Larissa Alves
1903larissaalves@gmail.com
(54) 9 9949-7433

Silmara Leite
silmara.leitec@gmail.com
(47) 9 9144-0912

Nathália Locatelli
nathalia.lcz@hotmail.com
(47) 9 9256-0565

Vanessa Lage
vanessa.vidallage@gmail.com
(47) 9 8814-5817

Paula Campigotto
paula_campigotto@hotmail.com
(47) 9 9998-2083

LadyBug

Central de erros

Introdução

Em projetos modernos é cada vez mais comum o uso de arquiteturas baseadas em serviços ou microsserviços. Nestes ambientes complexos, erros podem surgir em diferentes camadas da aplicação (backend, frontend, mobile, desktop) e mesmo em serviços distintos. Desta forma, é muito importante que os desenvolvedores possam centralizar todos os registros de erros em um local, de onde podem monitorar e tomar decisões mais acertadas. Neste projeto vamos implementar um sistema para centralizar registros de erros de aplicações.

Objetivos

1. **Backend:** criar endpoints para serem usados pelo frontend da aplicação, criar um endpoint que será usado para gravar os logs de erro em um banco de dados relacional. A API deve ser segura, permitindo acesso apenas com um token de autenticação válido.

2. **Frontend:** deve implementar as funcionalidades apresentadas nos wireframes, ser acessada adequadamente tanto por navegadores desktop quanto mobile, consumir a API do produto e ser desenvolvida na forma de uma Single Page Application.

Especificações

1. Requisitos

As funcionalidades do sistema *LadyBug* podem ser visualizadas na Figura 1, que representa um diagrama de caso de uso, cujos casos são especificados abaixo.

- **Cadastrar usuário:** o cadastro de usuário pode ser realizado por qualquer agente, quando fornecidos o nome, nome de usuário de no mínimo três caracteres, e-mail válido, e senha de no mínimo três caracteres.
- **Efetuar login:** para realizar o login é necessário fornecer o nome de usuário e a senha cadastrados. O acesso é autorizado a partir da autenticação com JWT (*JSON Web Token*).
- **Cadastrar erro:** os erros são cadastros na API a partir de requisições HTTP (*Hypertext Transfer Protocol*) do tipo *POST*, por outras aplicações. Para isso, tais aplicações precisam enviar os dados encapsulados em JSON (*JavaScript Object Notation)*, contendo as seguintes informações: título, detalhes, ambiente, origem, nível e ID do usuário.

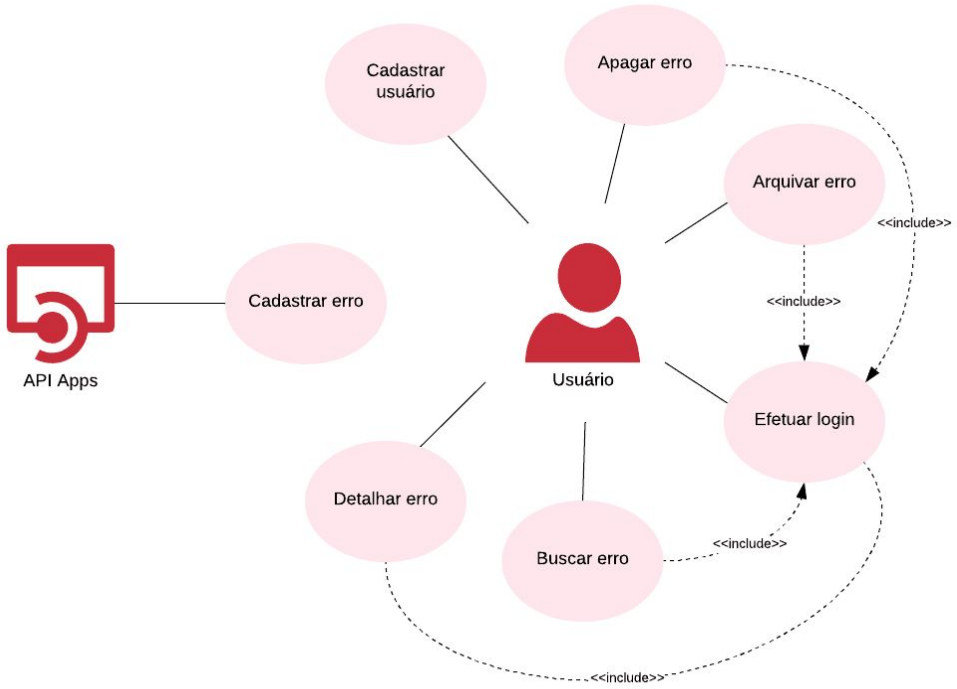
As funcionalidades a seguir só podem ser realizadas após a efetuação do login, pois elas são acessadas a partir de requisições HTTP do tipo *GET* que necessitam de um *header* (cabeçalho) contendo o JWT gerado no momento do login.

