UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA - UESB
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS - DCET
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

JOÃO VITOR OLIVEIRA FERRAZ SILVA

LARAaaS:
IMPLEMENTAÇÃO DO LARA (LABORATÓRIO REMOTO EM AVA) COMO UM SERVIÇO

Vitória da Conquista - BA
Dezembro de 2018

JOÃO VITOR OLIVEIRA FERRAZ SILVA

LARAaaS:
IMPLEMENTAÇÃO DO LARA (LABORATÓRIO REMOTO EM AVA) COMO UM SERVIÇO

Trabalho de conclusão de curso, para aprovação na disciplina Trabalho Supervisionado II e como requisito parcial para obtenção do título de Bacharel em Ciência da Computação, na Universidade Estadual do Sudoeste da Bahia - UESB

Orientadora: Prof.ª Dra. Maísa Soares dos Santos Lopes

Vitória da Conquista - BA

Dezembro de 2018

RESUMO

Este trabalho apresenta uma proposta de arquitetura orientada à serviços para o LARA (LABORATÓRIO REMOTO EM AVA) através da implementação de um webservice seguindo o padrão REST para substituir o back-end atual a fim de facilitar o desenvolvimento de novas funcionalidades, permitir a comunicação com outras aplicações, delimitar o controle dos dados de entrada e saída do sistema, dar mais segurança nas trocas de dados, e dar suporte a adaptação do sistema às diversas plataformas disponíveis. A implementação deste webservice foi feito utilizando a plataforma nodeJS, em conjunto com os pacotes npm, express e mysql, a organização do projeto seguiu o padrão MVC. A metodologia utilizada durante o desenvolvimento e levantamento de requisitos foi uma adaptação do SCRUM, o Solo SCRUM. Para validar o software desenvolvido foram feitos testes de unidade, utilizando as ferramentas mocha e chai.

Palavras-chave: Laboratório Remoto; REST; Escalabilidade; Arduino; Arquitetura.

ABSTRACT

This paper presents a service-oriented architecture proposal for LARA (REMOTE LABORATORY IN AVA) by implementing a webservice following the REST standard to replace the current backend in order to facilitate the development of new functionalities, to allow communication with other applications, delimit control of the input and output data of the system, give more security in data exchanges, and support the adaptation of the system to the various platforms available. The implementation of this webservice was done using the nodeJS platform, in conjunction with the npm, express and mysql packages, the project organization followed the MVC standard. The methodology used during the development and requirements survey was an adaptation of the SCRUM, the SCRUM Soil. To validate the developed software, unit tests were done using the mocha and chai tools.

Keywords: Remote Laboratory; REST; Scalability; Arduino; Architecture.

AGRADECIMENTOS

Agradeço a todos os professores, em especial, professora Maísa Soares (minha orientadora) que teve uma paciência infinita comigo durante todo o curso e deu suporte total durante a produção deste trabalho, a professora Alzira Ferreira por ter me incentivado a continuar no curso quando ainda tinha dúvidas no início da caminhada, a professor Roque Mendes por sempre fazer seus alunos acreditarem que podem mais, a professor Hélio Lopes pela sua empolgação contagiante, a professor Marco Antônio pelas suas verdades dolorosas, a professor Marlos Marques pelo alto nível de exigência, a professor Stenio Longo pelo profissionalismo. Aos meus pais pelo suporte total, broncas e ensinamentos, caronas, incentivos, em especial a minha mãe que esteve mais de perto no dia-dia, e assim foi até o fim, ao meus avós pelas orações, aos meus tios Marcos e Tata que me ajudaram a comprar o notebook sem o qual não teria conseguido caminhar tranquilamente no curso, ao meu irmão por não se incomodar com a luz acessa durante as madrugadas estudando, aos meus primos, Augusto e Rondinelly pelo companheirismo, Maone e Yuri pela má influência na escolha do curso. Aos meus colegas de curso, Rodrigo pelo título deste trabalho, Wali pelas aulas extras, Leandro pelas caronas, Lucas pelas risadas, Brunna pelos doces, Bernardo por dormir, Matheus por chegar mais cedo que eu. E por fim a todo mundo que participou de alguma forma dessa longa caminhada. Obrigado.

“O pessimista vê dificuldade em cada oportunidade; o otimista vê oportunidade em cada dificuldade. ”
(Winston Churchill)

LISTA DE SIGLAS

LARA - Laboratório Remoto em AVA
AVA - Ambiente Virtual de Aprendizagem
API - Interface de Programação de Aplicação
CGI - Common Gateway Interface
LMS - Sistema de Gestão da Aprendizagem
GOLC - Consórcio Global de Laboratórios Online
REST - Transferência de Estado Representacional
HTTP - HiperText Transfer Protocol
RLMS - Sistema de Gerenciamento de Laboratórios Remotos
SOAP - Protocolo Simples de Acesso a Objetos
UTS - Universidade Tecnológica de Sydney
SGBD - Sistema de Gerenciamento de Banco de Dados
MVC - Model, View e Controller
JSON - JavaScript Object Notation

LISTA DE QUADROS

LISTA DE FIGURAS

SUMÁRIO

INTRODUÇÃO

As aplicações web se tornaram parte importante do dia-dia das pessoas, elas estão em tudo, previsão do tempo, notícias, redes sociais, incontáveis funções, e esse ganho de importância e diversificação das aplicações web tornou necessário a criação de interfaces de comunicação entre esses sistemas de forma a tornar as interações entre elas algo natural. Além disso o crescimento de diferentes plataformas de utilização da web como celulares, tablets, e pôr fim a internet das coisas, fez-se ainda mais necessário o uso das interfaces, chamadas de APIs (Interface de Programação de Aplicativos), de forma a facilitar a troca de dados entre as diferentes plataformas.

O LARA (Laboratório Remoto em AVA) consiste numa plataforma web que permite acesso à um ambiente de desenvolvimento colaborativo, com o objetivo de melhorar o processo de aprendizagem das disciplinas do curso de Ciência da Computação (LOPES, 2017). Atualmente essa plataforma não possui uma interface que possibilite a comunicação com aplicações que utilizem suas funcionalidades e/ou adicionem novas funcionalidades.

Segundo Ngolo (2009, p.2-3) os laboratórios remotos inicialmente foram desenvolvidos como softwares específicos, sendo necessário a instalação na máquina do cliente, posteriormente com a evolução da plataforma web eles migraram para o navegador, foram adotadas novas tecnologias, como o uso de Java applets, mini aplicações que são executadas no navegador do cliente, páginas HTML e scripts CGI (Common Gateway Interface).

Em 2012, Tawfik et al., realizou um estudo sobre o estado das arquiteturas dos principais laboratórios remotos, durante essa pesquisa, ele percebeu uma tendência de mudança nas arquiteturas, com o objetivo de torná-las orientadas a serviços, um movimento que toda a web seguia na época, também foi seguido por eles como uma solução para possibilitar a comunicação entre diferentes instituições e com softwares externos, o principal exemplo seriam os sistemas LMS (Sistema de Gestão da Aprendizagem).

Apesar da arquitetura orientada a serviços permitir a disponibilidade de interfaces entre uma aplicação e outras de terceiros, a comunicação não ocorre de

maneira inata, é necessário um mínimo de padronização no formato dos dados transmitidos na comunicação, e por quais canais a comunicação deve ocorrer, as instituições acabaram utilizando padrões e tecnologias diferentes o que impediu um dos objetivos pretendidos com a utilização das arquiteturas orientadas a serviços, a comunicação entre as diferentes instituições.

Segundo Tawfik et al. (2012), por esse motivo LiLa[1], iLabs[2], Labshare[3], Deusto[4], algumas das principais distribuições de laboratórios remotos resolveram formar o Consórcio Global de Laboratórios Online (GOLC), de forma a pesquisar e incentivar a criação de uma arquitetura unificada e interoperável, capaz de compartilhar laboratórios online ao redor do mundo de forma eficiente. Em 2015 o GOLC propôs um esquema de API e metadados (LOWE et al., 2015, apud ZUTIN, 2018), entretanto até o presente momento nenhum movimento de convergência para um padrão na arquitetura dos laboratórios remotos foi observado (ZUTIN, 2018).

Dessa forma, a fim de reestruturar a arquitetura do LARA, esse trabalho tem como objetivo geral implementar uma API REST para o LARA (Laboratório Remoto em AVA), a fim de facilitar o desenvolvimento de novas funcionalidades, permitir a comunicação com outras aplicações, delimitar o controle dos dados de entrada e saída do sistema, dar mais segurança nas trocas de dados, e dar suporte a adaptação do sistema às diversas plataformas disponíveis e tendo como objetivos específicos, identificar requisitos funcionais e não funcionais para a construção da API; definir uma arquitetura para a API, levando em conta as funcionalidades já estabelecidas e futuras funcionalidades que podem ser inseridas e testar a qualidade do artefato produzido ao longo da pesquisa.

