UNIVERSIDADE FEDERAL DE ALAGOAS - UFAL
CAMPUS A. C. SIMÕES
CIÊNCIA DA COMPUTAÇÃO

LUCIANO DE MELO SILVA

JUWEB: UM ARCABOUÇO PARA APLICAÇÕES WEB EM JULIA

MACEIÓ–AL
2017

Luciano de Melo Silva

JUWEB: UM ARCABOUÇO PARA APLICAÇÕES WEB EM JULIA

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal de Alagoas - UFAL, Campus A. C. Simões.

Orientador: Prof. Dr. André Lage Freitas

Maceió–AL
2017

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária Responsável: Lívia Silva dos Santos - CRB 1670

S586j Silva, Luciano de Melo.
Juweb : um arcabouço para aplicações web em Julia / Luciano de Melo Silva. –2017.
47 f.:il.

Orientador: André Lage Freitas.
Monografia (Trabalho de Conclusão de Curso em Ciência as Computação) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2017.

Bibliografia: f. 46-47

1. Linguagem de programação - JuWeb,. 2. Computação científica. 3. Arquitetura do arcabouço - JuWeb. I. Título.

CDU: 004.43

Luciano de Melo Silva

JUWEB: UM ARCABOUÇO PARA APLICAÇÕES WEB EM JULIA

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal de Alagoas - UFAL, Campus A. C. Simões.

Data de Aprovação: 01/02/2018

Banca Examinadora

Prof. Dr. André Lage Freitas
Universidade Federal de Alagoas
Campus A. C. Simões
Orientador

Prof. Dr. Fabio José Coutinho da Silva
Universidade Federal de Alagoas
Campus A. C. Simões
Examinador

Prof. Dr. Leandro Melo de Sales
Universidade Federal de Alagoas
Campus A. C. Simões
Examinador

A Deus, por ser essencial em minha vida, pois sem Ele eu não teria forças para essa longa jornada.

AGRADECIMENTOS

E é a Deus que dirijo minha maior gratidão. Por tudo que sou, todas as vezes que Ele me acolheu em seus braços e me mostrou o caminho correto a percorrer.

Aos meus pais, Angela e Leonildo, que me proporcionaram uma infância com responsabilidade, no aspecto moral e também acadêmico. Sou grato por fundamentarem meu caráter de forma correta.

Ao professor e orientador, André Lage Freitas, pelo desprendimento ao escolher-me para ser seu orientando. Seu apoio em nossas reuniões, e pelo tempo que dispôs a me ajudar com sua competência, paciência e conhecimento.

Aos amigos de curso, de perto e de longe, pelos momentos que passamos durante nossas graduações. Obrigado, vocês foram um apoio nas horas mais difíceis, trazendo força e alegria.

A Jennyfer de Oliveira Lima, que me ajudou minuciosamente em cada parte deste trabalho, com seu conhecimento teórico e sua criatividade. Seu companheirismo e contribuição por todos esses anos, ajudou-me a ser um ser homem muito melhor. Não esquecerei por tudo que você fez por mim.

Ao Valmir e Maria José, por todo apoio e acolhimento que me deram, incentivando minha caminhada, fortalecendo-me cada dia mais.

Obrigado por serem a minha referência e por estarem presentes na minha vida de uma forma indispensável, mesmo separados por tantos quilômetros.

Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito. Não sou o que deveria ser, mas Graças a Deus, não sou o que era antes.

Marthin Luther King

RESUMO

Nos últimos anos houve um avanço significativo das tecnologias voltadas para aplicações científicas na Web, oferecendo muito mais vantagens para transferência de conhecimento científico. Por consequência, foi inevitável o crescimento do número de ferramentas de desenvolvimento de aplicações que disponibilizem recursos Web. Julia é uma linguagem de programação dinâmica nova, de alto nível, que foi projetada com foco em desempenho para computação científica e numérica, graças à compilação *just-in-time* (JIT) de seu compilador e suas características. Entretanto, sua abordagem para o desenvolvimento Web é muito limitada, com pacotes defasados e incompatíveis com versões atuais da própria linguagem. Diante disso, é proposto o arcabouço JuWeb, que disponibiliza ferramentas de apoio ao desenvolvimento de aplicações Web para Julia. O estudo de caso foi a implementação da aplicação HugePolSARWeb, que permite o processamento e visualização de imagens de satélite através de uma interface Web. Através de avaliação do JuWeb, foi possível demonstrar que a experiência de desenvolvimento Web com JuWeb é viável e facilitada por este arcabouço. Além disso, o tempo de execução de requisições da HugePolSARWeb manteve valores aceitáveis, o que possibilita a implementação de serviços Web em produção.

Palavras-chave: Julia, arcabouço, *framework*, desenvolvimento Web, serviços Web.

ABSTRACT

In recent years, there has been a significant advance in technologies focused on scientific applications on the Web, offering many more advantages for the transfer of scientific knowledge. Consequently, the number of application development tools for Web applications has significantly increased. Julia is a new high-level dynamic programming language whose designed focuses on efficient scientific computing, thanks to Julia just-in-time (JIT) compiler and its LLVM architecture. Nonetheless, Julia Web development support is very limited, with outdated and incompatible packets to current versions of Julia Therefore, we propose the JuWeb framework for developing Web applications by using Julia programming language. JuWeb differs from other related work owing to its complete set of functinalities its and layered design based on the Model-View-Controller (MVC) approach. For the sake of usability, we developed the HugePolSARWeb Web application by using JuWeb for processing radar data in the Web. Moreover, JuWeb proved to be efficient by executing client requests executed accordingly.

Keywords: *Julia, reuse, framework, Web applications, Web service.*

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
ASA	Árvore de Sintaxe Abstrata
CGI	*Common Gateway Interface*
CORBA	*Common Object Request Broker Architecture*
CPU	*Central Processing Unit*
CSS	*Cascading Style Sheets*
DAO	*Data Access Object*
DBC	Desenvolvimento Baseado em Componentes
DLTK	*Dynamic Languages Toolkit*
EPUB	*Electronic Publication*
GUI	*Graphical User Interface*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
I/O	*Input/output*
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
JDK	*Java Development Kit*
JIT	*Just-in-time compilation*
JSON	*JavaScript Object Notation*
JVM	*Java virtual machine*
LLVM	*Compiler Low Level Virtual Machine*
MVC	*Model–view–controller*
PDF	*Portable Document Format*
PKG	*Package*
PNG	*Portable Network Graphics*
RAM	*Random Access Memory*
REPL	*Read-Eval-Print-Loop*
REST	*Representational State Transfer*
ROI	*Region of Interest*

SUMÁRIO

1 INTRODUÇÃO

A sociedade moderna está em constante evolução, em vista disso enfrenta cada vez mais grandes desafios que envolvem o processamento de enormes quantidades de dados e a realização de cálculos complexos (EU, 2013). É sabido que o conhecimento científico ajuda a produzir novas soluções tecnológicas, e as aplicações científicas têm em vista a resolução de diversos problemas de ordem prática e, bem como teórica, sendo essenciais para o desenvolvimento intelectual e produtivo da humanidade. Essas aplicações podem ser implementadas em diversas áreas multidisciplinares, incluindo: Biosciência, Dados Geográficos, Modelagem da Indústria Petrolífera e de Gás, Automação de Design Eletrônico, Modelagem Climática, Mídia e Entretenimento (TECHOPEDIA). Como exemplos mais específicos, podemos citar as aplicações de processamentos de algoritmos complexos em imagens robustas de satélites, a descoberta e sequenciação de estruturas genéticas do DNA para associação e rastreio de padrões de doenças genéticas, nos ramos da computação e biologia, respectivamente.

Segundo (DONGARRA et al., 2005), embora a medida final da eficácia da execução de um determinado sistema computacional seja seu tempo de resposta ou tempo de execução para uma determinada aplicação, outras variáveis correlacionam-se com essa métrica fundamental. O desempenho na execução de aplicações científicas é um fator qualitativo importante, e por isso deve ser levado em consideração durante os processos de produção dessas soluções. A computação de alto desempenho é o conjunto de várias tecnologias, dentre elas a arquitetura de computadores, programas, algoritmos e softwares e aplicativos em um único sistema para resolver problemas avançados de forma rápida e eficaz (EZELL; ATKINSON, 2016, p. 2). Para que essa necessidade computacional seja suprida, tradicionalmente são utilizadas linguagens de programação que apresentam melhores precisões numéricas e maiores velocidades de execução, tais como Fortran e C. Além dessas, linguagens com sintaxes de mais alto nível e/ou numéricas são frequentemente utilizadas, tais como Python, MATLAB, R e Julia (BEZANSON et al., 2017).

A linguagem de programação Julia foi criada recentemente, mas surpreende ao se apresentar competitiva em um mercado habitado por linguagens já consolidadas. O ponto forte dessa linguagem é seu projeto e seu compilador *Low Level Virtual Machine* (LLVM), que a partir de suas modernas técnicas de compilação permitem eliminar principalmente a ponderação entre desempenho e produtividade, equiparando-se às linguagens C e Fortran no quesito desempenho e igualmente fácil de aprender como MATLAB e R. O crescimento da linguagem Julia deu-se

inicialmente no *Massachussetts Institute of Technology* (MIT) e, em seguida, pelo investimento da produção de pacotes de serviços voltados às várias áreas multidisciplinares, através de implementações feitas em cooperação por desenvolvedores ao redor do mundo, participantes de comunidades virtuais para produção de módulos e pacotes integrantes da linguagem. Ademais, fornece suporte nativo à programação paralela, é livre, multi-paradigma e combina recursos da programação imperativa, funcional, orientada a objetos.

Com o avanço das tecnologias e a importância da acessibilidade à ciência de forma dinâmica, a relação entre as aplicações atuantes na *World Wide Web* e a ciência é um ponto fundamental para se eliminar barreiras que excluam pessoas da sociedade de participar e contribuir para o atual ramo científico e tecnológico. Assim, a Web tornou-se um meio indispensável para disponibilizar conhecimento.

Atualmente, o suporte de programação em Julia para Web consiste em poucas ferramentas que ainda estão em fase desenvolvimento, incompletas ou seus desenvolvimentos não foram continuados. Dentre os pacotes existentes da linguagem, o HttpServer.jl é responsável pelo compartilhamento de rotas e serviços simples, o Mustache.jl pelo processamento e renderização de objetos com *tags*. Já o pacote Requests.jl trata requisições simples e o HTTP.jl é o pacite com a principal dependência integradora do protocolo HTTP para Julia. Contudo, esses pacotes apresentam limitações, por exemplo, o pacote HttpServer.jl não é compatível com a versão mais atual de Julia (versão 0.6 para a data deste trabalho) e o Requests.jl apresenta documentação escassa e muito superficial. O pacoite Mustache.jl está com seu projeto inativo além de não manter um controle de estados. Por útlimo, o HTTP.jl não apresenta exemplos de implementação. Além dessas limitações, nenhum desses pacotes apresentam uma infraestrutura completa e estruturada em camadas de arcabouço para desenvolvimento Web em Julia.