- **Buscar erro:** os logs de erro salvos na API da *LadyBug* são apresentados na tela inicial do sistema, após o login, e podem ser buscados de maneiras diferentes, a

partir dos filtros de: ambiente, nível, descrição e origem. Os logs de erros arquivados também podem ser buscados, com uma requisição diferente. Ademais, a busca pode ser ordenada pelo nível ou pela frequência (quantidade de ocorrências) dos erros.

- **Detalhar erro:** para visualizar os detalhes dos erros é feita uma requisição com o identificador ID do erro como parâmetro, assim, serão retornados todos os seus atributos, os quais foram enviados no momento do cadastro.
- **Arquivar erro:** os erros podem ser arquivados, dessa forma, o atributo do tipo *boolean* correspondente será alterado para verdadeiro, e ao detalhar o erro será possível identificar se ele está arquivado ou não.
- **Apagar erro:** essa funcionalidade realiza uma requisição HTTP do tipo *DELETE* e remove o erro do banco de dados da *LadyBug*. Tal ação, quando realizada, é irreversível e o identificador do erro apagado não será mais utilizado.

Figura 1: Diagrama de caso de uso



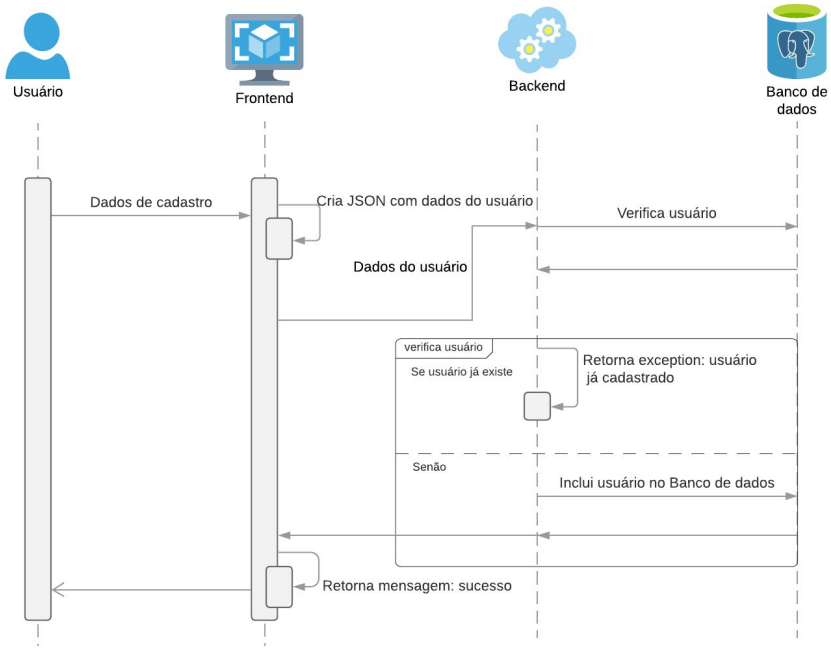
Fonte: Elaborado pelas autoras (2020)

2. Diagrama de Sequência

A Figura 2 apresenta o diagrama de sequência correspondente ao caso de uso *cadastrar usuário*, enquanto a Figura 3 refere-se ao caso de uso *consultar erros*. Tais diagramas têm a finalidade de especificar os seus casos de uso, isto é, detalhar como ocorre a transição de dados e o funcionamento do sistema quando são executados.