A metodologia de pesquisa escolhida foi a Design Science, que consiste num conjunto de técnicas a serem seguidas com o objetivo de gerar a melhor solução para um problema, essa solução deve ser viável, relevante para o sistema ou negócio alvo da pesquisa, a qualidade dessa solução deve ser demonstrada através

[1] http://www.lila-project.org/

[2] https://wikis.mit.edu/confluence/display/ILAB2/Home

[3] http://www.labshare.edu.au/

[4] https://www.weblab.deusto.es/web/

de teste bem executados e verificáveis, tanto na construção quanto na avaliação dela (HEVNER et al,2004).

O desenvolvimento do projeto seguiu os princípios da metodologia SCRUM Solo, uma adaptação do SCRUM proposta por Pagotto et al. (2016), a qual foi construída com o objetivo de auxiliar desenvolvedores individuais, um cenário comum no país. Segundo pesquisa realizada pela Secretaria de Política de Informática do Ministério de Ciência e Tecnologia aponta que cerca de 60% das empresas de software começaram dessa forma (MINISTÉRIO DE CIÊNCIA E TECNOLOGIA, apud PAGOTTO, 2016).

Nesse processo a quantidade de atores é bem reduzida, sendo constituída principalmente pelo proprietário do produto, desenvolvedor único, responsável por executar o processo e construir o produto, e orientador, caracterizado como um consultor que conhece a fundo o processo. O autor do processo sugere um Grupo de Validação, para testar o produto, o que não foi possível ser feito neste trabalho.

O SCRUM Solo sugere sprints de 1 semana, em compensação não havendo a necessidade das reuniões diárias, sendo a revisão e discussão feitas no momento da finalização do sprint, e se forem necessárias reuniões podem ser marcadas para orientação dos sprints

Durante as orientações eram definidos novos pontos para serem entregues nos encontros, assim como o sprint backlog.

2. REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta o referencial teórico sobre os principais conceitos abordados neste trabalho. A primeira seção trata dos principais conceitos em que se baseia a arquitetura proposta para o LARA, relacionados à arquitetura orientada a serviços. Seguida por uma seção descrevendo a plataforma em que o web service foi implementado, demonstrando suas principais características. Por último, é feito uma análise do estado atual das arquiteturas de outros laboratórios remotos.

2.1 Arquitetura Orientada a Serviços

De acordo com Souza (2006, p.5), pode-se dizer que a arquitetura orientada a serviços divide as aplicações em unidades lógicas, chamadas serviços, esses serviços possuem métodos simples ou recursos mais complexos, que tem como objetivo a distribuição dos mesmos de forma rápida, bem definida e aberta para o cliente. Para tal é possível afirmar que se faz necessário um padrão de comunicação, e uma interface que forneçam um canal de interação bem definido entre cliente e servidor (provedor do serviço).

Ainda segundo o mesmo, o protocolo de comunicação não necessariamente precisa ser um protocolo web, a comunicação pode ocorrer em qualquer meio, nesse trabalho especificamente foi usado o protocolo web, *HiperText Transfer Protocol (HTTP)*, pois um dos objetivos do mesmo é prover serviços para seus clientes. Com base numa análise da características da arquitetura orientada a serviços, Souza (2006, p.9-12) constata que a utilização da mesma traz várias vantagens, entre elas:

- **Modularidade**

Os serviços são inerentemente modularizados, e podem ser acoplados a qualquer outro software contanto que ele siga o protocolo de comunicação, e utilize a interface pré-definida, permitindo a reutilização de softwares legados, a utilização de diferentes linguagens de programação e plataformas de desenvolvimento. As limitações quanto a necessidade de a comunicação ser síncrona pode ser aliviada por uma camada superior que faça a abstração dos dados de forma assíncrona.

- **Interoperabilidade**

Os serviços devem seguir padrões acessíveis e largamente aceitos, de forma a permitir que eles sejam operados em diferentes ambientes.

- **Composição**

Um serviço pode ser composto de outros serviços já implementados previamente, permitindo que novas funcionalidades sejam adicionadas.

- **Reusabilidade**

Por serem fracamente acoplados os serviços podem ser facilmente reutilizados.

- **Alta Granularidade**

Como a aplicação é dividida em pequenos pedaços de código, serviços, ela possui uma alta granularidade, o que facilita a divisão de tarefas em um projeto maior.

2.2 REST (Transferência de Estado Representacional)

A arquitetura orientada a serviços possui diversas formas de ser aplicada. Mais recentemente a arquitetura que vem sendo muito utilizada é a REST que sobrepôs a forma predominante no início da utilização dos webservices, SOAP (Protocolo Simples de Acesso a Objetos), que acabou em desuso devido a sua sobrecarga de dados, desnecessários na maioria dos casos de uso na web atual (MULLIGAN e GRACANIN, 2009).

Segundo Fielding (2000, p.76), o REST é um estilo arquitetural para web services, composto por restrições que tem como objetivo manter os princípios da arquitetura. As restrições consistem em:

A - Cliente-Servidor

A primeira restrição consiste na arquitetura cliente-servidor, que permite uma separação entre a camada que interage com o usuário e a camada que se preocupa com os dados. Essa separação permite que diferentes clientes possam ser utilizados com a mesma estrutura de servidor, aumentando a portabilidade das aplicações. Essa separação também permite que ambas as partes possam se desenvolver independentemente.

B - Sem estado

Essa restrição cobre a comunicação cliente-servidor que não deve possuir estados, ou seja toda requisição do cliente para o servidor deve possuir toda informação necessária e não deve contar com qualquer informação guardada no servidor, dessa forma o cliente deve ser o responsável por guardar o patamar atual da comunicação.

C - Cache

As informações passadas durante uma requisição devem ser marcadas como passíveis de serem guardadas em cache ou não, caso seja possível o cliente tem a possibilidade de guardar dados em cache para que elas sejam re-utilizadas futuramente sem a necessidade de uma nova requisição.

D - Sistema em camadas

Para evitar problemas de escalabilidade foram adicionadas restrições com relação ao sistema de camadas, devido a sua capacidade de melhora na escalabilidade, aumento na flexibilidade durante o desenvolvimento, facilitando divisão de tarefas, revisão de código e manutenção devido à modularidade, apesar de trazer desvantagens como o aumento de complexidade e perda de perfomance, seus benefícios suplantam as suas desvantagens.

E - Code-On-Demand

REST permite ao cliente requisitar código do servidor e executá-lo, nos dias atuais não é uma restrição muito útil, à época que o REST foi definido era algo muito usado com os applets java, é uma restrição opcional.

2.3 HTTP (Protocolo de Transferência de Hipertexto)

Segundo Tanenbaum (p.493, 2003), HTTP é um protocolo de transferência de dados utilizado por toda a web, ele define códigos e verbos (métodos) pelos quais cliente e servidor se comunicam.

Os métodos HTTP consistem na principal forma de o cliente comunicar ao servidor qual o objetivo de sua requisição e a natureza da resposta esperada. Estes métodos são mostrados no Quadro 1.

Quadro 1 - Métodos HTTP

Método	Descrição
GET	Transfere a representação atual do recurso alvo.
HEAD	O mesmo que o GET mas apenas transfere o status e a seção header.
POST	Faz o processamento da carga de dados vinda na requisição.
PUT	Substitui todas as representações atuais do recurso alvo com a carga de dados advinda da requisição.
DELETE	Remove todas as representações atuais do recurso alvo.
CONNECT	Estabelece um túnel para o server identificado pelo recurso alvo.
OPTIONS	Descreve as opções de comunicação para o recurso alvo.
TRACE	Envia uma mensagem loop-back de teste ao longo do caminho para o recurso alvo.

Fonte: [RFC 7231 - Seção 4]

Toda requisição possui um status no retorno com o objetivo de informar ao cliente a natureza do retorno, um cliente não necessariamente precisa reconhecer todos os códigos mas deve ao menos reconhecer todas as classes de código de forma que ao receber um código desconhecido ele possa classificá-lo de forma que ele representa sua classe (Quadro 2).

Quadro 2 - Códigos HTTP

Código	Significado	Exemplos
1xx	Informação	100 = Servidor aceita requisição do cliente
2xx	Sucesso	200 = Requisição bem sucedida; 204 = Nenhum conteúdo presente
3xx	Redirecionamento	301 = Movido; 304 = Não Modificado
4xx	Erro do cliente	403 = Proibido; 404 = Não Encontrado
5xx	Erro do servidor	500 = Erro interno no servidor; 503 = Serviço Indisponível

Fonte: Tanenbaum, pág 494-495.

2.4 NodeJS

NodeJS é uma plataforma de desenvolvimento que utiliza a linguagem javascript. Na página oficial[5] da plataforma, ela é descrita como javascript orientado a eventos assíncronos em tempo de execução, desenhado para construir aplicações em rede escaláveis (NODE.JS FOUNDATION, 2018).

A grande vantagem do node é sua natureza assíncrona que permite mesmo uma linguagem *single-thread* como a javascript lidar de forma concorrente com operações de entrada e saída, utilizando as capacidades *multi-thread* do kernel do sistema operacional, e empilhando as diferentes instruções para serem executadas assim que possível (NODE.JS FOUNDATION, 2018).