Neste cenário, o trabalho ora apresentado busca proporcionar subsídios para o desenvolvimento de aplicações científicas na linguagem Julia. Para isso, foi projetado e produzido um arcabouço Web que facilita a produção de aplicações Web em Julia, disponibilizando-as a partir do paradigma *Model-View-Controller* (MVC), na arquitetura cliente e servidor. Em suma, as contribuições desse trabalho são destacadas nos seguintes itens:

- A principal contribuição dá-se ao estudo e a elaboração de um arcabouço web baseado no paradigma MVC (apresentado na Seção 3.3) denominado JuWeb, que tem por finalidade o incentivo da produtividade e do desenvolvimento de aplicações científicas para a Web.
- A segunda contribuição é a implementação de uma aplicação científica com interface Web

para o processamento de dados de satélite. Trata-se do estudo de caso utilizado para validar a aplicabilidade do JuWeb. Ou seja, se essa aplicação tem viabilidade funcional, havendo a possibilidade de sua utilização efetiva em produção.

As próximas seções deste trabalho são organizadas da seguinte maneira: O capítulo 2 apresenta uma abordagem geral da linguagem Julia, suas principais características e suporte para desenvolvimento Web. O capítulo 3 apresenta uma explicação sobre Reúso de *software*, um estudo realizado sobre conceitos, classificações, benefícios e desafios de *Frameworks* e do padrão *Model-View-Controller* (MVC). O capítulo 4 apresenta todas as informações sobre o arcabouço JuWeb e detalhes de implementação. O capítulo 5 apresenta a aplicação HugePolSARWeb, estudo de caso deste trabalho. O capítulo 6 mostra as avaliações realizadas no JuWeb, metodologias, cenários aplicados e resultados. Por fim, no capítulo 7 são apresentadas as considerações finais e conclusões acerca do trabalho.

2 JULIA

Neste capítulo, são apresentadas as características da linguagem Julia, sua expansão global e elementos que a compõem. Igualmente, são apresentadas tecnologias e ferramentas da linguagem usadas no desenvolvimento deste trabalho.

2.1 UMA LINGUAGEM LIVRE E COM COMUNIDADE ATIVA

Julia (BEZANSON et al., 2017) foi desenvolvida pelo Instituto de Tecnologia de Massachusetts (MIT), Cambridge, Massachusetts, nos Estados Unidos. Sua produção iniciou em agosto de 2009, até que em fevereiro de 2012 Julia foi projetada para atuar em diversas áreas científicas, com foco na computação científica. Assim, apresenta uma sintaxe numérica, dinâmica e é executada através de um ambiente interativo e com interface simples, que aguarda uma expressão como entrada de usuário, as processa e exibe os resultados – *Read - Evaluate - Print - Loop* (REPL).

O compilador de Julia utiliza a abordagem *Low-Level Virtual Machine* (LLVM) e possui arquitetura moderna baseada nos mecanismos de compilação *just-in-time* (JIT). Além disso, o compilador disponibiliza suporte nativo para a programação distribuída através de macros que abstraem detalhes operacionais do ambiente de execução. Por último, os compiladores de Julia estão disponíveis para os sistemas operacionais mais populares, como Windows, MacOS e Linux. Os binários dos compiladores são disponibilizados também para as arquiteturas x86 e ARM, de 32 e 64 bits.

Stefan Karpinski, Jeff Bezanson, Viral Shah e o professor de Matemática Aplicada Alan Edelman do MIT foram os principais projetistas e desenvolvedores da linguagem Julia. A equipe se uniu para produzir uma linguagem de programação para suprir a necessidade de várias outras linguagens simultaneamente. Por exemplo, a linguagem R, apesar de ser voltada para cálculos estatísticos e gráficos, sua execução não é ágil tanto quanto Fortran, que é otimizada para análises numéricas. Então, (KARPINSKI, 2014) decidiu se juntar com outros cientistas da computação e resolveram criar uma nova linguagem que fosse adequada para a computação científica, combinando uma sintaxe numérica com alta eficiência.

A equipe de desenvolvedores de Julia propôs uma linguagem fundamentalmente de código aberto, licenciada pelo MIT, para que usuários deste tipo de ferramentas tenham uma alternativa a *softwares* proprietários, como por exemplo o MATLAB, ambiente e linguagem de computação numérica multi-paradigma, ou a linguagem R, que não apresenta desempenho de

execução no nível de linguagens compiladas.

Softwares que apresentam licenças livres trazem mais autonomia aos seus usuários finais, acrescendo sua acessibilidade de forma dinâmica a todos que a utilizarem. A seguir, está a descrição da licença de Julia:

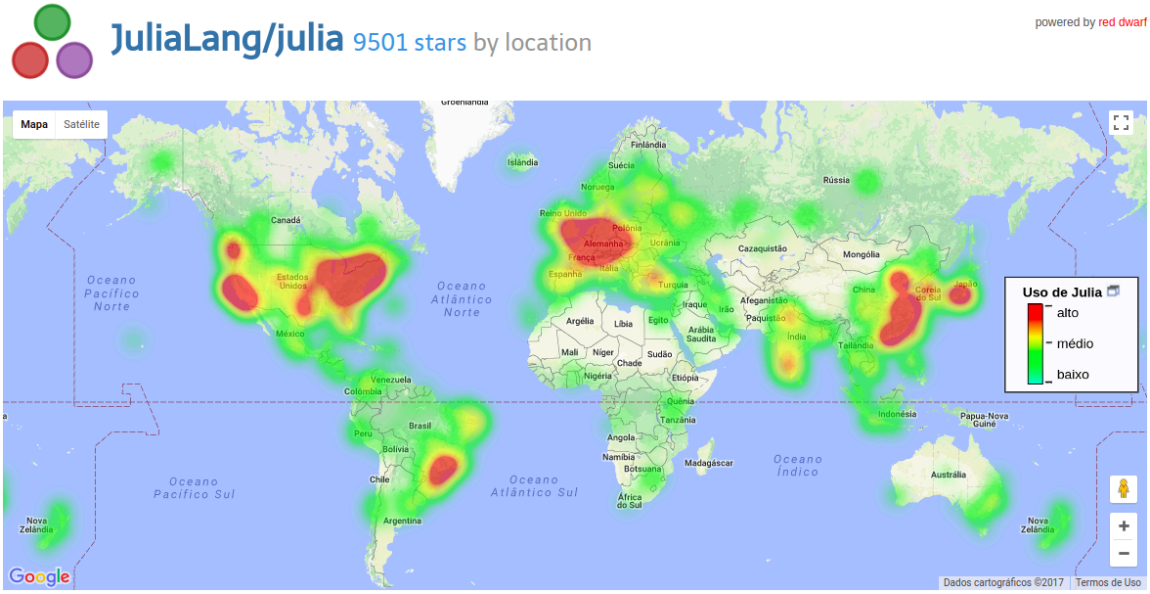
> É concedida permissão, gratuitamente, a qualquer pessoa que obtenha uma cópia deste *software* e seus arquivos de documentação associados (o "*Software*"), para lidar com o *Software* sem restrições, incluindo, sem limitações, os direitos de uso, cópia, modificação, mesclagem, publicação, distribuição, sublicenças e/ou venda de cópias deste *software* [. . .][1] (MIT, 2009) (Tradução do autor).

Por ser de código-fonte aberto, Julia se tornou uma linguagem com um grande número de colaboradores, ou seja, usuários que participam expressivamente na produção de pacotes para seu código nativo. Atualmente, existem várias comunidades de desenvolvedores de códigos em Julia, que por sua vez possibilitam maiores interações em abordagens mais específicas de produção, em diversas áreas do conhecimento. Esse cenário também é amplamente expandido com conferências e eventos que ocorrem ao redor do mundo, e.g. JuliaCon, proporcionando uma experiência de discussões acerca de pontos importantes, concernentes à ampliação da linguagem. São apresentadas na Figura 1 as áreas geográficas de acordo com o uso dos repositórios do GitHub oficiais da linguagem Julia, escaláveis em três níveis de utilização: alto para um conglomerado aproximado de 9914 repositórios, médio para um valor acima de 2183 repositórios e baixo para um valor abaixo de 2183.

---

[1] Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software [. . .]

Figura 1 – Adoção da Linguagem Julia no mundo, Julho de 2017.



Fonte: Disponível em: Red Dwarf. Acesso em: 12 nov. 2017.

Em suma, a comunidade global de Julia está em constante crescimento, cenário bastante notável quando avaliamos as informações de *commits* e *pull requests* apresentadas em sua página oficial do GitHub (<https://github.com/JuliaLang/julia>). Atualmente, existe uma alta demanda por programadores de Julia, em todas as áreas de conhecimento. Dentre as empresas contratantes, estão Google, Apple, Facebook, Amazon, Oracle, bem como utilizam-na em seus projetos.

Julia possui um gerenciador de pacotes embutido, baseado em Git, para organização e controle das versões dos códigos. Seu sistema gerenciador de pacotes possibilita a construção, instalação e gerência facilitada de bibliotecas. Os pacotes oficiais da linguagem estão registrados no arquivo METADATA.jl,localizado no repositório principal de Julia (<http://github.com/JuliaLang>). As funções referentes à gerência de pacotes são definidas no módulo chamado **Pkg**, dentre as principais estão:

- Pkg.update(): efetua atualizações dos pacotes instalados na biblioteca;
- Pkg.add("NomeDoPacote"): faz a instalação do pacote de nome NomeDoPacote, registrado na METADATA, à biblioteca;
- Pkg.build("NomeDoPacote"): faz a construção do pacote NomeDoPacote, função útil caso seja necessário compilar o pacote para adequação de dependências;
- Pkg.clone("Git_url"): instalação de um pacote a partir de um endereço Git;

2.2 ALTO DESEMPENHO

O alto desempenho é essencialmente importante para a computação científica devido à alta necessidade de processamentos computacionais não-triviais que devem ser executados. Pode-se aumentar o desempenho de aplicações científicas através de otimização em nível de compilação e através da execução paralela de algoritmos. Além disso, tornou-se necessário maior agregação de poder computacional às soluções de grandes problemas científicos atuais, tais como: aplicações que precisem de processamentos algorítmicos complexos em imagens robustas, para que essas sejam processadas, geradas e apresentadas em um mapa, em tempo real, em escala definida a partir de valores de parâmetros de um serviço web. Ou até mesmo para cálculos matemáticos de valores grandes e complexos.

O projeto de Julia foi pensado de modo a eliminar a ponderação entre a facilidade de prototipagem utilizando uma linguagem dinâmica e o desempenho de uma estática. Ou seja, permitindo maior flexibilidade em executar códigos arbitrários ou transformá-los em tempo de execução, de forma clara, ágil e sem a obrigatoriedade de compilação. O projeto de Julia também atende às necessidades de usuários que precisam das características de uma linguagem dinâmica e os que precisam da velocidade de execução encontrada em C por exemplo.