Para o cadastro de usuário (Figura 2) é possível verificar que, após a entrada dos dados de cadastro pelo usuário, esses dados são coletados e encapsulados no formato *JSON* pelo *frontend,* o qual os enviará, a partir de uma requisição *HTTP POST* para a *API* (backend). A *API*, por sua vez, consultará o banco de dados e, caso algum dos dados fornecidos pelo usuário já esteja em uso, será retornada uma exceção (usuário já cadastrado). Caso contrário, ou seja, se os dados não estiverem sendo utilizados por outro usuário, a *API* realizará a inserção do novo usuário no banco de dados, retornando uma mensagem de sucesso.

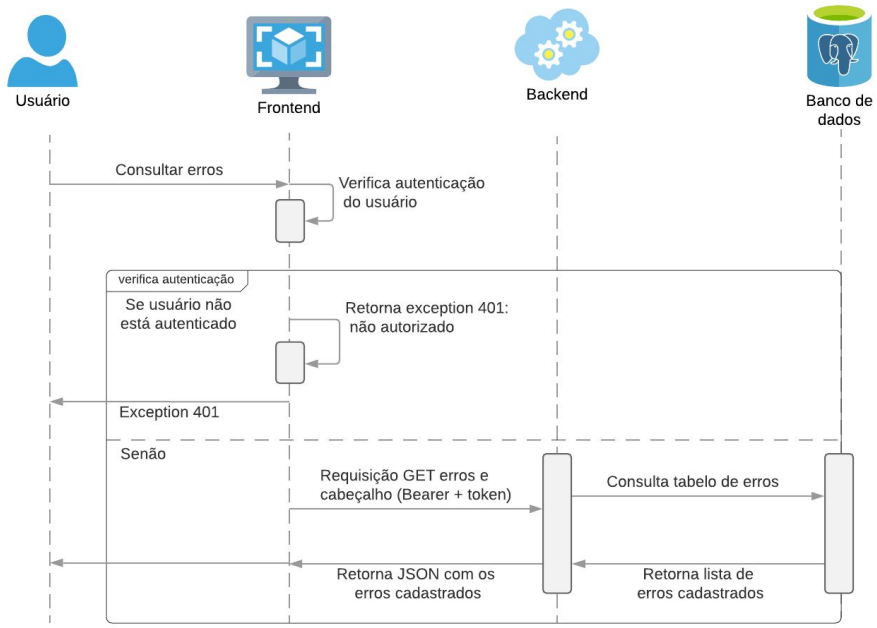
Figura 2: Diagrama de sequência (cadastro de usuário)



Fonte: Elaborado pelas autoras (2020)

Na Figura 3, o diagrama de sequência possibilita a visualização do funcionamento do caso de uso consultar erros, em que o frontend, após receber a solicitação referida, verifica a autenticação do usuário, isto é, se ele está logado e dispõe de um token de acesso. Caso a solicitação não tenha sido realizada por um usuário autenticado será retornada uma exception 401 (não autorizado), entretanto, se o usuário dispor de um token, o *frontend* fará a requisição *(HTTP GET /erros)* à *API,* a qual selecionará os erros do banco de dados e os retornará no formato *JSON*.

Figura 3: Diagrama de sequência (consultar erros)