Além da questão de performance, o node também oferece acesso ao npm (Gerenciador de Pacotes para o NodeJS), que é o maior registro de pacotes do mundo atualmente, esses pacotes oferecem as mais diversas bibliotecas (roteamento, segurança, streams, ...) que podem ser acopladas a aplicação, agilizando o tempo de desenvolvimento (BROWN, 2018).

2.5 Laboratório Remoto

Laboratórios remotos são aplicações que permitem estudantes, professores e pesquisadores a conduzir experimentos remotamente, eles são disponibilizados em diferentes formas de aplicações clientes como: web browser, sistemas desktop e sistemas embarcados. Essa aplicação cliente deve interagir com um servidor remoto que controla o experimento em questão (ZUTIN, 2018).

Todo laboratório remoto deve gerenciar pelo menos algumas dessas funcionalidades: autenticação, autorização, e ou reserva de horários (ORDUÑA, 2013).

[5] https://nodejs.org/en/about/

2.5.1 RLMS

Um RLMS (Sistema de Gerenciamento de Laboratórios Remotos) pode ser definido como um conjunto de funcionalidades disponíveis em um laboratório que podem ser reutilizadas em outro, que tem como foco principal funcionalidades de gerenciamento, como por exemplo (ZUTIN, 2018):

- Gerenciamento de usuários.
- Implementação de um serviço de reserva.
- Suporte para escalabilidade de uma federação de laboratórios.
- Suporte para entrada comum em uma federação de laboratórios de diferentes instituições.
- Integração com Sistemas de gerenciamento de aprendizagem (LMS).

Os RLMSs contribuíram bastante para avanços no campo dos laboratórios remotos, abrindo possibilidades de compartilhamento de forma escalável experimentos entre diferentes instituições de ensino, através do agrupamento das funcionalidades (ZUTIN, 2018).

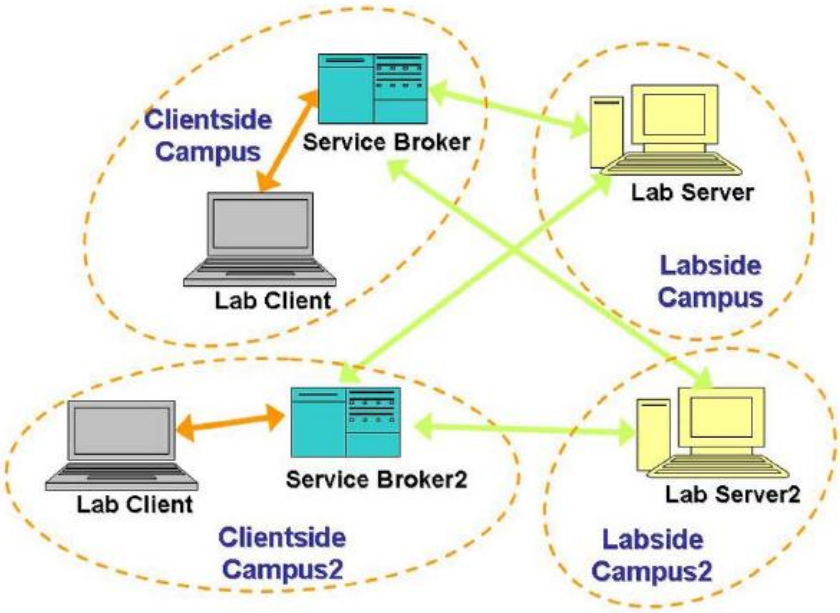
Os RLMSs otimizam o processo de desenvolvimento de laboratórios remotos, permitindo que professores, sem experiência com desenvolvimento, não tenham de perder tempo desenvolvendo a infraestrutura mínima para um laboratório remoto funcionar, focando apenas na disponibilização do experimento e nos alunos (ORDUÑA, 2013).

2.5.2 Arquitetura do iLab

O iLab teve origem em 1998, e era liderado por Jesús del Álamo, com o objetivo de permitir que seus alunos tivessem acessos à experimentos com dispositivos semicondutores, e ele foi um sucesso, inclusive conseguindo patrocínio da Microsoft, motivo pelo qual o projeto foi desenvolvido utilizando as ferramentas da empresa. Com esse sucesso outras universidades começaram a utilizar o iLab, hoje em dia universidades nos 5 continentes o utilizam (ORDUÑA, 2013).

Os experimentos disponibilizados pelo Lab, podem ser divididas em duas categorias, os *Batch Laboratories* (laboratórios gerenciados através de filas), e os *Interactive Laboratories* (laboratórios gerenciados através da reserva de horário).

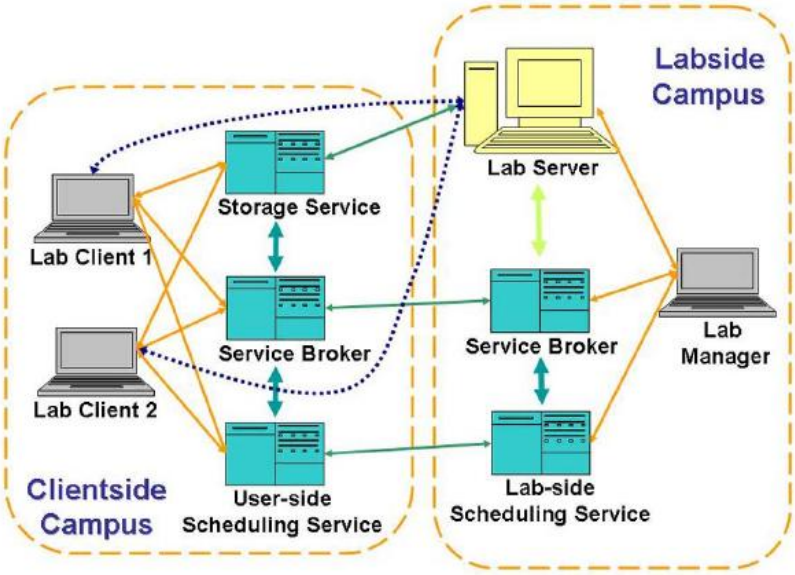
Figura 1 - Arquitetura dos Batch Laboratories do iLab



Fonte: Orduña (2013).

A arquitetura dos Batch Laboratories, como mostra a Figura 1, consiste do cliente, o Service Broker, que é responsável por validar o cliente, e o experimento solicitado, salvando as solicitações válidas do usuário num banco de dados, que no momento certo é processado pelo servidor do laboratório, que pode ou não retornar algo ao service broker, e por último o Servidor do Laboratório que é responsável por gerenciar as filas dos experimentos, todo esse processo de comunicação é feito utilizando SOAP (Protocolo Simples de Acesso a Objetos). A Figura 1 mostra como ocorre a comunicação de alunos de uma universidade com os experimentos do servidor do laboratório de outras universidades (ORDUÑA, 2013).

Figura 2 - Arquitetura dos *Interactive Laboratories* do iLab



Fonte: Orduña (2013).

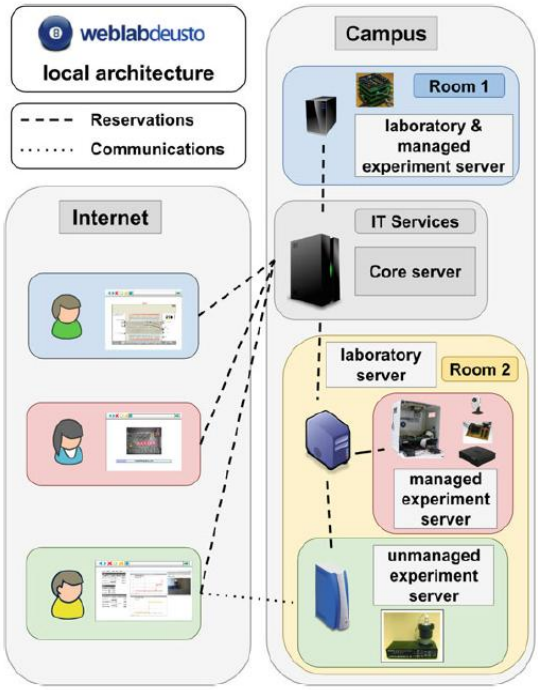
Diferentemente a arquitetura Interactive Laboratories busca uma forma de otimizar a comunicação entre cliente e servidor do laboratório, visto que a maioria dos experimentos interativos possuem um demorado tempo de conclusão de 20 minutos, fez-se necessário o uso de reservas. A Figura 2 demonstra de maneira mais sucinta a arquitetura, apesar do cliente se comunicar diretamente com o servidor do laboratório, ainda é preciso salvar antes a requisição tanto num banco de dados direcionado ao cliente, quanto num banco direcionado ao servidor do laboratório, de forma que se algum dos lados precisar efetuar uma operação em algum registro, ele possa checar com servidor do outro lado (ORDUÑA, 2013).

2.5.3 Arquitetura do Deusto

A arquitetura do Weblab-Deusto (Figura 3) é dividida em camadas, quando um usuário tenta logar no sistema, suas credenciais são enviadas para um servidor especificamente conFigurado para esse serviço, uma vez que esse servidor valida os dados, o servidor central é notificado com a solicitação de uma nova sessão. O servidor central é responsável por todas as regras de negócio, desde a checagem de

autenticação do usuário, até a interação com a maioria dos experimentos (ORDUÑA, 2013).

