos os projetos usados nas estatísticas) usam o tipo *any*, isto é, uma junção de todos os tipos, tornando assim TypeScript uma linguagem também com tipagem dinâmica e fraca. Com essa observação, pode-se dizer que TypeScript pode ser usado tanto como linguagem de tipagem dinâmica, quanto como linguagem de tipagem estática, dependendo como se programa.

Sobre o tamanho do código no desenvolvimento das duas APIs, comparando os tamanho entre os dois, no lado servidor, o PHP tinha apenas dois arquivos: itinerario.php (50 linhas) e repository (103 linhas). Enquanto isso o TypeScript contou com as seguintes classes e módulos: DbManager (10 linhas), DiaSemana (1 linha), Endpoint (27 linhas), Itinerario (12 linhas), Repository (136 linhas), Router (72 linhas) e main (3 linhas). O total de linhas do PHP é 153 no lado servidor, enquanto no TypeScript chega a 261. O script escrito em PHP têm 42% menos linhas de código.

4.3.1 Tipagem estática vs. Tipagem dinâmica: vantagens e desvantagens

(Meijer e Drayton, 2004) afirma que a tipagem estática é uma poderosa ferramenta que ajuda programadores a expressar suas premissas sobre o problema que eles estão tentando resolver e permite assim escrever um código mais conciso e correto. Lidando com premissas incertas, dinamismo e mudanças (inesperadas) estão se tornando cada vez mais importantes em um mundo com tanta diversidade. Ao invés de brigar pelas diferenças entre linguagens dinamicamente e estaticamente tipadas, é preferível encontrar uma integração pacífica com aspectos dinâmicos e estáticos em uma mesma linguagem. Tipagem estática quando possível, tipos dinâmicos quando necessário.

(Kleinschmager e Sebastian, 2012) afirma que sistemas de tipos estáticos desempenham um papel essencial em linguagens de programação contemporâneas, mas há um longo debate sobre os possíveis prós e contras do uso de tipos estáticos ou dinâmicos. Muitos autores dizem que sistemas de tipos estáticos são extremamente importantes, como (Kim B. Bruce, 2002), (Pierce, 2002) e (Cardelli, 1997). Já (Tratt, 2009) tem opinião contrária.

Para (Bracha, 2004), linguagens de programação com tipagem estática podem incluir vantagens imprescindíveis como detecção de erros em tempo de compilação e declaração de tipos como forma de documentação do código. Para (Pierce, 2002), sistemas de tipos estáticos ajudam a manter um estilo de design de código mais abstrato, onde interfaces são desenhadas e discutidas independentemente das suas eventuais implementações. Um pensamento mais abstrato sobre interfaces geralmente leva para um design melhor. (Pierce, 2002) também aponta que linguagens de programação estaticamente tipadas são mais seguras, eficientes e extensíveis, podendo-se expandir o

programa de forma organizada, dando manutenção ou adicionando novas funcionalidades.

(Tratt, 2009), em contraponto, diz que linguagens de tipos estáticos são inexpressivas e restritivas. Ele explica que existem tipos excessivamente permissivos, assim como tipos excessivamente restritivos. Também explica que sistemas de tipos podem ser complexos, dificultando assim o desenvolvimento do código. Afirma ainda que sistemas de tipo tornam o aprendizado de uma linguagem de programação mais longo, pois além da semântica e sintaxe particular da linguagem, o sistema de tipos também precisa ser aprendido, assim como erros ocasionados pelo sistema de tipos são difíceis de compreender.

(Tratt, 2009) ainda defende linguagens dinamicamente tipadas, argumentando que nelas existem qualidades indispensáveis, como a simplicidade, tanto para aprender quanto para usar. Também fazem parte dessas linguagens tipos de dados embutidos, como arrays e dicionários. Uma das características principais de linguagens dinamicamente tipadas é o gerenciamento automático da memória. (Jones e Lins, 1999) afirmam que o gerenciamento manual da memória é uma significante fonte de erros. Linguagens dinamicamente tipadas foram pioneiras em gerenciamento automático de memória, como a implementação do *garbage collector*[19]. (Tratt, 2009) ainda traz como benefício o argumento de que linguagens dinamicamente tipadas possuem muitas funcionalidades de meta-programação[20] e reflexão[21]. Linguagens de programação dinamicamente tipadas ainda podem possuir boa portabilidade. Costumam ser fáceis de instalar em múltiplos sistemas operacionais. Possuem também bibliotecas de código embutidas, para cumprir tarefas comuns, como fornecer data e hora. Linguagens dinamicamente tipadas costumam ser multiparadigma, ou ser também linguagens de *script,* o que permite ao programador ter interatividade sobre elas, como por exemplo, inspecioná-las em um terminal.

(Tratt, 2009) ainda é imparcial ao falar das desvantagens de linguagens dinamicamente tipadas. Ele começa falando da performance, que é inferior, quando comparada com a maioria das linguagens estaticamente tipadas. A depuração[22] de linguagens dinamicamente tipadas é mais trabalhosa. Outra desvantagem, segundo o autor, ainda, é que linguagens de tipos dinâmicos impossibilitam o uso do código assistido, por parte dos editores de texto.

4.4 Comparando os erros entre as duas APIs

Enquanto a maioria dos erros encontrados no PHP durante o desenvolvimento da API foi por conta de lidar com tipos desconhecidos até o momento da execução, TypeScript

---

19 Processo usado para a automação do gerenciamento de memória que libera espaço na memória. Espaço este que era usado por variáveis que não fazem mais parte do contexto da execução do programa.