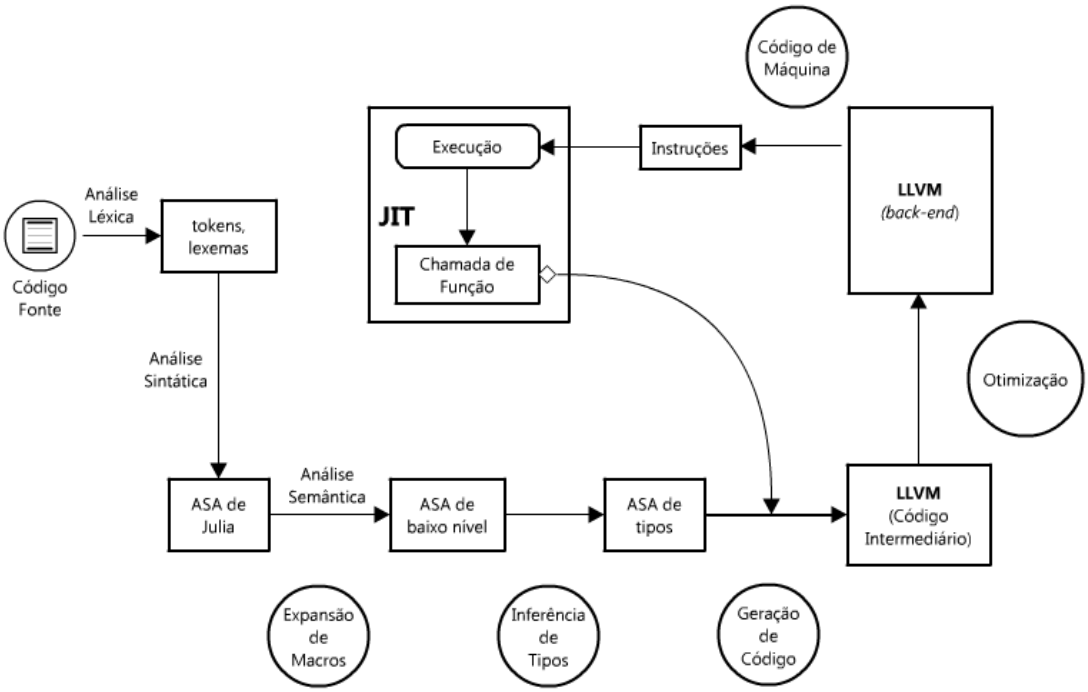
Para alcançar alto desempenho na execução do programa, o compilador de Julia utiliza recursos da compilação JIT da infraestrutura *Low-Level Virtual Machine*. A LLVM é um projeto desenvolvido por uma equipe da Universidade de Illinois em Urbana-Champaign, cujo objetivo é prover um conjunto modular de componentes para o desenvolvimento de compiladores e outras ferramentas de compilação como analisadores estáticos de código. A JIT é um tipo de compilação em tempo de execução, que combina a performance de um código compilado à flexibilidade de um interpretado, efetuando transformações em tempo real, atingindo ganhos de desempenho e permitindo que Julia alcance valores aproximados a C/C++.

A Figura 2 ilustra os processos de compilação do LLVM, a partir da execução de um código em Julia. Na primeira etapa é feita a análise do código fonte na ASA de Julia, onde as construções de metaprogramação da linguagem serão expandidas (macros) e a avaliação e fixação de tipos (inferência de tipos), para fins de otimização e geração de código, mais especificamente funções, uma vez que é a unidade de trabalho dos compiladores JIT. Por conseguinte, após os processamentos das ASAs de Julia é feita a geração de um código intermediário pelo compilador LLVM, que é previamente tratado dependendo da plataforma, visto que há dependência entre ele e a constituição do código intermediário, dentre alguns fatores estão: os tamanhos de ponteiros,

o alinhamento de memória e a própria nomenclatura das chamadas, que são distintas entre outras plataformas.

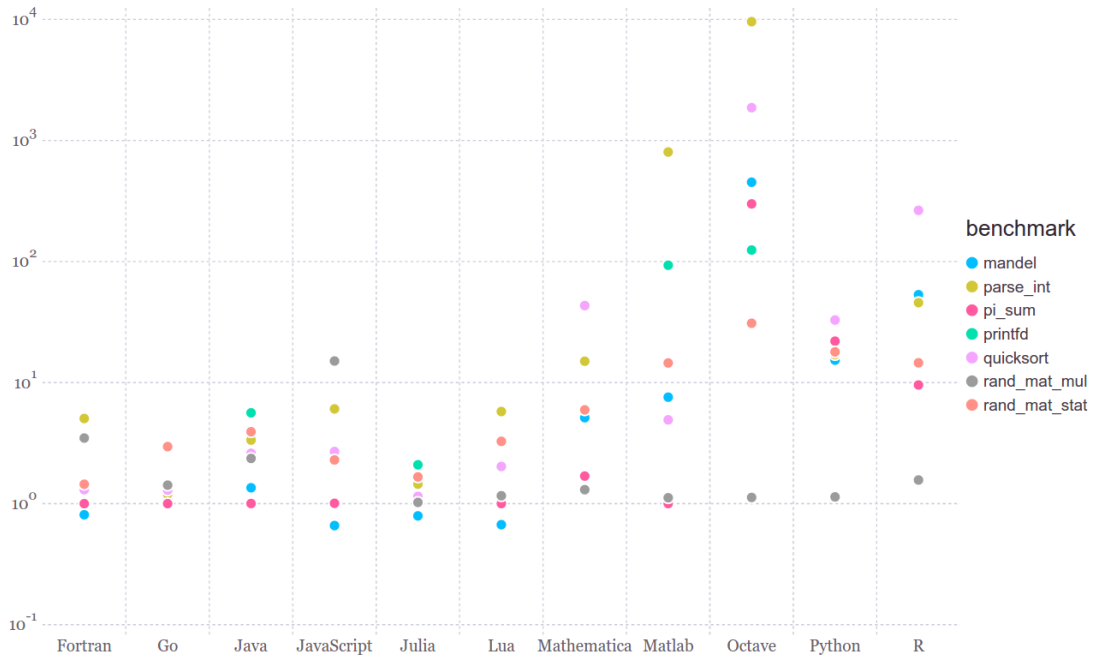
Por fim, o código intermediário poderá ser otimizado e compilado para o código de máquina final a partir das camadas mais internas e complexas do compilador LLVM (*back-end*), ver Figura 2, etapas de otimização, geração de código de máquina, até a execução das instruções e chamada de função. Entretanto, esse pode não ser o estágio final de compilação, durante a execução ainda há a possibilidade do desencadeamento de uma nova compilação, semelhante a um processo recursivo. Isso acontece pelo fato do LLVM executar sua compilação JIT baseada em gerações de funções, traduzindo cada uma sob demanda. Por exemplo, ao executar uma função ainda indefinida no ponto em que a chamada foi gerada, a função chamada será compilada em tempo de execução, denominada de compilação *on-the-fly*.

Figura 2 – Compilador LLVM de Julia, principais componentes e respectivos processos de compilação e execução de um código.



A Figura 3 demonstra o quanto Julia é uma linguagem bem rápida comparada às linguagens de programação Fortran, Go, Java, JavaScript, Lua, Mathematica, Matlab, Octave, Python, R. Foi utilizada a técnica de *benchmark* para que seja permitido realizar comparações entre os tempos de execução entre essas linguagens. Assim, os *benchmarks* são implementações de algoritmos matemáticos a fim de identificar seus valores, vantagens e desvantagens diante de várias aplicações científicas.

Figura 3 – *Benchmark* referente ao tempo relativo à linguagem C (quanto menor melhor, desempenho de C = 1.0)



Fonte: Disponível em: <http://julialang.org>. Acesso em: 18 set. 2017.

É notório, a partir dos valores apresentados na Figura 3, a proximidade dos tempos encontrados Julia para com C, sendo um resultado bastante superior, quando comparada a Python ou R. Portanto, Julia mostrou-se com desempenho satisfatório em suas execuções, comprovando sua capacidade de prover uma rápida resposta. Além disso, "Tempo de execução é um parâmetro chave no desenvolvimento e projeto de *software*, pois podem existir limites mínimos ou máximos de tempo que a aplicação deve executar para atender seu propósito" (ZAPALOWSKI, 2011, p. 28).

2.3 DESENVOLVIMENTO WEB COM JULIA

A *World Wide Web*, ou simplesmente Web, é o espaço de conteúdos mais popular para retorno de informações, interesses e compartilhamento de recursos. No início da Web, os navegadores retornavam apenas páginas puramente estáticas. Em seguida, incorporou-se conteúdo dinâmico na Web a partir de resultados de programas através de *scripts Common Gateway Interface* (CGI[2]) em um servidor. Essa abordagem evoluiu para o uso de linguagens

[2] Tecnologia usada para geração de páginas dinâmicas, a partir da passagem de parâmetros para uma aplicação simples em servidor web.

voltadas especificamente para a Web, tais como: Java, Ruby e PHP. A popularidade da Web fez com que esta se tornasse um meio indispensável para comunicação.

Julia apresenta soquetes TCP/IP implementados em seu core, suporta CURL, SMTP e *WebSockets*. Além disso, possibilita trabalhar no protocolo HTTP a partir do uso de um conjunto de pacotes, como: HTTP.jl, HttpServer.jl, JSON.jl e Mustache.jl (SHERRINGTON, 2015, p. 28). Por exemplo, uma aplicação web precisa de um servidor que disponibilize os seus recursos. Geralmente, os sistemas Web têm uma característica em comum: bloqueiam um processamento enquanto utilizam um I/O no servidor. Esse bloqueio é chamado de modelo bloqueante (*Blocking-Thread*). Assim, por necessidade, os usuários podem efetuar requisições a esses servidores. Entretanto, os pacotes HTTP.jl e HttpServer.jl são os principais fornecedores dos serviços HTTP, compostos por objetos que possibilitam o trabalho com o protocolo TCP a partir de um servidor simples e não bloqueante, para compartilhamento de recursos a clientes.

Para complementar os serviços dos pacotes HTTP.jl e HttpServer.jl, deve-se utilizar pacotes auxiliares que disponibilizam os tipos necessários para as trocas de mensagens relativas as solicitações e respostas. Dentre esses pacotes, o Mustache.jl é o responsável pela geração de páginas HTML, a partir do processamento e renderização dos objetos com suas tags internas, expandindo-as e aplicando seus valores. O HttpCommon.jl e JSON.jl fazem parte dos pacotes que aderem tipos a serem compartilhados nas trocas de mensagens internas.

3 *FRAMEWORKS*

Neste capítulo será abordado o estudo realizado sobre as características, conceitos, classificações, benefícios e desafios da utilização de arcabouços com o intuito de manter um viés motivacional da aplicabilidade, este capítulo iniciará com uma explicação breve sobre reúso de *software*.