Figura 3 - Arquitetura do weblab-deusto



Fonte: Auer (2018).

O Deusto classifica os experimentos em duas categorias (ORDUÑA, 2013):

a. Experimentos Gerenciados: Esses experimentos foram desenvolvidos utilizando a API do Deusto, e todas interações com eles passam pelo servidor central.

b. Experimentos Não-Gerenciados: Esses experimentos, se comunicam diretamente com o usuário.

O Weblab-Deusto não utiliza agendamento de reservas, o gerenciamento de tempo para os usuários é feito utilizando filas de prioridade, os experimentos são desenhados de forma a serem breves para que a fila não congestione (ZUTIN et al., 2016).

As tecnologias utilizadas para a montagem dessa estrutura, foram o sistema de gerenciamento de banco de dados, MySQL, os clientes utilizam javascript para se comunicar, e os servidores de laboratórios rodam utilizando python (AKINWALE e KEHINDE, 2017).

2.5.4 LabShare Sahara

Desenvolvido pela Universidade Tecnológica de Sydney (UTS) a partir do consórcio Labshare, o Sahara é um sistema que permite o acesso a experimentos remotos (SIMÃO,2018).

Em suas versões iniciais ele consistia de um conjunto de ferramentas para auxiliar o desenvolvimento de laboratórios remotos, com o tempo ele se tornou um RLMS. A sua arquitetura é dividida em 3 componentes (ORDUÑA, 2013):

a. A interface web, desenvolvida em php, é responsável pelas interações básicas com o usuário.

b. Servidor de agendamento, desenvolvido em Java, ele gerencia a distribuição dos recursos do laboratório de acordo com os agendamentos.

c. O cliente do equipamento, é responsável por abstrair o aparato físico contidos nos experimentos.

O Sahara possui um dos mais complexos e avançados sistemas de agendamento para laboratórios remotos, ele permite o balanceamento de carga entre cópias de experimentos. Além disso ele possui tanto o agendamento através da fila, quanto o agendamento de horários (ORDUÑA, 2013).

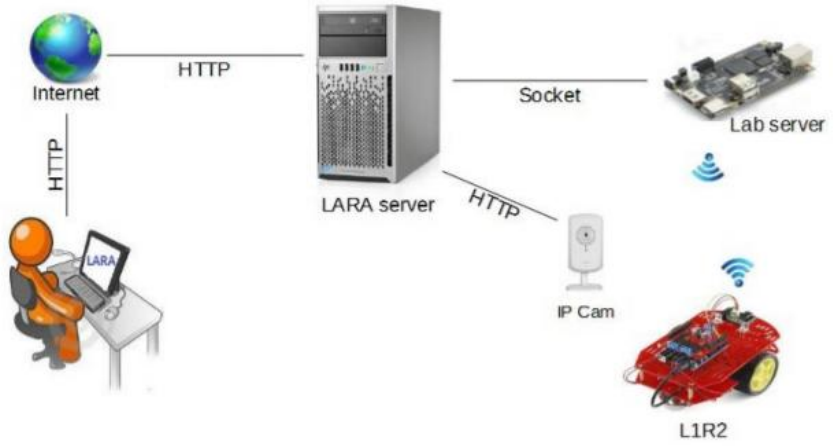
3. LARAaaS (LARA as a Service)

O LARA consiste num laboratório remoto que utiliza um AVA (Ambiente Virtual de Aprendizagem) para torná-lo colaborativo, atualmente apenas um experimento é disponibilizado pelo laboratório, esse experimento possibilita que o usuário controle um robô móvel via código (LOPES, 2017).

Para que a comunicação entre robô e cliente web aconteça, o código é passado para um servidor responsável por validar os dados e a reserva do cliente, e então o código é repassado para um segundo servidor responsável pela comunicação direta com o robô onde o código é compilado e o upload do mesmo é feito, essa arquitetura também é utilizada para que o cliente possa enviar e receber dados através da porta serial do robô (LOPES, 2017).

Toda essa comunicação é feita utilizando websocket que é um tipo de conexão que se mantém até que uma das partes queira finalizá-la (Figura 4).

Figura 4 - Arquitetura LARA



Fonte: Lopes (2017).

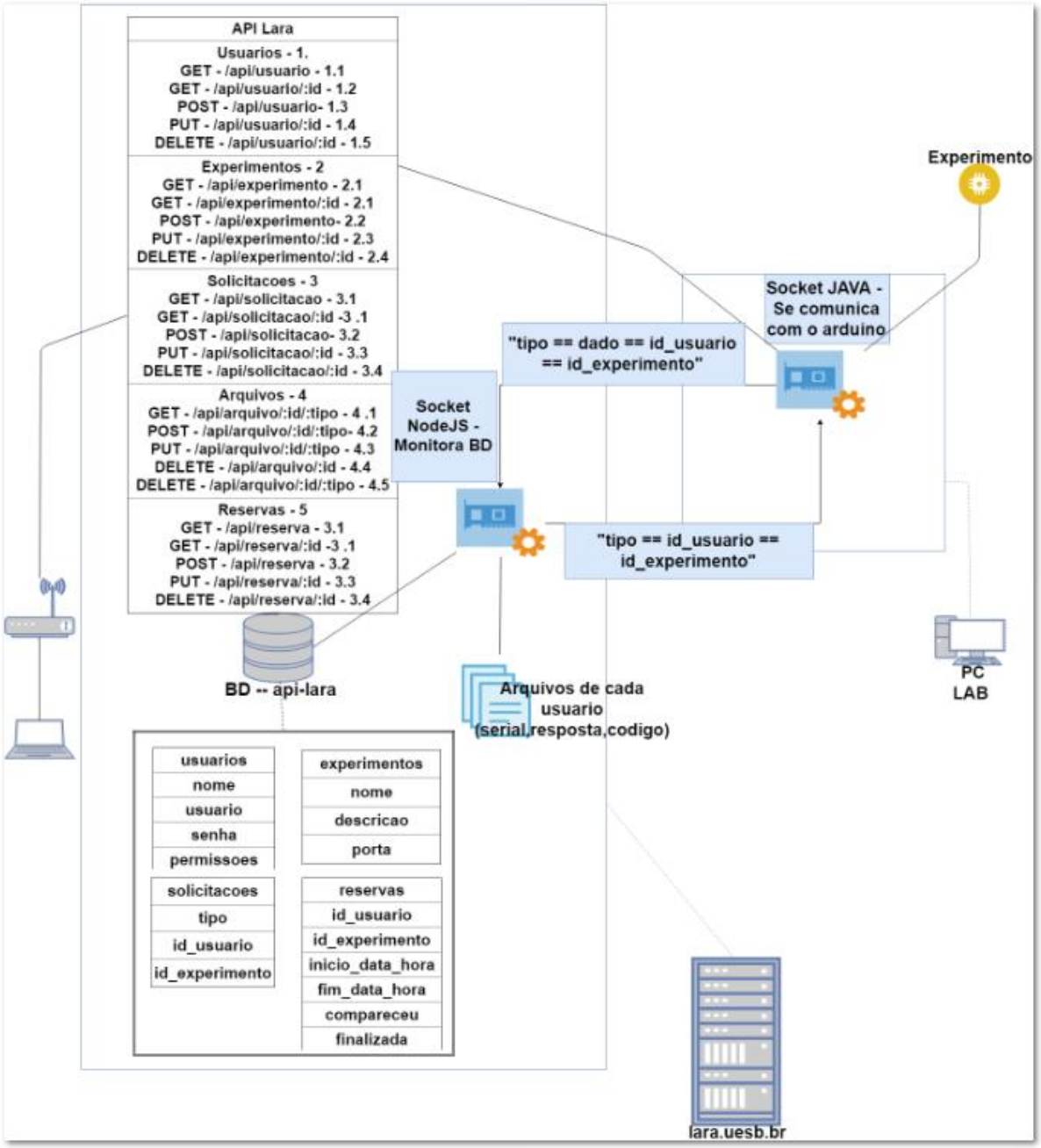
A arquitetura atual do LARA (Figura 4), possui algumas limitações que debilitam a escalabilidade do laboratório, durante o desenvolvimento o código back-end ficou muito atrelado à infraestrutura do LMS utilizado, o Moodle. Além disso, devido ao foco inicial em um único experimento, não foi feito um planejamento arquitetural que desse suporte ao acoplamento de novos experimentos à

infraestrutura do projeto, toda a arquitetura é dependente do Moodle, e das comunicações específicas utilizadas pelo experimento atual, o L1R2.

3.1 Arquitetura proposta

A nova arquitetura proposta (Figura 5) tenta corrigir os pontos citados anteriormente, com uma arquitetura baseada num webservice seguindo o padrão de arquitetura REST, essa arquitetura tem como principal objetivo melhorar a escalabilidade do LARA, permitindo agregar novas funcionalidades, trocas de componentes da arquitetura, a utilização da infraestrutura por aplicações de terceiros, além disso a criação de uma API abre a possibilidade da criação de novas instâncias do LARA em outras instituições com maior facilidade, o LARA se torna um serviço, o LARAaaS (LARA as a Service).