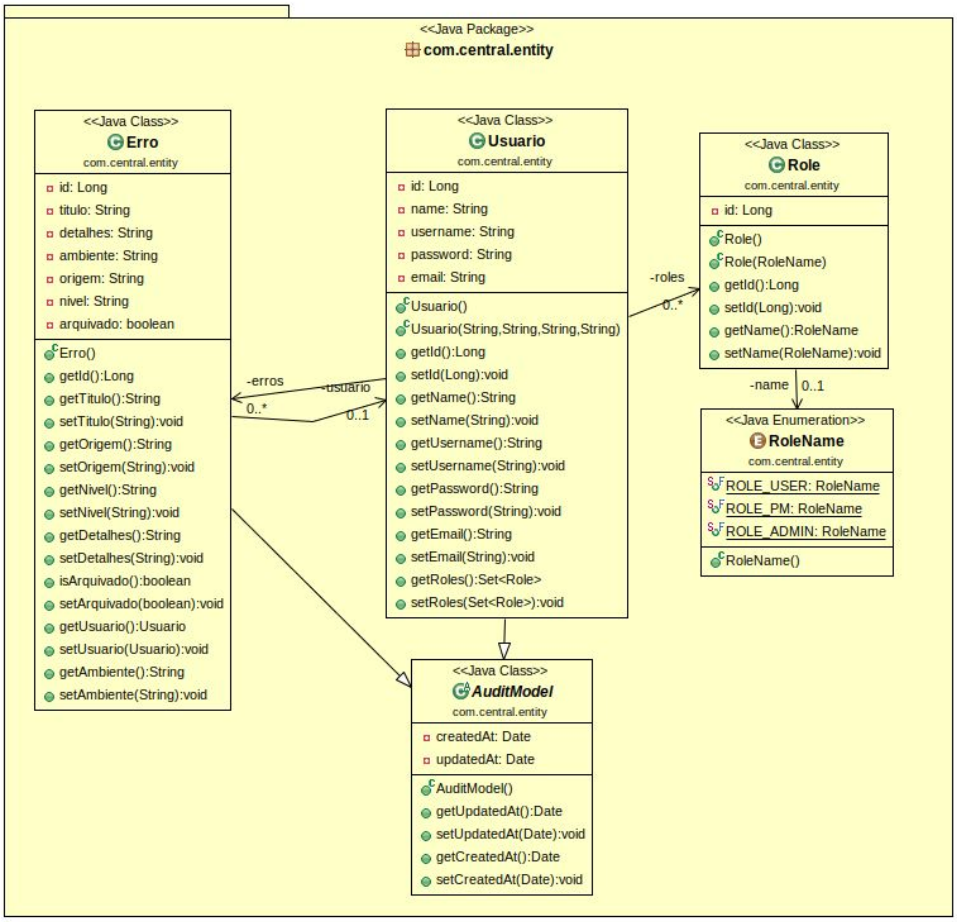
Fonte: Elaborado pelas autoras (2020)

3. Diagrama de classe

A figura 4 demonstra o diagrama de classes da aplicação em que cada classe possui seus atributos e métodos que representam as ações e comportamento de cada classe. Através do diagrama podemos observar o relacionamento entre as classes da aplicação

de maneira que temos a classe Usuario que possui um perfil registrado por na classe role que por sua vez se relaciona com a classe RoleName onde temos os perfis pré-estabelecidos; a classe Usuário se relaciona também com a classe de erros de maneira que um usuário pode acessar os erros armazenados na aplicação e esses acessos são controlados por autenticação por meio da classe AuditModel.

Figura 4: Diagrama de classes (entidades)

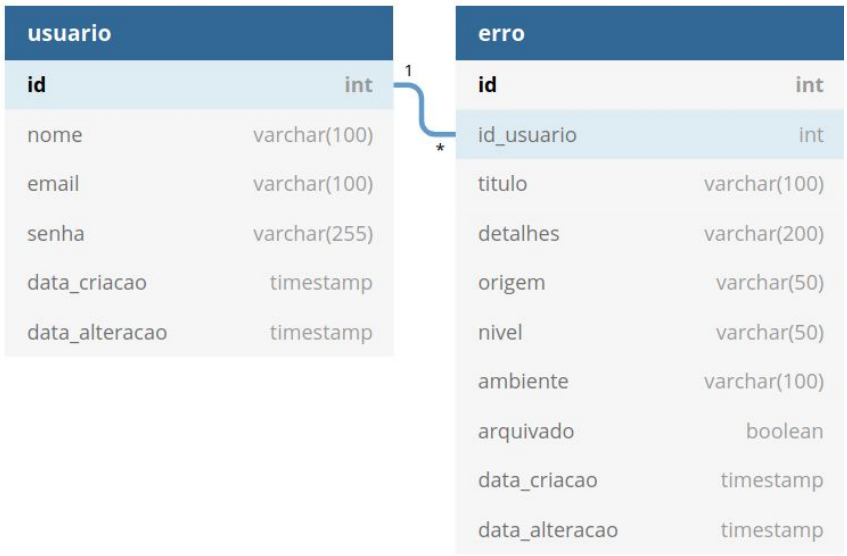


Fonte: Elaborado pelas autoras (2020)