3.1 REÚSO

A concepção de reúso foi datada em 1968, a partir de um artigo de (MCILROY, 1968) apresentado na conferência de Engenharia de Software NATO SOFTWARE ENGINEERING CONFERENCE, na Alemanha. É uma das técnicas mais importantes da Engenharia de *Software*, sendo essa uma chave primária para grandes melhorias na produtividade de um *software*. (MORISIO et al., 2002, p. 3) define reúso como:

> A reutilização de *software* é a prática sistemática de desenvolvimento de *software* a partir de uma junção de blocos de construção, de modo que as semelhanças em requisitos e/ou arquitetura entre aplicativos possam ser exploradas e que sejam obtidos benefícios substanciais em produtividade, qualidade e desempenho do negócio.[3] (Tradução do autor)

Conforme um *software* é progressivamente desenvolvido, sua complexidade produtiva e de codificação poderá aumentar na mesma proporção, negativando sua expressividade e qualidade. Fatores estruturais e organizacionais devem ser levados em consideração. Uma boa prática para a qualidade de *software* é a escrita de códigos em componentes simples, pequenos, e juntamente com o princípio da responsabilidade única. Para (FOWLER, 1999, p. 9), "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.[4]" Dito isto, atualização dos componentes também é um fator importante, ao longo do tempo é necessária para adequação de requisitos funcionais e não funcionais do sistema.

Por exemplo, a linguagem Java não apenas incentiva a reutilização de *software*, como é tratada como uma exigência. Para que um programa seja escrito em Java, não importa o grau de dificuldade de implementação, ele deve ser implementado a partir das classes e métodos da API Java. De forma semelhante, Python e Julia apresentam suporte a classes e possuem

---

[3] Software reuse is the systematic practice of developing software from a stock of building blocks, so that similarities in requirements and/or architecture between applications can be exploited to achieve substantial benefits in productivity, quality and business performance. (MORISIO et al., 2002, p. 3)

[4] Refatorar é o processo de alteração de um sistema de software de tal forma que não altere o comportamento externo do código e ainda melhore sua estrutura interna. (FOWLER, 1999, p. 9, tradução do autor).

mecanismos de organização por pacotes e também reúso de módulos. Em Julia, podemos citar sua abordagem conveniente para criar e gerenciar pacotes (apresentado na Seção 2.1), de forma que o encapsulamento de funcionalidades em componentes concisos e menores é uma prática de reúso amparada por (MORISIO et al., 2002, p. 3).

Portanto, a praticidade na implementação, além da clareza e facilidade na escrita de código são motivações do desenvolvimento de *software* com a utilização das técnicas de reúso. Na busca de promover acesso controlado e coeso dos dados compartilhados entre componentes de um *software*, houve a necessidade da utilização de um conceito abstrato, o *Framework*.

3.2 DEFINIÇÃO E CARACTERÍSTICAS

Frameworks podem fornecer um alto grau de reutilização (FIRESMITH, 1993, p. 6), sua definição mostra que há uma variedade de interpretações advindas de autores da área, apesar disso, elas convergem para um conceito único e aplicável. Sobredito, este arcabouço é definido de diferentes formas; para (BRAUDE; BERNSTEIN, 2016, p. 358), "é uma coleção de artefatos de *software* utilizável por várias aplicações diferentes, artefatos que são tipicamente implementados como classes, juntamente com o *software* necessário para utilizá-los". Já para (APEL et al., 2013, p. 79), "é um conjunto incompleto de classes em colaboração, que pode ser estendido e adaptado para um caso de uso específico". Entretanto, (GAMMA et al., 1995, p. 40) dizem que "é um conjunto de classes cooperantes que compõem um *design* reutilizável para uma classe específica de *software*".

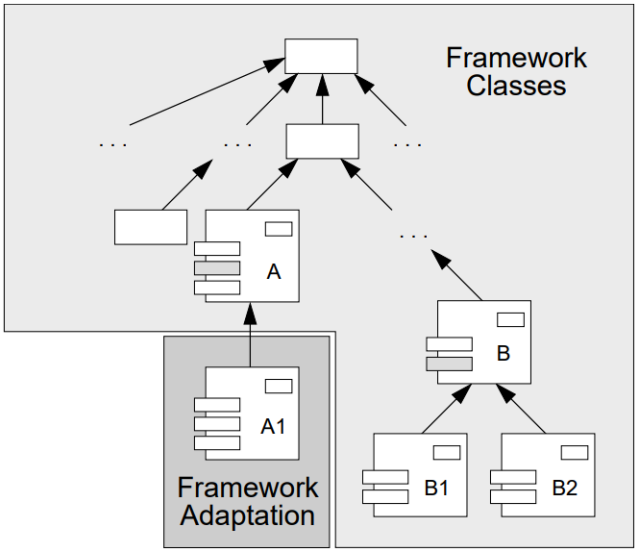
Sucintamente, é uma estrutura genérica e expansível para se criar uma aplicação com domínio específico, servindo como um modelo para o desenvolvimento de aplicações. Exempli gratia, GUIs são geralmente projetadas e estruturadas em arcabouços, para que não seja necessário reescrever códigos para implementá-las em novas aplicações. Portanto, o desenvolvimento e utilização dessas ferramentas têm proporcionado maior reutilização e rapidez na disponibilidade de novas aplicações no mercado. Atualmente, a criação de um arcabouço para auxílio nos processos de produção de *softwares* está cada vez mais comum, e sua utilização é proeminente, demandando capacitação constante dos desenvolvedores. De acordo com (SONMEZ, 2014, p. 197), todos os dias novas tecnologias são introduzidas e a grande diferença do desenvolvimento de *software* para as muitas outras áreas de produção, é a mudança constante.

As implementações e complexidades são encapsuladas, de forma que são abstraídas e agrupadas em uma série de sub-unidades distintas, apresentadas através de interfaces bem definidas ao desenvolvedor, i.e., que tenham alta coesão. Quando são construídas aplicações,

a modularidade de seus componentes proporciona grande benefício, pois quanto maior for o isolamento de unidades funcionais mais fácil será para um projetista de aplicações entender uma parte específica (KRASNER; POPE, 1988, p. 26). Em suma, a modularidade aumenta qualitativamente o *software*, uma vez que, ao ocorrer alterações durante o processo de implementação, estas serão encontradas mais facilmente, por conseguinte, incrementando a manutenibilidade de *software*.

Um arcabouço deve permitir a integração de serviços adicionais sem a necessidade de modificações na estrutura de sua arquitetura principal. Para isso, contém pontos explícitos de flexibilização, que permitem a sua adaptação e extensão, denominados *hot-spots*, assim como é mostrado na Figura 4. Em particular, desacoplam as partes dependentes da aplicação e as partes independentes da plataforma, aumentando assim a extensibilidade, a flexibilidade e a portabilidade do *software* (SCHMIDT et al., 2004).

Figura 4 – Hierarquia entre classes de um *framework*



Fonte: (PREE, 1995, p. 2)

3.3 PADRÃO *MODEL-VIEW-CONTROLLER* (MVC)

O padrão *Model-View-Controller* (MVC) é considerado um conjunto de padrões organizacionais essenciais para que as aplicações tenham qualidade, uma vez que sua essência é uma adequação de responsabilidades para módulos específicos e bem definidos, ou seja, trazendo agilidade no desenvolvimento e na manutenibilidade de software. Muitas aplicações, ferramentas e arcabouços utilizam deste modelo para alcançar benefício arquitetural de *software*. Especificamente, *frameworks* podem se beneficiar do MVC pela atribuição de responsabilidades

em camadas na aplicação, tornando-as partes independentes. Por consequência, uma maior flexibilidade para efetuar mudanças em setores específicos sem afetar os demais, melhorando qualitativamente o processo de desenvolvimento e testes. Por exemplo, um projeto de desenvolvimento de uma aplicação Web pode se tornar grande, consequentemente, precisa-se de uma estruturação bem definida. Assim como é mostrado na Figura 5, o MVC provê uma solução que é apresentada em três camadas coesas: *model, view, controller*. Cada uma assume operações relacionadas ao domínio da aplicação. Há uma troca de informações entre os objetos internos deste padrão, que segue uma hierarquia definida.

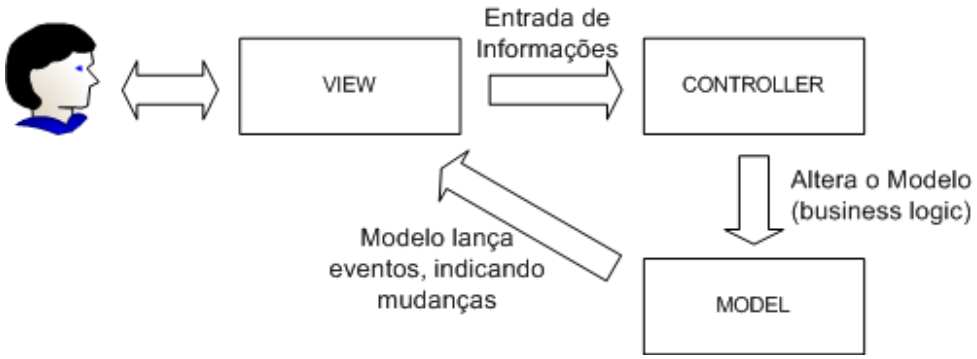
Na Figura 5, tem-se a camada *Model* é responsável por manter as informações, lógicas de manipulação e negócios, comportamentos e características intrínsecas dos objetos que compõem o domínio da aplicação. Tendo como exemplo, um objeto Pessoa na aplicação é um componente objeto da camada modelo. Contém dados que descrevem atributos que o definem, como o primeiro nome e data de nascimento, e tem comportamentos (métodos) que possibilitam o acesso e sua modificação, os chamamos *getters* e *setters*. A camada de modelo é caracterizada por ser reutilizável, distribuída, persistente e pode ser aplicável em várias plataformas.

Em seguida, a camada *View* representa objetos que compõem a exibição da interface do usuário da aplicação, v.g., um componente canvas, um painel ou botão. Geralmente, a interface é usada para representar os dados dos objetos do modelo, como um gráfico, por exemplo. Comumente, cada *view* apresenta seu próprio objeto controlador, com funcionalidades específicas a ela. Seguindo o exemplo dado no tópico da camada *Model*, um controlador faz a interceptação do primeiro nome de um objeto Pessoa (modelo), a partir de dados digitados em uma caixa de texto, visível na UI de uma página web (*view*).

A camada *Controller* (ver Figura 5) obtém os dados de entrada a partir dos objetos da UI (*view*) e os leva à camada de modelo, em formato preferencial. Isto é, o controlador é responsável pela mediação, comunicação e gerenciamento de dados (entrada e saída) entre os objetos das camadas de apresentação (*View*) e modelo (*Model*). O *controller* atua juntamente com a camadas de serviços mais baixas do sistema. Segundo (FOWLER et al., 2002, p. 137), uma camada de serviços (mostrado na Figura 5) define um conjunto comum de operações disponíveis para vários tipos de clientes e coordena a resposta de uma aplicação em cada operação.