20 Consulta, manipulação ou criação de um programa por outro. Frequentemente um programa pode performar alguns tipos de ações sobre ele mesmo.

21 Capacidade que um programa tem de realizar instrospecção, alterar seu comportamento ou se auto-modificar em tempo de execução.

22 Processo de encontrar e reduzir defeitos num aplicativo de software ou mesmo em hardware

teve problemas com importação de bibliotecas, fluxo assíncrono, e tipagem fraca em tempo de execução. No entanto estes erros não foram suficientes para mostrar uma diferença relevante para apontar vantagem no desenvolvimento de uma ou de outra linguagem.

5. Conclusões

(Ray, Posnett, Filkov, Devanbu, 2014) nos sugere que tipagem estática é melhor que tipagem dinâmica, como conclusão de seu estudo, no entanto, essa diferença é pequena. (Meijer e Drayton, 2004) afirma que a solução ideal seria uma linguagem que combinasse ambos aspectos, e que se usasse tipagem estática sempre que possível e tipagem dinâmica sempre que necessário, Linguagem essa aqui apontada como o TypeScript. O experimento corrobora com a ideia de (Hanenberg, 2010), que declara que sistemas dinamicamente tipados tem benefício em pequenos projetos, enquanto este benefício é reduzido (ou mesmo inexistente) em projetos maiores.

Através dos resultados trazidos pelo artigo e pelo experimento de desenvolvimento das duas APIs, não se notou nenhum benefício ou vantagem contrastante entre uma e outra linguagem. Existem prós e contras para o uso de cada uma, como visto na seção 4.3.

Referências

Ray, B., Posnett, D., Filkov, V., & Devanbu, P. (2014, November). A large scale study of programming languages and code quality in github. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 155-165). ACM.

O'Reilly, Tim, What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. Communications & Strategies, No. 1, p. 17, First Quarter 2007.

Papazoglou, M. (2008). Web services: principles and technology. Pearson Education.

E. Meijer and P. Drayton. Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages. In OOPSLA'04 Workshop on Revival of Dynamic Languages, 2004.

Ralph, P. and Wand, Y. (2009). A proposal for a formal definition of the design concept. In Lyytinen, K., Loucopoulos, P., Mylopoulos, J., and Robinson, W., editors, Design Requirements Workshop (LNBIP 14), pp. 103–136. Springer-Verlag, p. 109

Pierce, Benjamin C. Types and programming languages. MIT press, 2002.

TRATT, Laurence. Dynamically typed languages. Advances in Computers, v. 77, p. 149-184, 2009.

SOUZA, Carlos; FIGUEIREDO, Eduardo; VALENTE, Marco Tulio Oliveira. Tipar ou não Tipar? Compreendendo Quais Fatores Influenciam a Escolha por um Sistema de

Tipos. In: First Brazilian Workshop on Visualization, Evolution and Software Maintenance (VEM). 2013.

Da Penha, Dulcinéia O., et al. "Performance evaluation of programming paradigms and languages using multithreading on digital image processing." Proceedings of the 4th WSEAS International Conference on Applied Mathematics and Computer Science. World Scientific and Engineering Academy and Society (WSEAS), 2005.

ZEND, 2016. History of PHP <http://php.net/manual/en/history.php.php>. Acesso em: 20/06/2017

Kleinschmager, Sebastian, et al. "Do static type systems improve the maintainability of software systems? An empirical study." Program Comprehension (ICPC), 2012 IEEE 20th International Conference on. IEEE, 2012.

Kim B. Bruce. Foundations of object-oriented languages: types and semantics. MIT Press, Cambridge, MA, USA, 2002.

Benjamin C. Pierce. Types and programming languages. MIT Press, Cambridge, MA, USA, 2002.

Luca Cardelli. Type systems. In Allen B. Tucker, editor, The Computer Science and Engineering Handbook, chapter 103, pages 2208–2236. CRC Press, 1997.

Laurence Tratt. Dynamically typed languages. Advances in Computers, 77:149–184, July 2009.

Hanenberg, Stefan. "An experiment about static and dynamic type systems: Doubts about the positive impact of static type systems on development time." ACM Sigplan Notices. Vol. 45. No. 10. ACM, 2010.

Scriptol, 2016. PHP, a language for building Web pages on the server <http://www.scriptol.com/programming/php.php>. Acesso em: 20/06/2017

Jungthon, G., & Goulart, C. M. (2009). Paradigmas de Programação. Monografia (Monografia) Faculdade de Informática de Taquara, Rio Grande do Sul, 57.

Britton C. 2017. Choosing a Programming Language <https://msdn.microsoft.com/en-us/library/cc168615.aspx>. Acesso em: 20/06/2017

Knuth, D. E. (1997). Seminumerical Algorithms. Third edn. Volume 2 of The Art of Computer Programming.

WSEAS International Conference on Applied Mathematics and Computer Science. World Scientific and Engineering Academy and Society (WSEAS), 2005.

Bracha, G. Pluggable type systems. In OOPSLA'04 Workshop on Re-vival of Dynamic Languages, October 2004.

Jones, R., Lins, R. Garbage Collection: Algorithms for Automatic Dynamic Memory Management. Wiley, 1999

Reghunadh J., Jain N., 2017, Selecting the optimal programming language <https://www.ibm.com/developerworks/library/wa-optimal/>. Acesso em: 20/06/2017