Figura 5 - Nova arquitetura proposta para o LARA



Fonte: Autoria própria

O primeiro passo tomado para o desenvolvimento da API foi a análise de quais dados seriam expostos na mesma, a lógica de autenticação e autorização de usuários foram os primeiros requisitos indicados, de forma a desenvencilhar o LARA do sistema de autenticação do moodle, permitindo a troca de LMS se necessário. Além de mais segurança ao ter acesso direto ao sistema responsável pelo controle

de seus usuários, esse é um dos pontos centrais na grande maioria dos laboratórios, que permite a eles uma maior mobilidade entre instituições, no caso do nosso laboratório, uma instituição tanto poderá diretamente utilizar a infraestrutura online do LARA, quanto instalar uma versão local do mesmo.

O novo sistema de autenticação não é somente uma forma de ter acesso aos laboratórios do LARA, e sim à API como um todo, abrangendo desde alunos dos cursos da instituição, à clientes de fora que necessitam apenas dos dados disponibilizados pela API.

Após definir esses pontos primários, a parte de experimentos veio em seguida, atualmente a plataforma é limitada ao L1R2(experimento de robótica móvel), esse ponto debilita a escalabilidade do laboratório, de forma que a cada novo experimento um ajuste em toda a estrutura do sistema se faz necessário. Com a interface proposta para o laboratório, os experimentos se tornam recursos da API, e podem ser atualizados, criados e deletados (Figura 5) através dessa interface comum a todos os experimentos.

Uma melhora que não tem relação direta com o novo padrão arquitetural, e sim com uma mudança na forma de utilizar uma das tecnologias centrais do laboratório, a compilação de código arduino, que era feita no servidor interno do laboratório. Na nova versão o código é compilado no mesmo servidor em que webservice estará localizado, evitando sobrecarga e erros durante uma comunicação desnecessária, essa mudança foi possível devido a disponibilização da interface de linha de comando por parte do arduino (apenas na distribuição linux), o arduino exigia a interface gráfica durante a compilação previamente quando essa etapa do laboratório foi desenvolvida.

Também foi implementado o sistema de gerenciamento de reservas, após pesquisar sobre vários outros exemplos de laboratórios fez-se perceptível a importância dessa funcionalidade, inclusive alguns dos laboratórios mais destacados até pouco tempo não possuíam a mesma. O LARA já possuía um sistema de reserva, foi feito, então, uma adaptação para o novo padrão de forma a facilitar modificações e adaptações futuras.

Umas das principais barreiras para qualquer laboratório remoto é a comunicação com os componentes dos diferentes experimentos (ORDUÑA, 2013), por possuírem componentes bastante heterogêneos em suas características, eles

dificultam a padronização na forma de comunicação. O L1R2, em alguns momentos, requer comunicação em tempo real, por exemplo, o que foge dos requisitos do padrão REST. Na implementação anterior, é feita uma comunicação serial utilizando sockets diretamente após receber uma solicitação, um canal é aberto entre servidor local do laboratório e servidor interno da UESB, onde o LARA está hospedado, e só é fechado quando uma das partes solicita o fechamento. Essa forma de comunicação continua sendo usada na nova implementação, mas de forma a isolá-la da API a fim de manter os princípios do padrão REST.

3.2 Implementação

Para o desenvolvimento da API, foram utilizadas diversas tecnologias todas elas bastante utilizadas pelo mercado, como o Sistema de Gerenciamento de Banco de Dados, MySql. Este SGBD possui uma biblioteca de integração (Figura 6) com nodejs com diversas funcionalidades, além de ser muito bem otimizada.

Figura 6 - Exemplo comando sql utilizando a biblioteca mysql do nodeJS

```
this.conexao.query('SELECT * FROM usuario where id=?', [id], (err, results, fields) => {
    if (err) {
        return reject(err.code);
    }

    return resolve(results);
});
```

Fonte: Autoria própria.

Outra tecnologia utilizada é o framework express, segundo o site oficial[6], ela é uma framework minimalista e flexível para nodeJS que provêm um conjunto robusto funcionalidades para aplicações web e mobile, utilizando o express é possível colocar um servidor javascript em execução rapidamente, facilitando a definição de rotas, acoplação de middlewares, e organização do projeto (Figura 7 e 8).

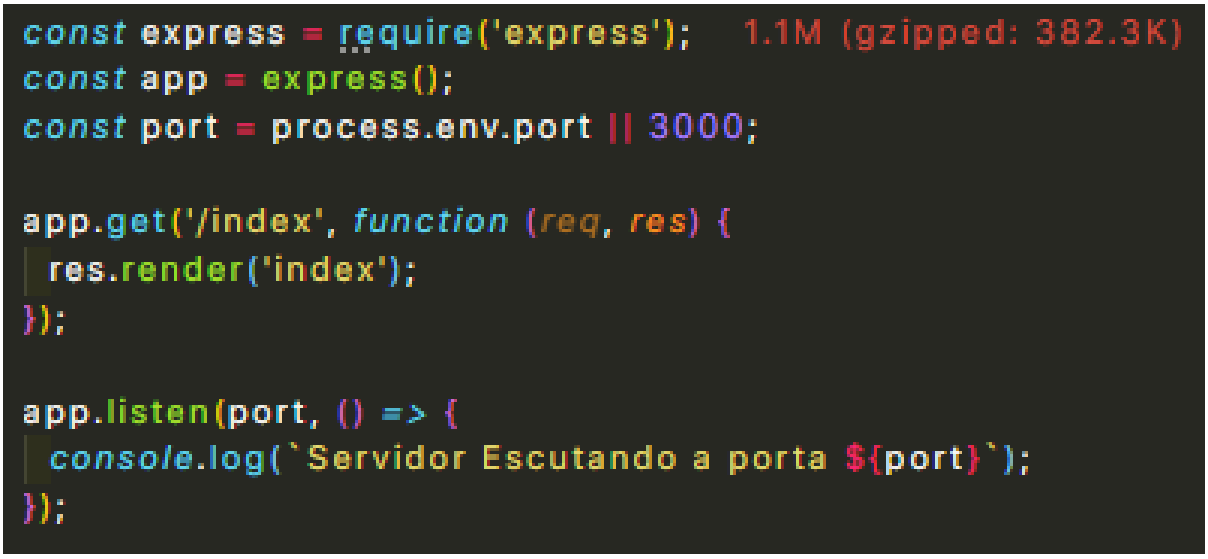
[6] https://expressjs.com/

Figura 7 - Exemplo de código node sem utilização do express

```
const app = require('http');
const fs = require('fs');
const port = process.env.port || 3000;

app.createServer(function (req, res) {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  var url = req.url;
  if (url === '/index') {

    fs.readFile('index.html', function (err, data) {
      res.writeHead(200, { 'Content-Type': 'text/html', 'Content-Length': data.length });
      res.write(data);
      res.end();
    });

  }
}).listen(port, function () {
  console.log(`Servidor Escutando a porta ${port}`);
});
```

Fonte: Autoria própria

Figura 8 - Exemplo de código utilizando o express



```
const express = require('express');   1.1M (gzipped: 382.3K)
const app = express();
const port = process.env.port || 3000;

app.get('/index', function (req, res) {
  res.render('index');
});

app.listen(port, () => {
  console.log(`Servidor Escutando a porta ${port}`);
});
```

Fonte: Autoria própria

A estrutura do projeto seguiu o padrão MVC (Figura 9), onde a aplicação é dividida em 3 partes: Model, View e Controller. O Model é a parte em que os dados são tratados e as regras de negócio definidas. A View é a tela de apresentação da

aplicação por onde o usuário interage. O Controller é responsável pelo controle do fluxo de dados entre a View e o Model.

Devido ao LARA utilizar um AVA (Ambiente Virtual de Aprendizagem) como responsável pela interação dos usuários, a aplicação REST em si não possui a responsabilidade de renderizar a View, de forma que ela é composta apenas de Models e Controller. Nem o nodejs e nem o express definem um padrão de organização da aplicação, no caso deste trabalho, foi escolhido criar um único controller responsável pelo roteamento da aplicação e diversos models cada um representando uma tabela do banco de dados, neles estão implementadas as funções responsáveis por se comunicar com o banco de dados.

Figura 9 - Estrutura da Aplicação

Fonte: Autoria própria

Assim que a estrutura foi definida, foi iniciado o processo de definição das rotas ou endpoints, para os *models* usuário, experimento e solicitação seguindo o padrão REST e utilizando os métodos HTTP (Figura 10), com o objetivo de obter recursos, criar recursos, atualizar e/ou deletá-los.

Um dos principais obstáculos durante a implementação da API do LARA foi lidar com os requisitos exigidos pelos experimentos. Muitas vezes a forma de se comunicar com seus componentes forçam a arquitetura a não seguir os princípios do padrão REST, como por exemplo a comunicação serial do Arduino, placa de prototipagem que serve de base para o experimento atual do LARA.