Portanto, ele executa todos serviços mais específicos da aplicação e necessários para atualização do modelo, a fim de garantir que sejam adotadas regras de negócios e que os eventos de alteração do modelo sejam disparados, assim como o processamento de uma entrada do usuário ou o retorno de um arquivo do servidor.

Figura 5 – Arquitetura do padrão MVC



Fonte: Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm>. Acesso em: 27 out. 2017

4 JuWeb

Os atuais suportes para programação Web disponíveis para a linguagem Julia são limitados, caracterizando-se por suporte à programação de requisições cliente-servidor como um *Common Gateway Interface* (CGI), disponibilizando um container com pelo menos um servidor compartilhador de recursos. Atualmente, esse serviço está implementado em alguns pacotes voltados à produção de serviços web para Julia, como HTTP.jl, HttpServer.jl, JSON.jl e Mustache.jl, descritos na Seção 2.3.

Contudo, apesar desses suportes para programação Web em Julia, a responsabilidade pelo tratamento e controle entre os processos e os objetos fica a cargo do desenvolvedor. Por exemplo, não existem gerenciadores de rotas das requisições de clientes. Ou seja, a partir de uma requisição do cliente, não se tem um controle de redirecionamento de serviços. Dessa maneira, com as camada de roteamento e de controle do arcabouço JuWeb é permitido a realização de tais ações. Ademais, é notória a limitação de documentação e de compatibilidade de algumas funções internas dos atuais pacotes para com versões mais recentes de Julia.

A fim de suprir as limitações das atuais ferramentas disponíveis para o desenvolvimento Web para Julia, este Trabalho de Conclusão de Curso propõe o projeto e a criação do arcabouço JuWeb para o desenvolvimento de aplicações Web em Julia, utilizando o modelo Model-View-Controller (MVC), apresentado na Seção 3.3.

A Tabela que é apresentada na Figura 6 traz um resumo comparativo entre JuWeb e as atuais ferramentas de apoio ao desenvolvimento de aplicações Web para a linguagem Julia. O critério comparativo *Controle de Estados* significa a capacidade que o artefato tem de manter seu controle junto ao resto da aplicação, o critério *Documentação* descreve o quão maduras estão as informações que explicam e ajudam os usuários utilizarem de seus serviços, o critério *Projeto ativo* informa a atual situação da produção do artefato, o critério *Casos de testes* informa se há pelo menos um caso de teste disponível, o critério *Exemplo de Implementação* informa se existe um exemplo prático de utilização da ferramenta, o critério *Finalidade* apresenta o principal objetivo do artefato, o critério *Última Versão de Julia Compatível* cita a última versão de Julia compatível para o artefato, o critério *Licença* descreve a licença atual, o critério *Granularidade* descreve a escala de medida de trabalho do artefato, por fim, o critério *Facilidade de Integração* descreve o grau de facilidade de integração, conforme à codificação e expressividade das linhas de código do artefato.

Figura 6 – Tabela de critérios avaliativos de artefatos

CRITÉRIOS DE COMPARAÇÃO	HTTPSERVER .JL	MUSTACHE .JL	REQUESTS .JL	HTTP .JL	JUWEB
CONTROLE DE ESTADOS	Simples.	Não.	NA.	Não.	Completo, pela arquitetura MVC.
DOCUMENTAÇÃO	Superficial,.	Boa e exploratória.	Superficial.	Completa,.	Básica.
PROJETO ATIVO	Não.	Não.	Não.	Sim.	Sim.
CASOS DE TESTES	Sim.	Sim.	Sim.	Sim.	Sim.
EXEMPLOS DE IMPLEMENTAÇÃO	Sim.	Não.	Não.	Não.	Sim.
FINALIDADE	Compartilhamento simples de rotas para web.	Geração de objetos para camada de apresentação.	Efetuar requisições.	Dependência de Julia (protocolo HTTP).	Compartilhamento de recursos para web.
ÚLTIMA VERSÃO DE JULIA COMPATÍVEL	Julia v0.4.	Julia v0.3.	Julia v0.4.	Julia v0.6.	Julia v0.4.7.
LICENÇA	MIT License.	MIT License.	MIT License.	MIT "Expat" License.	*Open source.*
GRANULARIDADE	Função e parâmetros.	Função e parâmetros.	Função e parâmetros.	Função e parâmetros.	Camadas, serviços, funções e parâmetros.
FACILIDADE DE INTEGRAÇÃO	Alta.	Baixa.	Alta.	Alta.	Alta.

É perceptível a partir da Tabela as diferenças entre JuWeb e demais artefatos de desenvolvimento Web em Julia. Destaque para os critérios *Controle de Estados*, que é beneficiado pelo arcabouço JuWeb, que por manter em sua arquitetura viável para organização e controle de comportamentos em camadas, graças à arquitetura MVC, explicada na Seção 3.3. Ademais, também é importante frisar que o critério *Facilidade de Integração* é beneficiado ao se utilizar do arcabouço JuWeb, visto que ele diminui a complexidade de relacionamento entre objetos, através do reúso de software proveniente de sua modelagem expansiva.

Nas próximas seções, serão apresentados os detalhes pertinentes à contribuição do trabalho, maiores detalhes das tecnologias utilizadas na implementação do arcabouço JuWeb, baseando-se nas informações conceituais e projetuais descritas nos capítulos anteriores.

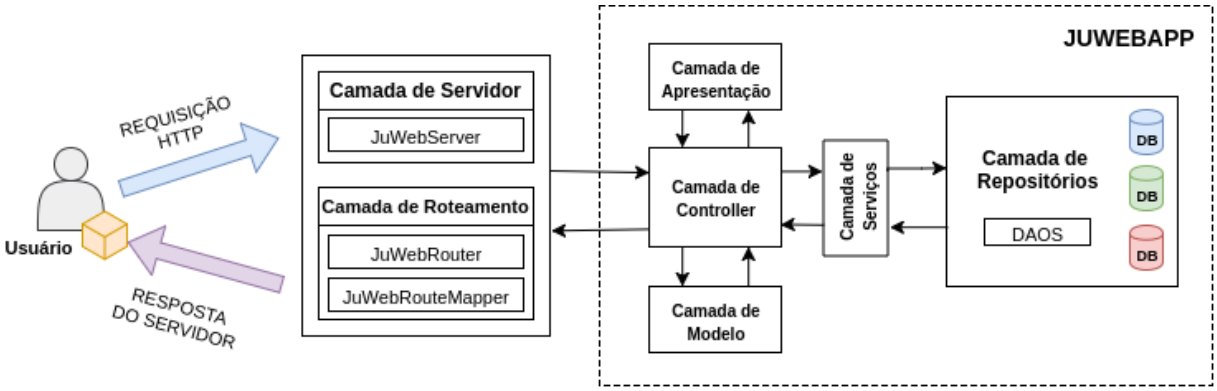
4.1 ARQUITETURA

Esta seção apresentará os componentes que compõem o arcabouço JuWeb, bem como a explanação de cada um deles, descrevendo seu contexto de responsabilidade e atuação, perante todo o processo.

4.1.1 VISÃO GERAL

O JuWeb é dividido em módulos (apresentados na Figura 7), cada um com determinadas funções e responsabilidades, em conformidade às ideias de (FOWLER, 1999), concebida pela aplicabilidade efetiva de reúso de software para ganhos em qualidade (explorada no Capítulo 3). Desta forma, o programador encontrará características que beneficiará a modularização, reutilização, extensibilidade, e consequentemente, trará facilidades e maior qualidade durante o processo de desenvolvimento de sua aplicação (como é informado na Seção 3.2).

Figura 7 – Arquitetura do arcabouço JuWeb



A arquitetura de JuWeb foi projetada seguindo o paradigma arquitetural MVC, explanado a partir da Seção 3.3, e possui componentes que fornecem um fluxo de trabalho simplificado e eficiente para o desenvolvimento de aplicações web, serviços, assim não lidando com detalhes de baixo nível como soquetes, protocolos, processos gerenciáveis, *threads*, dentre outros. Consiste em arquitetura aberta, onde outros desenvolvedores terão livre acesso ao código fonte, isto é, desenvolvida sob a licença de direitos de propriedade intelectual livre. Classificado como um arcabouço web de infraestrutura, conforme descrito dentre as classificações de escopo definidas por (FAYAD; SCHMIDT, 1997, p. 34), tem como principal objetivo, melhorar a capacidade de trabalho de desenvolvedores, à vista disso, sendo um facilitador no processo de produção de software web. A seguir, são descritos os conjuntos de responsabilidades das camadas pertencentes à arquitetura de JuWeb.

4.1.2 CAMADA DE ROTEAMENTO

Esta camada age como um "*proxy*" entre as requisições via URL e os métodos dos *controllers*. Quando uma rota é encontrada, o *Router* inclui a assinatura do respectivo método correspondente e o invoca, passando os argumentos da requisição para a tupla de controladores, responsável pelo retorno da ação desejada. Pode ser adicionado parâmetros adicionais que poderão ser empacotados na definição da rota. Eles serão passados para o módulo de controle como parte do parâmetro de dicionário.

Existe na camada um objeto responsável pelo tratamento e gerenciamento de todos Requests (requisições) efetuados ao JuWebServer e também pelo retorno da correta *Response* (resposta do servidor) ao usuário, o roteador JuWebRouter. É o ponto de partida e de retorno de uma chamada HTTP. Podemos afirmar, que é o meio que analisa a requisição HTTP, e em caso de veracidade quanto a existência de algum serviço para a determinada rota chamada, o *request* é encaminhado para o método mapeado, previamente, em um *controller*.

`register_controller(method::Any, url::Any, controller::Function)`: mapeia o serviço do controlador e sua respectiva rota, definida por uma expressão regular.

`callback(req::Request, res::Response)`: handler de requests e responses, responsável pela análise das requisições efetuadas por usuários e pelo redirecionamento de serviços internos.

`controller_not_found()`: método padrão para requisições indefinidas.

`is_static_file(resource::AbstractString)`: responsável pela análise de um possível arquivo no servidor, por meio de um diretório enviado como parâmetro.

`file_path(resource::AbstractString)`: retorna a pasta principal de recursos.

`serve_static_file(req::Request, res::Response)`: método que faz o parser de um arquivo encontrado no servidor, como resposta de uma requisição.

Um mapeador de rotas, chamado JuWebRouteMapper, atua em conjunto com o roteador JuWebRouter, sendo o responsável principal pelo mapeamento das rotas, guardando as informações dos tipos de métodos HTTP, as URI's para acesso aos recursos e seus respectivos serviços (métodos internos aos controllers).

`map_routes()`: responsável pelo mapeamento de todos os serviços do sistema, disponibilizando-os a partir de seu método, url de requisição e função de controle.