4. Banco de dados

Esse diagrama mostra a estrutura utilizada no banco de dados para o armazenamento de informações, onde as principais tabelas são usuario e erro. Cada tabela possui a definição dos campos para guardar os dados dos registros como id para identificação única, nome, email, data de criação, detalhes de um erro, etc. Esses dados são acessados pela camada de repositório da aplicação que se comunica com o FrontEnd por meio da camada de serviço.

Figura 5: Diagrama do Banco de dados

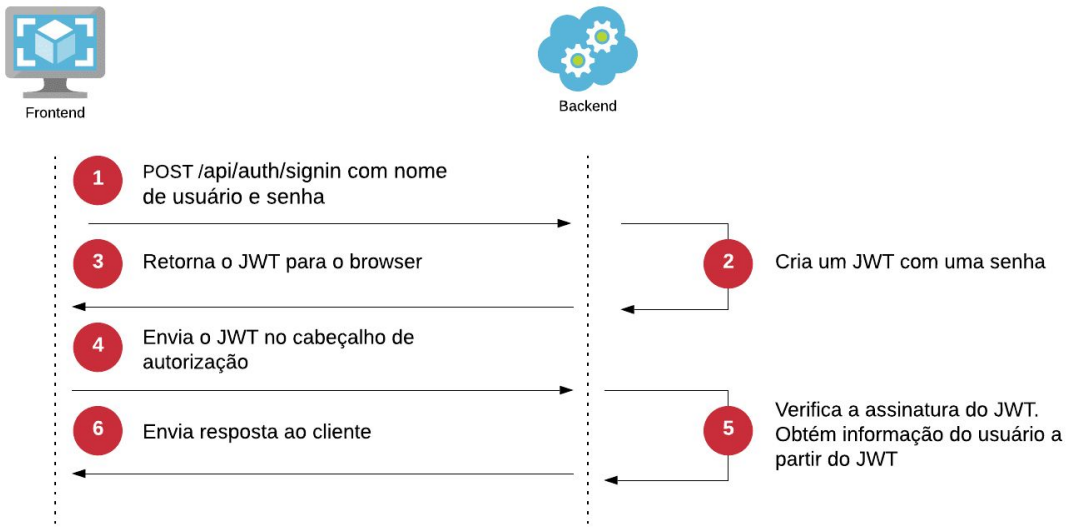


Fonte: Elaborado pelas autoras (2020)

Autenticação JWT

O Json Web Token determina uma maneira segura de troca de mensagens dentro de uma aplicação. Através das credenciais do usuário no login, após feito o cadastro, a aplicação retorna para o usuário na tela de home qual é o seu token de acesso à aplicação. Para cada página que o usuário tenta acessar, o token do mesmo é enviado por meio de cabeçalho (header). Por meio do token, é possível que o sistema verifique se o usuário está autenticado e se é possível permitir ou não o acesso para o mesmo em cada rota do sistema dependendo do seu tipo de acesso. O tipo de acesso pode ser do tipo: usuário, homologação e administrador. Desta maneira, é feita a segurança da aplicação.

Figura 6: Diagrama de sequência (login) com JWT

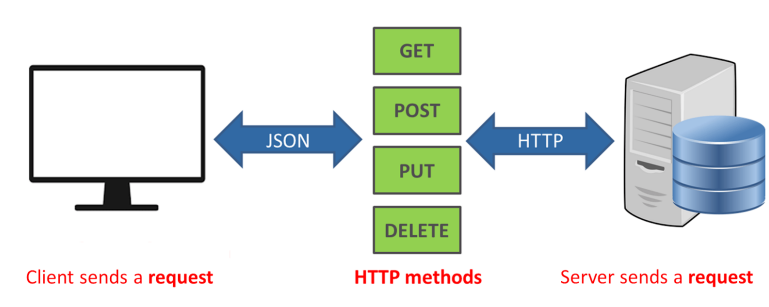


Fonte: Adaptado de https://www.toptal.com/java/rest-security-with-jwt-spring-security-and-java