Existem casos que se exige que ela retorne dados de leituras a todo momento, o que demanda que a infraestrutura do LARA também retorne esses dados ao cliente de forma contínua até que a leitura seja finalizada. Essa conexão contínua quebra uma das restrições do REST, que exige uma conexão Cliente-Servidor sem estado, a requisição enviada pelo cliente deve ser completa e não deve esperar qualquer metadado do estado da comunicação por parte do servidor, o mesmo deve retornar apenas os dados, baseando-se no que o cliente enviou, sem guardar o estado.

Figura 10 - Rotas do model usuário

```
app.post('/api/usuario', verificar_token, function (req, res) {
    usuario_model
        .registrar(req.body)
        .then(results => {
            res.send(results);
        })
        .catch(err => {
            console.log("-" + err);
        });
});

app.get('/api/usuario/:id', verificar_token, function (req, res) {
    usuario_model
        .buscar_pelo_id(req.params.id)
        .then(results => {
            res.send(results);
        })
        .catch(err => {
            console.log(err);
        });
});

app.put('/api/usuario/:id', verificar_token, function (req, res) {
    usuario_model
        .atualizar(req.params, req.body)
        .then(results => {
            res.send(results);
        })
        .catch(err => {
            console.log(err);
        });
});

app.delete('/api/usuario/:id', verificar_token, function (req, res) {
    usuario_model
        .deletar(req.params)
        .then(results => {
            res.send(results);
        })
        .catch(err => {
            console.log(err);
        });
});
```

Fonte: Autoria própria

Em vários outros laboratórios remotos é possível observar essa dificuldade em lidar com os diferentes tipos de entradas e saídas que os experimentos exigem. Orduña (2012), por exemplo descreve o caso do iLabs, que utiliza a funcionalidade *Service Broker*[7] contido no *SQL Server*[8] para conseguir tratar comunicações solicitadas pelo cliente que um webservice não consegue tratar, salvando a solicitação no banco através do padrão SOAP que ele implementa, e monitorando o

[7] SQL Server Service Broker provêm suporte para formação de filas de aplicações e troca de mensagens ao SQL SERVER (RAY, 2018).
[8] É o SGDB que faz parte da plataforma da microsoft (GUYER, 2018).

mesmo para que a solicitação seja respondida futuramente por um outro processo interno do servidor do laboratório capaz de atendê-la.

Com base, na solução do iLabs, foi feita então uma alteração no planejamento da arquitetura, o usuário ao enviar algum código para o experimento, ele não irá diretamente para o experimento, o código, o identificador do usuário e o identificador dos experimentos são salvos numa tabela de solicitação, de forma a preservar as restrições do REST (Figura 5).

A responsabilidade de processar as solicitações é passada para um servidor também baseado na plataforma nodeJS que utiliza o pacote ZongJi, ele é responsável por monitorar a tabela de solicitações, assim que uma nova solicitação é adicionada, os dados são enviados utilizando o pacote nativo *net* do nodeJS que cria uma conexão socket entre o servidor local do experimento, que possui a tarefa de compilar e enviar o código ao arduino.

O servidor local e o servidor nodeJS de monitoramento, por estarem conectados via socket, possuem a capacidade de envio em tempo real, já que a conexão se mantém ativa até que um dos nós da conexão a feche, dessa forma assim que o arduino gere alguma saída, o servidor local envia os dados para o servidor de monitoramento e a resposta serial será recebida e salva num arquivo.

Estes arquivos são disponibilizados pela API na forma de streams de dados, onde o cliente tem a liberdade de escolher como melhor utilizá-los, acessando como qualquer outro recurso do webservice.

Figura 11 - Servidor que monitora a tabela de Solicitações

```
zongji.start({
    includeEvents: ['tablemap', 'writerows'],
    includeSchema: { lara_rest: ['solicitacao'] },
    startAtEnd: true
});
```

Fonte: Autoria própria

Streams são abstrações que o nodejs possui capazes de lidar com a escrita e leitura de arquivos, diferente das formas comuns de leitura e escrita, nelas os

arquivos não são carregados inteiramente, eles são carregados pedaço a pedaço evitando uso desnecessário de memória (NODE.JS FOUNDATION, 2018).

A autenticação do usuário é feita utilizando um método bastante utilizado, chamado *JSON web tokens,* esse método consiste em gerar um token baseado no login e senha do usuário, de forma que para utilizar a api seja necessário o uso desse token como autenticador em toda requisição, esse token tem um tempo de expiração, essa forma de autenticação evita que o cliente tenha que armazenar a senha e o login sempre que necessitar fazer uma requisição.

O experimento do LARA utiliza o modelo de laboratório interativo, (ORDUÑA,2013) que se baseia num sistema de reserva, nele os usuários devem efetuar a reserva de um horário previamente para ter acesso, quando o usuário acessa, ele inicia uma sessão, e tem acesso ao experimento até o fim da sessão.

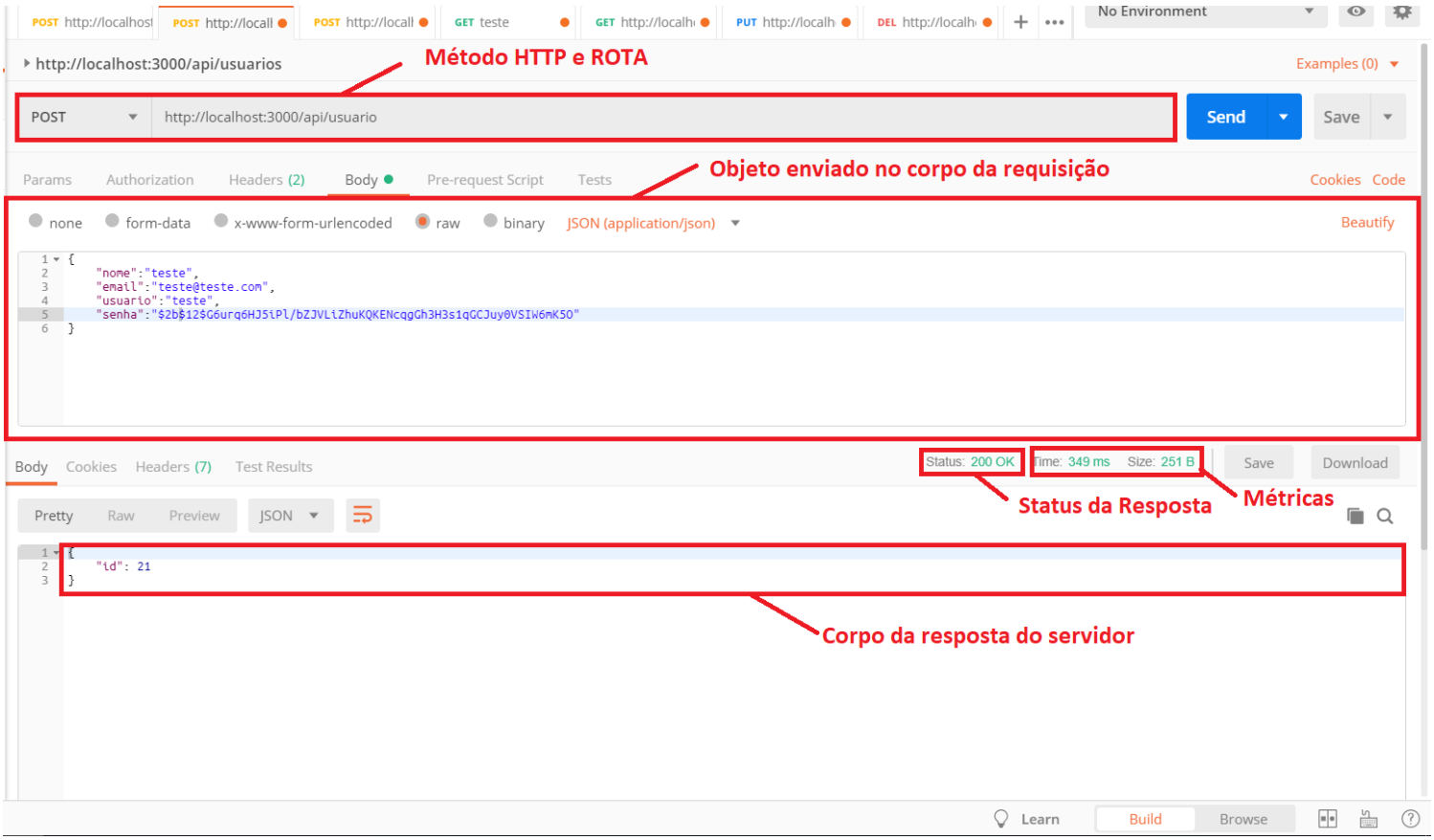
4. VALIDAÇÃO DO LARAaaS

Ao longo do desenvolvimento do webservice do Lara foram utilizadas várias ferramentas de teste, o *POSTMAN*, aplicação capaz de fazer requisições http de forma rápida, o *mocha*, pacote javascript, composto de funções que permitem a criação de um ambiente de testes com grande facilidade, o *chai*, outro pacote javascript, responsável pelas requisições http e checagem da validade dos dados retornados na requisição, e o *morgan*, um middleware com a função de criar logs com diversas informações (tempo de resposta, bytes retornados pela requisição, status, endpoint) a cada requisição feita.