Por exemplo, para o mapeamento de um serviço para exibição de uma página de login, temos:

```
router.register_controller(GET,"/login( \/)?$",login_controller.login_page)
```

4.1.3 CAMADA DE MODELO (*ABSTRACTMODEL*)

Representa as informações do domínio da aplicação, responsável pelas análises de regras que são definidas nos cenários reais envolventes nas funcionalidades dos dados da própria aplicação. Há uma gerência programática do *software*, em forma de serviços, no qual se preocupam em realizar um controle de execuções internas da aplicação a partir da *business logic*. Esta camada guarda os estados do negócio e se relaciona diretamente com a camada de *Controllers*. Ademais, o modelo é capaz de acessar e modificar seus próprios atributos a partir de suas funções encapsuladas, presentes em cada um dos objetos, i.e., seus comportamentos.

`AbstractModel`: classe genérica de suporte aos modelos do JuWeb.

```
type TrackModel      <: AbstractModel

  #attributes
  track_id
  name
  album_id
  media_type_id
  genre_id
  composer

  functionTrackModel()
     return new()
  end

end
```

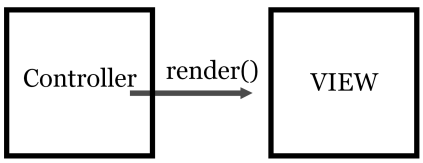
Modelagem do objeto *TrackModel*, na camada de modelo do JuWeb

Fonte: Disponível em: <https://github.com/lucdms/JuWeb.jl> Acesso em: 18 dez. 2017.

4.1.4 CAMADA DE *CONTROLLERS*

Módulo que tem como responsabilidade a orquestração das trocas de dados entre as outras partes do sistema, mais especificamente, os módulos de serviços e repositório. Em suma, como é mostrado na Figura 8 os *controllers* do JuWeb mapeiam as interações dos usuários com a interface genérica do modelo, efetuando alterações e controlando o fluxo de retorno dos serviços para as *views*.

Figura 8 – Comunicação entre um Controller e VIEW do JuWeb



4.1.5 CAMADA DE SERVIÇOS (*BUSINESS LAYER*)

JuWeb apresenta camadas de serviços, atua como intermediária entre as trocas e ações de dados entre as camadas de controle e apresentação. Assim como (FOWLER et al., 2002) afirma, na Seção 3.3, a adição de uma camada de serviços é importante para a gerência das respostas que envolvem vários tipos de papéis de clientes e que a lógica da aplicação seja avaliada, e posteriormente, processada em vários recursos transacionais. JuWebApp apresenta sua lógica de

negócios bem definida, pois a geração de respostas complexas são melhores gerenciadas, através de múltiplos serviços auxiliares.

4.1.6 CAMADA DE APRESENTAÇÃO (*VIEW*)

Camada da aplicação mais próxima do usuário, na qual é responsável pelo tratamento das informações de entrada e saída, e da maneira de como será apresentada. A camada de apresentação do JuWeb está definida no objeto View.jl, que utiliza do pacote Mustache.jl para renderizar as respostas da aplicação em páginas HTML e seus componentes, como é explicado na Seção 2.3.

`render()`: método responsável por renderizar uma página HTML e retorná-la via serviços.

4.1.7 CAMADA DE REPOSITÓRIOS (DAOs) (JuWebRepository)

Camada mais próxima ao banco de dados, funciona a partir de uma interface que se conecta aos *drivers* de um banco para efetuar consultas, inserções e retornar valores necessários nas realizações de serviços a partir de instruções SQL. JuWeb utiliza seu repositório com conexão única para um banco de dados SQLite, sendo necessária a dependência SQLite.jl para seu funcionamento. Por exemplo, como é mostrado no código abaixo, para a listagem dos objetos do tipo TrackModel, é preciso chamar a função list() do objeto TrackDAO, que utiliza dos serviços da camada de repositório.

```
1 function list()
2     bd = JuWebRepository.get_db()
3     sql = "SELECT * FROM tracks"
4     query=SQLite.query(bd,sql)
5
6     #list
7     list = TrackModel[]
8     (...)
9 end
```

Função de listagem de objetos *Tracks*, na camada de repositório do JuWeb

Fonte: Disponível em: <https://github.com/lucdms/JuWeb.jl> Acesso em: 18 dez. 2017.

4.1.8 CAMADA DE SERVIDOR

O servidor do JuWeb,chamado JuWebServer, utiliza o mecanismo denominado *Event loop*, responsável pelo gerenciamento de eventos. Na prática, como é visível na definição do

método do código abaixo a função **`run()`** é responsável por iniciar um servidor Web juntamente com seu *handler callback* **`application.handler_request (req,res)`** o qual é executado assim que o servidor recebe uma requisição, mais especificamente na linha 2. Isso ocorre porque o *Event loop* fica escutando se o servidor recebe alguma requisição e, caso receba, ele invoca um evento para que seja executado o respectivo *callback* chamado.

```
1  function run()
2    http = HttpHandler((req,res) -> application.handler_request(req,res))
3    http.events["error"]  = (client, error) -> println(error)
4    http.events["listen"] = (port)          -> print_with_color(:yellow,
       Listening on $port...\n")
5    if host=="localhost"
6    host="127.0.0.1"
7    end
8    try
9    IPv4(host)
10   HttpServer.run(Server(http),host=IPv4(host),port=port)
11   catch
12   "only IPv4 addresses"
13   end
14 end
```

Função que executa os serviços do JuWebServer, servidor do JuWeb

Fonte: Disponível em: <https://github.com/lucdms/JuWeb.jl> Acesso em: 18 dez. 2017.

4.2 FLUXO DE EXECUÇÃO

O termo fluxo de controle é um método de descrição de um sistema por meio dos principais blocos de código que controlam sua operação. Inicialmente, o usuário deve executar a aplicação padrão do JuWeb a partir do comando, o usuário entrará em <http://localhost:8080/Polsar>, então ao efetuar a chamada URL será enviada uma solicitação HTTP para o servidor web (JuWebServer), a partir do pedido dessa requisição (*GET*) via browser do cliente.

O JuWebServer, por sua vez, construirá a principal página HTML da aplicação JuWebApp, e a retornará ao usuário através de resposta HTTP, ou seja, o servidor web precisa ser capaz de retornar formatos de arquivos diferentes e o navegador deverá distingui-los. Por fim, o conteúdo dessa resposta será interpretada e exibida através do processamento do lado do cliente, pelo *browser*. As etapas do fluxo podem ser melhores visualizadas a partir da arquitetura geral apresentada na Figura 7, que inicia pela da solicitação do usuário através de um *browser*, até que esta seja encaminhada pelo JuWebRouter para um controlador frontal, com base no mapeamento

definido em JuWebRouteMapper. Ou seja, o JuWebRouter utiliza das informações capturadas do mapeamento e encaminha o *callback* para sua respectiva solicitação, em seu controlador correspondente, que processa o *callback* da solicitação, usa a camada de de modelo e serviços, para executar as regras de negócio e lógicas de domínio. Por fim, pela camada de apresentação, JuWeb usa o resolvedor de visualizações Mustache e envia os resultados obtidos das camada de serviços para a de apresentação, onde será construída uma *view* de resposta, que novamente pela camada de roteamento será retornada ao navegador do usuário.

5 ESTUDO DE CASO: APLICAÇÃO HUGEPOLSARWEB

5.1 POLSAR

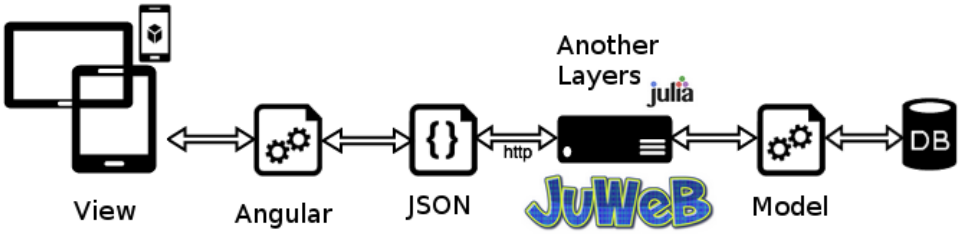
PolSAR.jl é um pacote, desenvolvido na linguagem Julia, que implementa algoritmos de processamento de dados de sensores do tipo *Polarimetric SAR (Synthetic Apperture Radar)*. Esses algoritmos realizam cálculos em dados de sensores PolSAR. Por exemplo, o UAVSAR (*Uninhabited Aerial Vehicle Synthetic Aperture Radar*), desenvolvido pelo *Jet Propulsion Laboratory* da *National Aeronautics and Space Administration* (NASA), coleta dados de sensores do tipo PolSAR e os disponibiliza publicamente. Para que as as informações coletadas por esses radares possam ser visualizadas por seres humanos é necessário realizar uma série de operações e colorir artificialmente cada pixel coletado. Um exemplo de algoritmo implementado pelo PolSAR.jl é a decomposição de Pauli, que permite gerar imagens para a identificação de áreas urbanas, hidrográficas e com vegetações, por exemplo.

O pacote PolSAR.jl possui uma interface através de funções em Julia. Atualmente, não exite uma interface gráfica, Desktop ou Web, para processar dados PolSAR em Julia. Dessa maneira, foi utilizado essa aplicação como estudo. Além disso, a implementação de uma aplicação Web com código livre é de grande relevância pois promove a utilização de dados PolSAR para a comunidade científica assim como permite que o público leigo possa extrair informações de dados PolSAR facilmente através de uma interface gráfica na Web.

5.2 HUGEPOLSARWEB

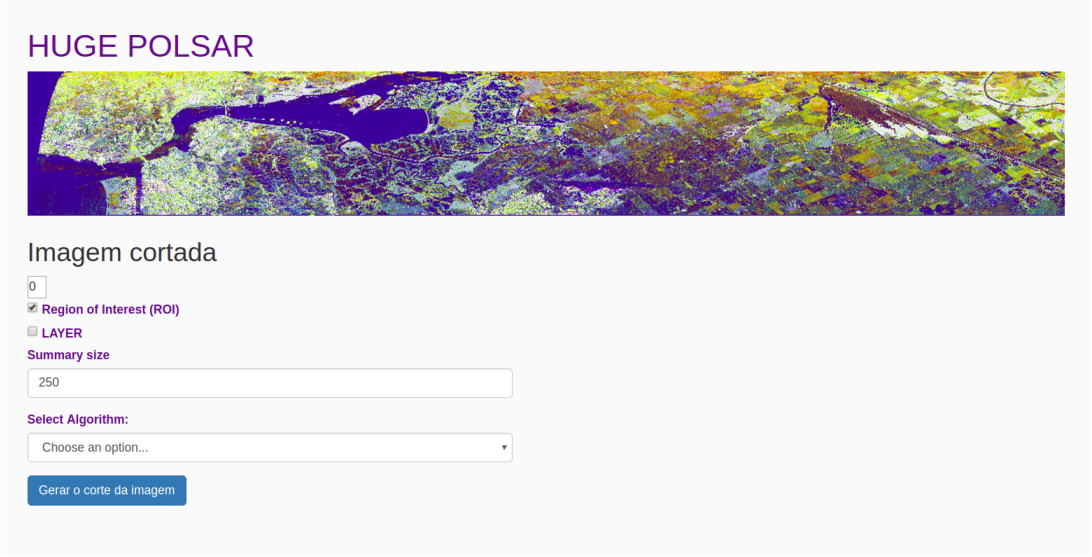
A aplicação HugePolSARWeb é uma aplicação complementar ao projeto PolSAR.jl (vide Seção 5.1). O objetivo dessa aplicação é permitir que imagens robustas sejam facilmente visualizadas e processadas, através de uma interface Web. Essas imagens são enormes matrizes de números complexos que alcançam dezenas de *gigabytes* ou mais. Então a aplicação HugePolSARWeb, a partir da organização arquitetural fornecida por JuWeb, poderá prover serviços para o controle do processamento dessas imagens, por manter um *back-end* ativo em Julia com compartilhamento de recursos na web. A Imagem 9 mostra o fluxo de controle da aplicação, e logo após há uma descrição de seu funcionamento.