Endpoints

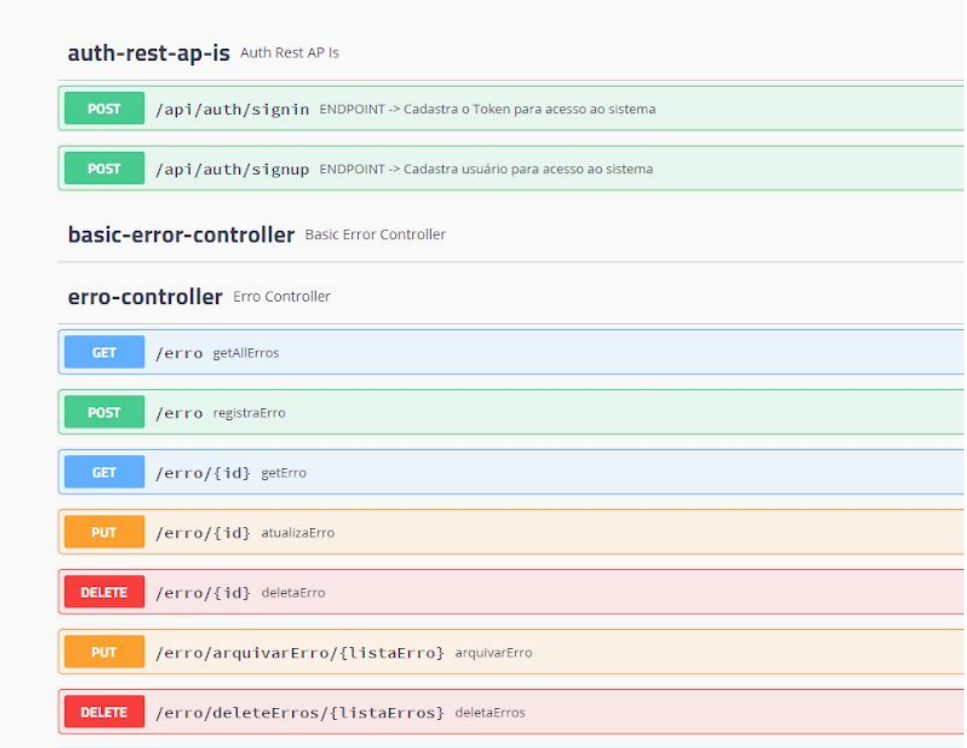
Através dos *endpoints* da API Rest LadyBug, os serviços poderão ser acessados por uma aplicação cliente, pois eles serão as interfaces entre a *API* e a aplicação consumidora. Nosso modelo foi implementado conforme a Figura 7.

Figura 7: Diagrama de sequência (login) com JWT



É importante que estes endpoints estejam devidamente protegidos e atualizados para evitar vulnerabilidades dentro da organização que for utilizá-lo. Os endpoints contidos na API LadyBug estão devidamente protegidos, sendo necessário a liberação do acesso através de um cadastro de usuário e registro de um token de segurança para acesso ao sistema. Para testar estes endpoints, bem como disponibilizar para que seja consumido e entendido por uma aplicação cliente, documentamos nossa API através do *Swagger*, conforme Figura 8:

Figura 8: Diagrama de sequência (login) com JWT



Fonte: Elaborado pelas autoras (2020)