O *POSTMAN* foi a ferramenta mais utilizada durante o desenvolvimento por permitir teste de maneira rápida e de fácil conFiguração. Foi possível testar os endpoints criados sem a necessidade de preparação de um script de teste, ou de uma camada de aplicação que requisite o webservice.

Na Figura 12, é possível observar como funcionam os testes utilizando o *POSTMAN*, primeiro são definidos o método HTTP e a rota que será testada. Neste caso está sendo testada uma rota do model Usuário com o método POST. São passados alguns dados do usuário à ser inserido, neste trabalho foi escolhido utilizar o JSON (JavaScript Object Notation) como formato padrão das requisições do webservice, dessa forma é passado um objeto JSON com os dados do usuário a ser inserido no corpo da requisição, e como resposta obteve-se o status 200, que remete à uma resposta bem-sucedida, além disso no corpo da resposta tem o Id do usuário adicionado.

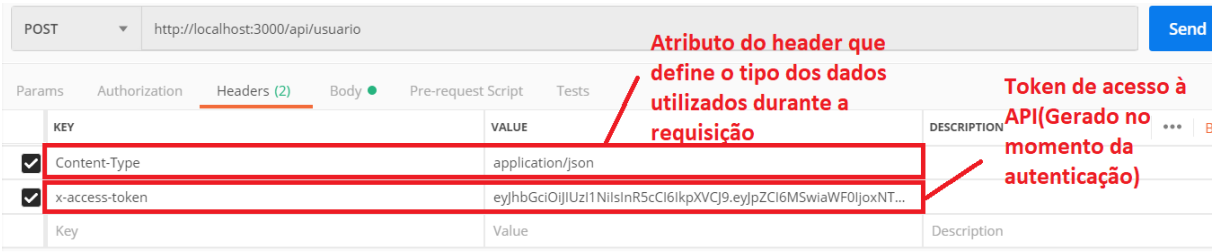
Figura 12 - POSTMAN, exemplo de inserção de usuário.



Fonte: Autoria própria

A Figura 13 demonstra como são adicionados atributos no header da requisição, a Figura 12, por exemplo, foi necessário adicionar 2 atributos, o *Content-Type* e o *x-access-token*, um é responsável por definir o formato dos dados enviados na requisição e como eles são esperados na resposta, no caso como informado previamente, o formato definido foi o JSON, o segundo atributo é o token de acesso, necessário para ter autorização de acesso às rotas da API.

Figura 13 - POSTMAN, exemplo de definição do header.



Fonte: Autoria própria

No caso de não envio do token, como mostra a Figura 14, o servidor retorna um json com uma flag indicando que a autorização foi rejeitada e uma mensagem descrevendo o motivo, com o status 403 indicando que o servidor recebeu a requisição e a recusou por falha na autorização, o content-type não seria obrigatório pelo formato JSON utilizado, ser o padrão da plataforma nodeJS, de forma que ele sempre assume que está recebendo um JSON.

Figura 14 - POSTMAN, exemplo de falha na autenticação

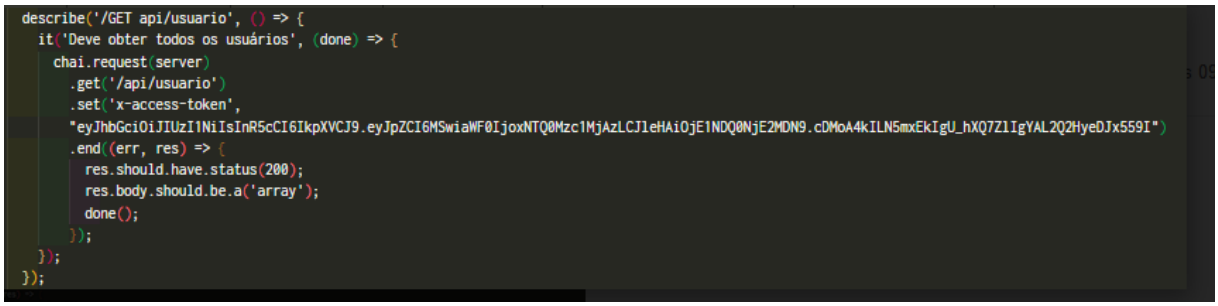
Fonte: Próprio autor.

Moncha e *chai* foram utilizados mais ao final do desenvolvimento de forma a validar os endpoints da API criada, diferentemente do *POSTMAN* em que os testes são feitos de forma mais empírica, utilizando as ferramentas citadas é possível definir condições de forma programática e testar cada funcionalidade dando maior

segurança na entrega da aplicação e que grande parte das possibilidades de erro foram conferidas.

O processo é bem parecido com o seguido nos testes feitos no *POSTMAN*, o método HTTP e a rota são informados na chamada da função *describe*, na função *it* deve se passar o objetivo do teste como primeiro argumento, esses primeiros dados servem apenas para melhor compreensão do resultado na hora que o script de teste for executado, como segundo argumento passamos a função responsável pela parte lógica do teste. Utilizando o chai é feita uma requisição, o nome da chamada varia de acordo com o método HTTP utilizado, e tem como argumento a rota a ser testada, a função *set* é usada para configurar os atributos do header da requisição, é com ela que definimos o token, e ao fim a função *end*, recebe o objeto de retorno da requisição e com ele são feitas validações.

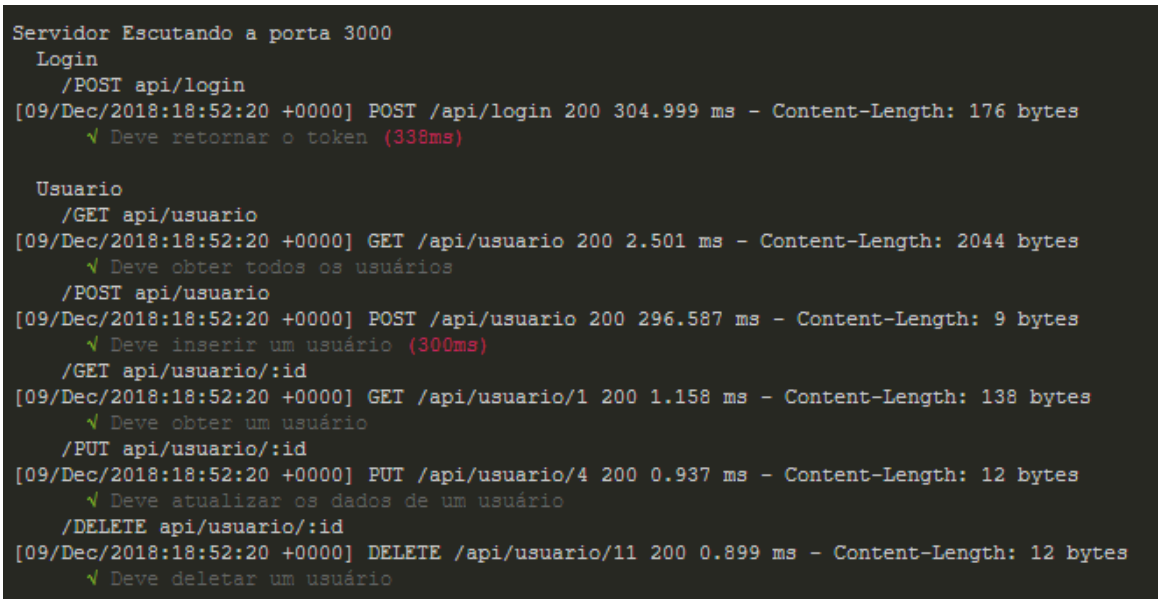
Figura 15 - Exemplo de código de um teste feito em um endpoint



Fonte: Autoria própria

Na Figura 15, é possível ver como as validações são feitas, o chai utiliza um vocabulário bem verboso, o trecho *res.should.have.status(200)*, por exemplo pode ser traduzido como a “resposta deve ter o status 200”, nesse caso são testados o status e o tipo do retorno que deve ser um array.

Figura 16 - Saída dos testes feitos utilizando os pacotes mocha e chai



Fonte: Autoria própria

A saída dos testes é exemplificada na Figura 16, o método e a rota seguida do resultado abaixo com o objetivo ao lado.

Esses testes tiveram como objetivo demonstrar o funcionamento efetivo das rotas do webservice, validando as entradas e saídas das rotas, foram escritos scripts de teste para cada model do webservice, testando cada rota existente.

Além disso foi feita a medição do tempo de resposta de forma a detectar algum erro lógico que estivesse acarretando em perda de performance, as medições não indicaram nenhuma anormalidade.

5. CONCLUSÃO

Ao longo deste trabalho foi possível analisar o panorama atual no universo dos laboratórios remotos, o que foi importante para entender qual o atual patamar do LARA em relação aos outros laboratórios ao redor do mundo com muito mais tempo em ação e tendo possivelmente passado por experiências que o LARA ainda não passou.

Um dos dados encontrados durante a pesquisa foi que a grande maioria dos laboratórios remotos já fizeram a transição para a arquitetura orientada a serviços, o que torna imprescindível que o LARA faça o mesmo.