Figura 9 – Fluxo de Controle do HugePolSARWeb



Ao efetuar a requisição para a página principal da aplicação HugePolSARWeb, pela URL <http://127.0.0.1:8000/polsar>, inicialmente, o sistema apresenta uma estrutura em canvas que resolve e desenha a imagem de satélite na página inicial, possibilitando ao usuário a aplicação de algoritmos de processamento de imagem em uma ROI[5] definida por ele, ou seja, efetua a seleção de uma subimagem a sua escolha, como mostrado na Figura 10:

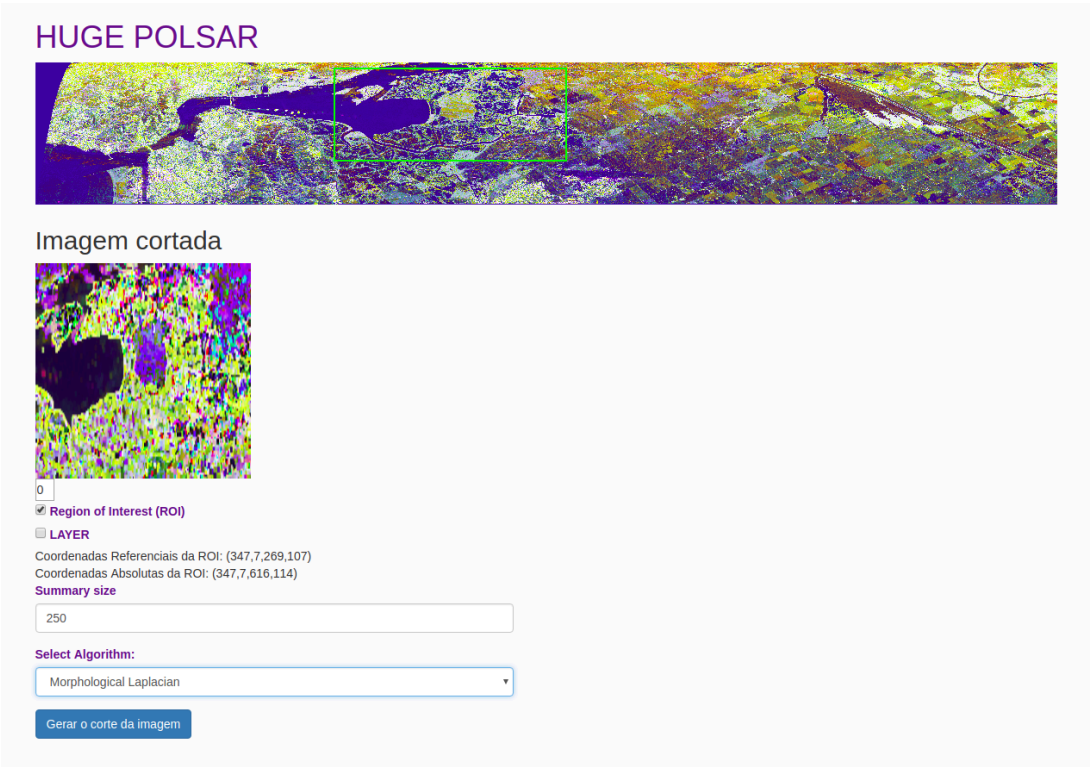
Figura 10 – Tela inicial do sistema



O usuário seleciona uma parte dessa imagem, define o *summary size* (tamanho do resumo da imagem) a ser aplicado, escolhe o algoritmo a ser aplicado em sua seleção, e por fim, basta que se clique no botão “Gerar corte da Imagem” para que o navegador envie uma requisição para o servidor, então gerará o corte da imagem, processará o algoritmo pedido e retornará uma imagem com extensão PNG ao navegador do cliente como resultado, assim como a Figura 11.

5 Região de Interesse - Seleção de uma área do mapa.

Figura 11 – Efetuando o corte da imagem de satélite e aplicando "*Morphological Laplacian*" (algoritmo de processamento de imagem)



6 AVALIAÇÃO

Este capítulo apresenta os objetivos da avaliação realizada no JuWeb. Assim, este capítulo apresenta a metodologia empregada, os cenários que foram utilizados para realização das avaliações e os resultados obtidos.

6.1 OBJETIVOS E MÉTRICAS

Os experimentos realizados durante este trabalho tiveram como objetivo capturar informações avaliativas acerca dos resultados gerados das aplicações desenvolvidas a partir da infraestrutura de JuWeb. Especificamente, no que diz respeito ao desempenho e retorno esperado das respostas ao lidar com processamentos a partir da troca de requisições HTTP entre duas máquinas, cliente e servidor. Portanto, a avaliação de JuWeb foi feita através de experimentos focados no tempo de execução específico para cada uma das camadas do arcabouço como métrica para mensurar o tempo, medidos através das coletas de tempos decorridos entre nas linhas de código correspondentes a cada etapa procedural do arcabouço, a partir das funções `tic()` e `toc()` de Julia, funções nativas da linguagem que permitem a impressão do tempo decorrido entre o intervalo das chamadas subsequentes de `tic()` e `toc()`. Também foi aplicado o uso da ferramenta POSTMAN[6] na máquina servidor, como métrica avaliativa para respostas esperadas, que se deu por testes de serviços compartilhados pela aplicação no POSTMAN, assegurando a adequação comportamental entre os processos e trocas de mensagens entre as camadas de JuWeb.

6.2 CENÁRIOS E CONFIGURAÇÃO DO AMBIENTE DE EXPERIMENTOS

Com o intuito de avaliar todas as etapas procedurais do arcabouço, desde os processos de mapeamento das rotas dos serviços (execução inicial de JuWeb) até o retorno da resposta para o cliente, foram subdivididos em 5 fases e mensuradas separadamente:

- Fase 1: Mapeamento de rotas em JuWebRouteMapper e encaminhamento para o JuWebRouter;
- Fase 2: Encaminhamento do *callback* da solicitação ao controlador;
- Fase 3: Processos das camadas de domínio (*domain logic*);

[6] Aplicação que permite a realização de requisições no protocolo HTTP a partir de uma GUI simples e intuitiva, facilitando o teste e depuração de serviços web.

- Fase 4: Camada de Apresentação: Geração da *view* a ser retornada ao controlador;
- Tempo total: da execução até o recebimento da resposta pelo cliente.

Figura 12 – Cenário 1 de avaliação

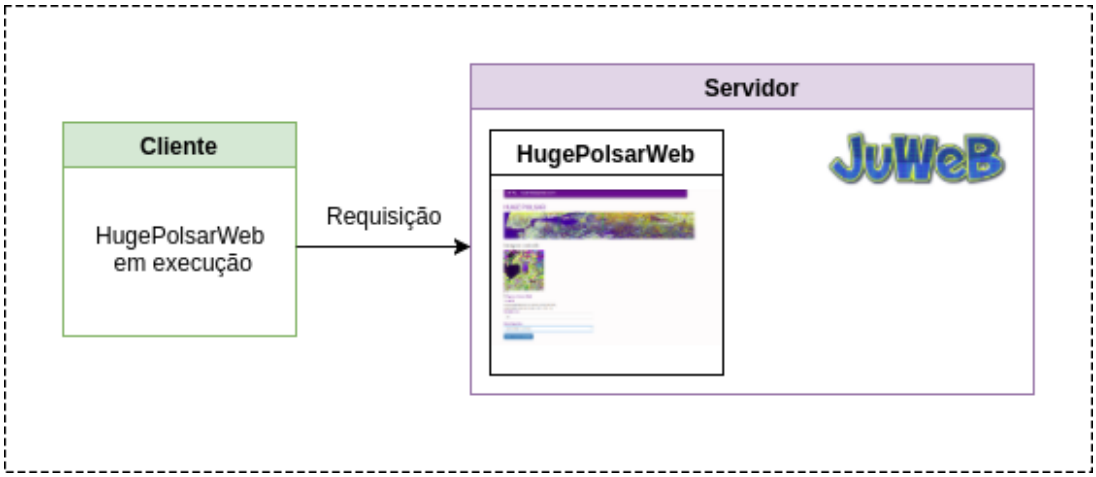


Figura 13 – Cenário 2 de avaliação

Com relação à configuração do ambiente de experimentos, conforme apresentado na Figura 12, no cenário 1 foram utilizadas duas máquinas cujos detalhes serão descritos abaixo. Dentre as duas máquinas, uma foi configurada especificamente para executar toda a infraestrutura do JuWeb, consequentemente, exercendo a função de servidor e a outra a de cliente. Para a realização da métrica de respostas esperadas, foi utilizado no cenário 2 o teste de serviços, a partir de uma requisição para a ferramenta POSTMAN, como foi mostrado na Figura 13.

A máquina servidor apresenta processador Intel R CoreTM i7-3537U CPU @2.00GHz (4 núcleos), 7.7GiB de memória, placa gráfica Intel R Ivybridge Mobile e Sistema Operacional

Ubuntu 16.04 LTS, 64-bit. Encontra-se na máquina os requisitos adequados, os *softwares* e dependências necessárias para a execução da infraestrutura de aplicações da linguagem Julia, do arcabouço JuWeb e da aplicação HugePolSARWeb. A seguir, são listados os *softwares* instalados, pacotes da linguagem Julia adicionados e suas respectivas versões: Julia 0.4.7, Ubuntu 16.04 LTS, JuWeb e suas dependências em Julia: HttpServer.jl, Mustache.jl, Requests.jl, SQLite.jl, JSON.jl, ImageView.jl, QuartzImageIO.jl, ImageMagick.jl.

A máquina cliente apresenta processador AMD Athlon II Dual-Core M300 CPU 2.00GHz, 3.0GiB de memória, placa gráfica Mobility Radeon HD4200 e Sistema Operacional Windows 7, 64-bit. Ela exerce a função de consumidora dos recursos compartilhados pela máquina servidor, para isso seu conjunto de *softwares* instalados se restringe a apenas o navegador Google Chrome na versão 62.0.3202.62 (Versão oficial para 64 bits), que foi utilizado para realizar as requisições.