Tecnologías utilizadas

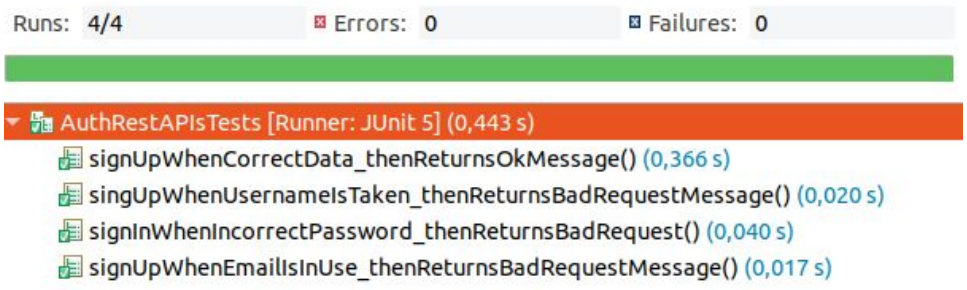
A API, backend, foi desenvolvida utilizando a linguagem de programação Java e o framework Spring. O gerenciamento do projeto foi realizado utilizando a interface de desenvolvimento Eclipse e a ferramenta Maven. Quanto ao banco de dados, foi utilizado o PostgreSQL em conjunto com o Hibernate. A autenticação, conforme supracitado, foi realizada por meio do JWT. Os testes unitários foram feitos com o JUnit enquanto o Postman foi utilizado para os testes manuais no processo de desenvolvimento. O Swagger foi usado para a descrição e documentação dos endpoints, sendo possível testá-los. Por fim, a hospedagem foi realizada pelo Heroku, na seguinte url: *https://ladybug-api.herokuapp.com/*.

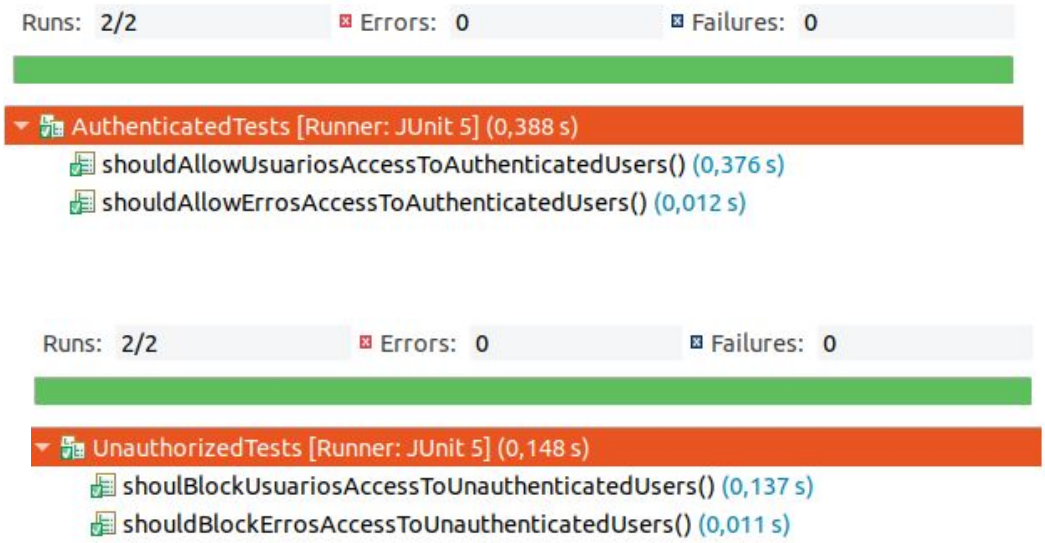
O frontend do projeto foi desenvolvido utilizando as ferramentas básicas de desenvolvimento web: CSS, HTML e JavaScript. Ademais, o framework escolhido foi o Vue, a conexão com o backend foi realizada por meio do Axios e também foram utilizadas as ferramentas NodeJS, Bootstrap e NPM. Para a hospedagem do front utilizou-se o Netlify, o qual pode ser acessado a partir da seguinte url: *https://lady-bug.netlify.app/*.

Testes

Os testes unitários do backend, API, foram realizados utilizando o JUnit, na Figura 8 é possível visualizar que existem 3 classes de testes. Na primeira classe foram feitos os testes de *signup* e *signin*, isto é, cadastro de usuário e *login*, respectivamente. Na segunda classe foram realizados os testes que autorizam acesso aos *endpoints* das entidades Erro e Usuário quando o usuário está autenticado, ou seja, quando envia o *token* do login como *header* da requisição. Por fim, na terceira classe foram realizados os testes que não autorizam as requisições referidas quando a requisição não dispõe de um *token* válido. A Figura 9 ilustra as três classes e seus respectivos testes.

Figura 9: Diagrama de sequência (login) com JWT





Fonte: Elaborado pelas autoras (2020)