Como visto anteriormente, um dos motivos para essa mudança por parte dos laboratórios é a possibilidade de acoplar um LMS (Sistema de Gerenciamento de Aprendizagem) de forma que o laboratório não fique preso ao mesmo, outro motivo é facilitar a utilização dos laboratórios por parte de outras instituições, disseminando-os como serviços.

Ambos os pontos anteriores são questões que faltavam a arquitetura do LARA, além da questão da escalabilidade, que é uma grande barreira para o crescimento do laboratório, devido a dificuldade para a inserção de um novo experimento na infraestrutura do projeto.

A atual arquitetura do laboratório além de pouco escalável, por não seguir um padrão de comunicação, impede diagnósticos precisos de gargalos de performance e falhas de segurança, aumenta o custo de manutenção do sistema e impossibilita a portabilidade do LARA para novas plataformas sem refatoração de código, tornando indispensável uma padronização em sua arquitetura.

Durante esse trabalho foi criada a arquitetura e realizada a implementação de um webservice que soluciona esses problemas, transformando os componentes do LARA em recursos e o LARA em um serviço que pode ser utilizado por qualquer um que tenha autorização para tal, fornecendo o básico que um laboratório remoto necessita, como Gerenciamento de usuários, de reservas e de experimentos. A API também permite serviços personalizados do arduino, como compilar código e trocar mensagens via serial.

A API foi validada através de testes de unidade que demonstraram que todas as rotas estão funcionando perfeitamente e retornando os dados esperados. A utilização dessa nova arquitetura deve facilitar e incentivar novos projetos utilizando

o LARA como infraestrutura base, fomentando cada vez mais o seu crescimento, sendo o primeiro passo para a evolução do LARA para um RMLS, que deve ser o próximo objetivo do projeto, de forma a preencher um espaço vazio no ensino à distância em nossa região.

5.1 Contribuições

As principais contribuições deste trabalho são:

- Desenvolvimento da arquitetura orientada a serviço para o LARA, o conceito do LARAaaS, na qual o laboratório se torna um serviço, permitindo a adição de novas funcionalidades, comunicação com outras aplicações portabilidade para novas plataformas.
- Adaptação da comunicação em tempo real exigida pelo experimento do LARA a arquitetura REST, essa solução não só viabilizou a utilização da arquitetura proposta como também flexibiliza a entrada de novos experimentos à infraestrutura do laboratório.

5.2 Trabalhos Futuros

Como sugestão de trabalhos futuros sugere-se:

1. Melhora no processamento das solicitações em tempo real, utilizando a biblioteca RabbitMQ, que consiste num software que permite a troca de mensagens entre aplicações, ela dá acesso a diversos protocolos de comunicação, enfileiramento de tarefas, permitindo assim um melhor tratamento das solicitações em tempo real que fogem aos padrões da API REST (PIVOTAL SOFTWARE).
2. Criação de uma rota que utilize GraphQL, uma tecnologia que permite a criação de interfaces de comunicação com muito mais expressividade que APIs REST, sem perda de performance, e utilizando menor número de requisições para conseguir os dados que a aplicação necessita (GRAPHQL FOUNDATION).
3. Complementação do webservice criado neste trabalho com um front-end, de forma a implementar um RLMS (Sistema de Gerenciamento de Laboratórios

Remotos) completo para o LARA, se tornar um RLMS é o próximo passo do LARA, de forma que seu crescimento não seja reprimido, de forma a permitir múltiplos laboratórios de diferentes instituições.

6. REFERÊNCIAS BIBLIOGRÁFICAS

AKINWALE, Olawale B. e KEHINDE, Lawrence O. **An Update to the iLab Shared Architecture for Mobile Device Support**. International Journal of Online and Biomedical Engineering (iJOE), p. 87-101, 2017.

BROWN, Ethan. **Web development with Node and Express: leveraging the JavaScript Stack**. [s.l.]: O'Reilly, 2014.

BROWN, Paul. **State of the Union: npm**. Linux.com. Disponível em: <https://www.linux.com/news/event/Nodejs/2016/state-union-npm/>, acesso em: 10 out. 2018.

COELHO, Paulo Rodolfo da Silva Leite. **Uma arquitetura orientada a serviços para laboratórios de acesso remoto**. 2006. 103p. Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação, Campinas, SP. Disponível em: <http://www.repositorio.unicamp.br/handle/REPOSIP/258974>. Acesso em: 7 ago. 2018.

CASCIARO, Mario e MAMMINO, Luciano. **Node.js design patterns: get the best out of Node.js by mastering its most powerful components and patterns to create modular and scalable applications with ease**. [s.l.]: Packt Publishing open course, p. 1423-1432, 2016.

FIELDING, Roy T. **Architectural styles and the design of network-based software architectures**. [s.l.: s.n.], 2000.

GUYER, Craig et al. **SQL Server Documentation**. Microsoft Docs. Disponível em: <https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation?view=sql-server-2017>, acesso em: 10 nov. 2018.

HEVNER, Alan and CHATTERJEE, Samir. **Design Science Research in Information Systems**. Integrated Series in Information Systems Design Research in Information Systems, p. 9–22, 2010.

LOPES, Maísa Soares dos Santos. **Ambiente colaborativo para ensino aprendizagem de programação integrando laboratório remoto de robótica**. Salvador, 2017. 105p. Dissertação de Doutorado, Programa de Pós-Graduação em Engenharia Industrial, Universidade Federal da Bahia.

MULLIGAN, Gavin e GRACANIN, Denis. **A Comparison Of Soap And Rest Implementations Of A Service Based Interaction Independence Middleware Framework**, Proceedings of the 2009 Winter Simulation Conference (WSC), 2009.

NGOLO, Márcio A. F. **Arquitectura Orientada a Serviços REST para Laboratórios Remotos**. Lisboa, 2009. 118p. Dissertação de Mestrado, Departamento de

Engenharia Electrotécnica Da Faculdade de Ciências de Tecnologia, Universidade Nova de Lisboa.

NODE.JS FOUNDATION. **About Node.js®**. nodejs.org. Disponível em: <https://nodejs.org/en/about/>, acesso em: 10 out. 2018.

NODE.JS FOUNDATION. **The Node.js Event Loop**, Timers, and process.nextTick(). nodejs.org. Disponível em: <https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/>, Acesso em: 10 out. 2018.

GRAPHQL FOUNDATION. **Introduction to GraphQL**. graphql.org. Disponível em: <https://graphql.org/learn/>, Acesso em: 10 out. 2018.

PIVOTAL SOFTWARE. **What can RabbitMQ do for you?**. Pivotal Software. Disponível em: <https://www.rabbitmq.com/features.html>, Acesso em: 10 out. 2018.

ORDUÑA, Pablo. **Transitive And Scalable Federation Model For Remote Laboratories**. Bilbao, 2013. 215p. Dissertação de Doutorado, Programa de Doutorado em Sistemas de Informação, Universidade de Deusto.

RAY, Mike. **SQL Server Service Broker**. Microsoft Docs. Disponível em: <https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/sql-server-service-broker?view=sql-server-2017>, acesso em: 10 nov. 2018.

SIMÃO, José P. S. **MODELO PARA REGISTRO DE DADOS DE EXPERIÊNCIA DE APRENDIZAGEM EM LABORATÓRIOS REMOTOS**. Araranguá, 2018. 115p. Dissertação de Mestrado, Programa de Pós-Graduação em Tecnologias da Informação e Comunicação, Universidade Federal de Santa Catarina.

SCHOTT, Fred K. **The Node Way**. Introduction: What is The Node Way? Disponível em: <http://thenodeway.io/>.

SOUZA, Victor Alexandre S. M. de. **Uma arquitetura orientada a serviços para desenvolvimento, gerenciamento e instalação de serviços de rede**. 2006. 82p. Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Eletrica e de Computação, Campinas, SP. Disponível em: <http://www.repositorio.unicamp.br/handle/REPOSIP/261749>. Acesso em: 9 set. 2018.

TAWFIK, Mohamed et al. **Shareable Educational Architectures for Remote Laboratories**, 2012 Technologies Applied to Electronics Teaching (TAEE), 2012.

TANENBAUM, Andrew S. **Redes de computadores**. Pearson Educación, 2003.

WILDERMUTH, Shawn. **REST matters (and you need more of it)**. Pluralsight. Disponível em: <https://www.pluralsight.com/blog/tutorials/representational-state-transfer-tips>, acesso em: 10 agosto. 2018.

ZAZA, Samuele. **Test a Node RESTful API with Mocha and Chai**. Scotch.io, 2017. Disponível em: <https://scotch.io/tutorials/test-a-node-restful-api-with-mocha-and-chai>. Acesso em: 9 dez. 2018.

ZUTIN, Danilo G. Online Laboratory Architectures and Technical Considerations. In: AUER, Michael E.; AZAD, Abul K.M.; EDWARDS, Arthur; JONG, Ton de. **Cyber-Physical Laboratories in Engineering and Science Education**. Springer International Publishing, 2018.cap.1, p.5-14.

ZUTIN, Danilo et al. **Online Lab Infrastructure as a Service: A new Paradigm to Simplify the Development and Deployment of Online Labs**. 2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV), p. 208-214, 2016.