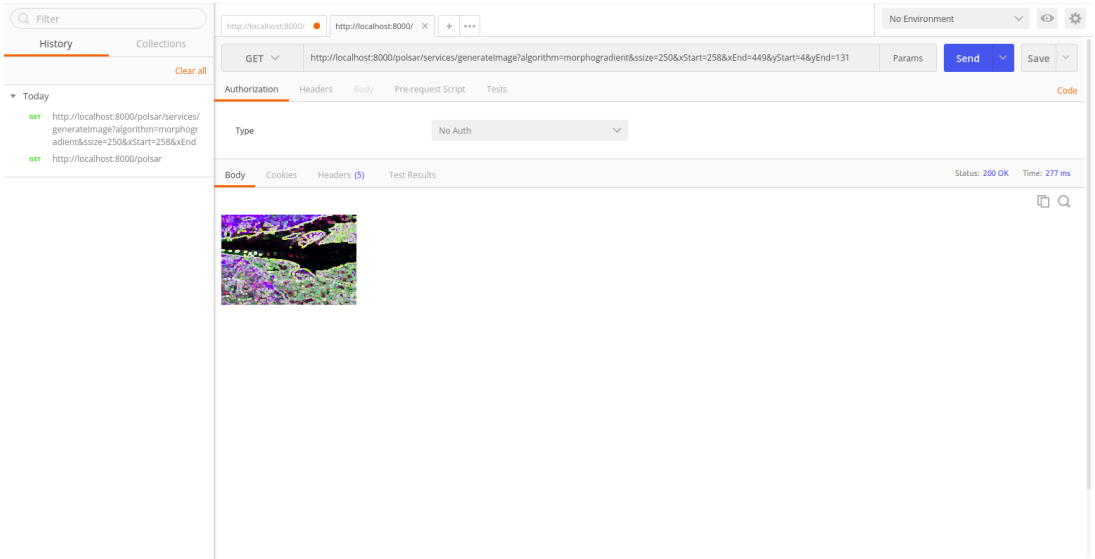
6.3 RESULTADOS

A Figura 14 exibe os resultados da avaliação da métrica de desempenho, pelos tempo de execução (ver Seção 6.1) para cada uma das fases apresentadas na Seção 6.2. A aplicação HugePolSARWeb, desenvolvida com o JuWeb, executou todas as requisições alcançando o resultado esperado, ou seja, a entrega da resposta de cada requisição GET retornou o código HTTP 200 (OK), que é a resposta de status de sucesso, indicando que a requisição foi bem sucedida. Além disso, o tempo total de execução obteve mediana de 0.1002444 segundos para todos os processos até a montagem do objeto resposta e despacho pelo handler do *framework* (ver Fase 5 na Figura 14). A fase 5 corresponde ao tempo total para responder à uma requisição, ou seja, o somatório das demais fases.

Figura 14 – Tempo de execução para cada uma das fases mensuradas do JuWeb.

Para o cenário 2, após a chamada do método GET, pela ferramenta POSTMAN, o serviço responsável pelo processamento e geração da imagem foi chamado pelo servidor, a montagem do objeto resposta (imagem de extensão PNG processada) foi corretamente desempenhada pelo arcabouço JuWeb, e por fim o handler e controlador da aplicação retornou o objeto com status 200 OK, em um tempo máximo de 277 ms, performance aceitável para um tempo de resposta, quando se trata de serviços REST. Portanto, pelo resultado encontrado pela ferramenta POSTMAN, os processos entre as camadas foram desempenhados corretamente.

Figura 15 – Resposta do serviço da aplicação HugePolSARWeb, via POSTMAN



7 CONCLUSÕES

Este trabalho apresenta o arcabouço JuWeb, um suporte de programação Web para aplicações desenvolvidas na linguagem Julia. O JuWeb utiliza a modelo *Model-View-Controller* (MVC) e possui arquitetura modular, incluindo aspectos fundamentais de arcabouço como reutilização e extensibilidade. O objetivo principal do JuWeb é melhorar a capacidade de trabalho de desenvolvedores de aplicações Web em Julia, assim, facilitando o processo de produção de *software* Web de aplicações científicas já existentes ou novas.

Para validar a aplicabilidade do arcabouço JuWeb, este foi utilizado para desenvolver a aplicação HugePolSARWeb, que processa dados de sensoriamento remoto de dados PolSAR (*Polarimetric Synthetic Apperture Radar*). O JuWeb permitiu melhorar a produtividade do desenvolvimento da aplicação Web e manteve resultados aceitáveis de desempenho. Por exemplo, as requisições GET (HTTP) foram respondidas aproximadamente em 0.10 segundos, o que era esperado para o ambiente de teste.

Ademais, este trabalho traz uma série de questões que ainda precisam ser investigadas. Primeiramente, é possível melhorar o desempenho das bibliotecas e objetos incluídos nos diversos módulos de JuWeb, para uma maior eficiência na execução de suas aplicações. Outras recomendações decorrentes deste trabalho referem-se especificamente aos módulos implementados no arcabouço:

- Adicionar mais tipos abstratos para representar as camadas de JuWeb;
- Uma macro para a organização das dependências;
- Avaliar a possibilidade de adição de parâmetros em um objeto de sessão;
- Utilizar uma metodologia genérica para geração de *views*, a partir de chamadas como: *View* (*Abstract Model*), possibilitando também saídas definidas por parâmetros;
- Acompanhar as novas versões de Julia, visto que ainda existem incompatibilidades;
- Verificar possíveis causadores de limitações de desempenho, para que esses sejam otimizados em futuras atualizações.

REFERÊNCIAS

APEL, S.; BATORY, D.; KSTNER, C.; SAAKE, G. **Feature-Oriented Software Product Lines: Concepts and Implementation**. Springer Publishing Company, Incorporated, 2013. ISBN 978-3642375217. Disponível em: <https://books.google.com.br/books?id=4WUnAQAAQBAJ>.

BEZANSON, J.; EDELMAN, A.; KARPINSKI, S.; SHAH, V. B. Julia: A fresh approach to numerical computing. **SIAM Review**, v. 59, p. 65–98, 2017.

BRAUDE, E.; BERNSTEIN, M. **Software Engineering: Modern Approaches**. 2. ed. Long Grove, Illinois: Waveland Press, Inc., 2016. ISBN 978-1478633037.

DONGARRA, J.; STERLING, T.; SIMON, H.; STROHMAIER, E. High-Performance Computing: Clusters, Constellations, MPPS, and Future Directions. 2005. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.8852&rep=rep1&type=pdf>. Acesso em: 17.01.2018.

EU. High Performance Computing PPP: Mastering the next generation of computing technologies for innovative products and scientific discovery. 2013. Disponível em: <http://ec.europa.eu/research/press/2013/pdf/ppp/hpc_factsheet.pdf>. Acesso em: 16.01.2018.

EZELL, S. J.; ATKINSON, R. D. The Vital Importance of High-Performance Computing to U.S. Competitiveness. **Information Technology & Innovation Foundation**, 2016. Disponível em: <http://www2.itif.org/2016-high-performance-computing.pdf>. Acesso em: 17.01.2018.

FAYAD, M.; SCHMIDT, D. Object-oriented application frameworks. v. 40, 10 1997.

FIRESMITH, D. G. Frameworks: the golden path to object nirvana. **Journal of Object-Oriented Programming**, p. 5–8., 1993.

FOWLER, M. **Refactoring: Improving the Design of Existing Code**. Boston, MA, USA: Addison-Wesley, 1999. ISBN 0-201-48567-2, 978-0201485677.

FOWLER, M.; RICE, D.; FOEMMEL, M.; HIEATT, E.; MEE, R.; STAFFORD, R. **Patterns of Enterprise Application Architecture**. Boston, MA, USA: Addison-Wesley Professional, 2002. ISBN 0-321-12742-0, 978-0321127426.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2, 978-0201633610.

HttpServer.jl. Repositório GitHub. Disponível em: <https://github.com/JuliaWeb/HttpServer.jl>.

HTTP.jl. Repositório GitHub. Disponível em: <https://github.com/JuliaWeb/HTTP.jl>.

Java. Disponível em: <https://www.java.com>.

JSON.jl. Repositório GitHub. Disponível em: <https://github.com/JuliaIO/JSON.jl>.

KARPINSKI, S. Out in the open: Man creates one programming language to rule them all. 2014.

KRASNER, E. K.; POPE, S. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. **Journal of Object Oriented Programming**, p. 26–49, 1988.

Low-Level Virtual Machine. Disponível em: <https://llvm.org/>.

MATLAB. Disponível em: <https://www.mathworks.com/products/matlab.html>.

MCILROY, M. D. Mass produced software components. **NATO Software Engineering Conference Report**, p. 79–85, 1968.

MIT. The MIT License. 2009. Disponível em: <http://opensource.org/licenses/MIT>.

Mustache.jl. Repositório GitHub. Disponível em: <https://github.com/jverzani/Mustache.jl>.

MORISIO, M.; EZRAN, M.; TULLY, C. Success and fuilure factors in software reuse. IEEE Transactions on Software Engineering, v. 28, p. 340–357, 2002.

PHP. Disponível em: <https://php.net>.

PolSAR.jl. Disponível em: <https://github.com/gsd-ufal/HugePolSAR.jl>.

Python. Disponível em: <https://www.python.org>.

PREE, W. Hot-spot-driven framework development. In: **Summer School on Reusable Architectures in Object-Oriented software Development**. New York, NY, USA: ACM, 1995. p. 123–127.

R. Disponível em: <https://www.r-project.org/>.

Requests.jl. Repositório GitHub. Disponível em: <https://github.com/JuliaWeb/Requests.jl>.

Ruby. Disponível em: <www.ruby-lang.org>.

SCHMIDT, D. C.; GOKHALE, A.; NATARAJAN, B. Leveraging application frameworks. **Queue**, ACM, New York, NY, USA, v. 2, n. 5, p. 66–75, jul. 2004. ISSN 1542-7730. Disponível em: <http://doi.acm.org/10.1145/1016998.1017005>.

SHERRINGTON, M. **Mastering Julia: Develop your analytical and programming skills further in Julia to solve complex data processing problems**. Packt Publishing, 2015. ISBN 9781783553327. Disponível em: <https://books.google.com.br/books?id=P-09CgAAQBAJ>.

SONMEZ, J. Z. **Soft Skills: The Software Developer's Life Manual** Greenwich, Connecticut, USA: Manning Publications, 2014. ISBN 1-617-29239-7, 978-1617292392.

TECHOPEDIA. What is High-Performance Computing (HPC)? Disponível em: <https://www.techopedia.com/definition/4595/high-performance-computing-hpc>. Acesso em: 16.01.2018.

World Wide Web. Disponível em: <https://www.w3.org>.

ZAPALOWSKI, V. **Análise quantitativa e comparativa de linguagens de programação**. Dissertação (Trabalho de conclusão de graduação) — Universidade Federal do Rio Grande do Sul – UFRGS, Rio Grande Sul – RS, 2011. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/31036/000782127.pdf>. Acesso em: 18.12.2017.