Revista Eletrônica em Gestão e Tecnologia
Print ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

Osvaldo Egger Junior

Silvio Cesar Viegas

RESUMO

O trabalho que a seguir apresenta visa demonstrar e aplicar todo o conceito e aprendizado obtido durante os anos de estudo no curso de Análise e Desenvolvimento de Sistemas.

A proposta trata-se do desenvolvimento de aplicação chamada “egger-library”, que nada mais é do que um sistema de gestão pessoal de livros ou de pequenas bibliotecas.

Com o passar dos anos, cada vez mais a nossa vida tem se tornado dependente da tecnologia. Estamos praticamente sempre conectados na internet seja através do celular, computadores, notebooks, entre outros dispositivos capazes desta conexão.

Será mostrado mais a seguir no projeto que, apesar dessa tecnologia de fato estar auxiliando e melhorando cada vez mais muitos dos aspectos das nossas vidas, há um contraponto, estamos cada vez mais dependentes do celular, tanto que não é incomum ouvir ou ler histórias de pessoas que ficam sem seu aparelho e acabam tendo muitos problemas em decorrência das informações contidas nele.

Dessa maneira, a proposta do projeto “egger-library” é dar a liberdade da gestão dos livros independente do dispositivo e de onde o usuário estiver, através de uma aplicação web.

Palavras-chave: Biblioteca, Java, SpringBoot, Aplicação Web, Backend, Frontend, Integrações, Banco de Dados.

INTRODUÇÃO

O intuito do artigo a seguir é documentar todo o processo de desenvolvimento e entrega de “egger-library”. Primeiro será explicado o tema do projeto e o porquê do tema ser escolhido. Em seguida será explicado a problematização e como foi pensado na solução.

Depois disso, o artigo foca em explicar as ferramentas utilizadas e como elas funcionam dentro do escopo do projeto, explicando suas funcionalidades e dando exemplos.

Por fim, será explicado como a estrutura do projeto foi criada e também como “egger-library” deve se comportar em seu estágio final. Após isso temos o fim do artigo apresentando os resultados obtidos e as considerações finais.

1.1 Tema

A tecnologia avançou muito nos últimos anos. A evolução é tão grande que hoje fica até complicado descrever com exatidão o tamanho da proeza que esses avanços conseguiram alcançar.

Essa evolução toda de hoje permite ao ser humano conectar-se em poucos segundos e com poucos toques com alguém do outro lado do mundo, da mesma forma conseguimos fazer uma compra de um produto que antigamente teríamos de deslocar até uma loja física e torcer para que tal produto ainda estive em estoque. Também conseguimos uma grande evolução na área da saúde, como as consultas on-line por exemplo, que facilita muito a vida de quem tem dificuldade de locomoção ou outras adversidades.

Tudo isso só foi e ainda é capaz graças ao avanço da tecnologia, que permitem aos desenvolvedores criarem softwares cada vez mais robustos e otimizados. Porém, apesar da rapidez dessa evolução, não significa que seja fácil criar essas soluções. A estrutura por trás de todas essas tecnologias é muito grande e complexa, além de necessitar que pessoas extremamente capacitadas as mantenham funcionando.

Segundo a Similar Web (2022), hoje em dia temos mais de 65% do tráfego da internet passando por aparelhos mobile. Isso acontece muito por conta de existir uma quantidade imensurável de aplicativos.

Pensando dessa maneira, muitas pessoas acabam não aproveitando de todas essas novas tecnologias por não possuírem condições financeiras para comprar um aparelho mobile, ou pela idade mais avançada acabem tendo mais dificuldade, ou simplesmente não querem fazer tudo pelo celular, tendo em vista que este acaba tirando muito a atenção. Segundo a ViverBem Unimed (2021), “o uso excessivo de aparelhos celulares pode acabar acarretando vários problemas, incluindo até a dificuldade de concentração”.

Pensando em tudo o que foi citado até aqui, torna-se importante que os desenvolvedores que trabalhem em aplicações que ofereçam de fato uma vantagem aos seus usuários, permitam que essas aplicações sejam disponibilizadas não apenas em aparelhos mobile, como também em aparelhos desktop.

1.2 Delimitação do Tema

Pensando nessas soluções que atendam a um público mais amplo, será criado a “egger-library”, um sistema de gestão pessoal de livros que poderá servir tanto a clientes que apenas queiram gerir sua biblioteca de casa, como a de pequenas livrarias, que muitas vezes ou ainda fazem toda a sua gestão em planilhas de papel, ou em softwares um pouco mais avançados como o Excel, como esse último demanda uma certa aptidão do usuário com as ferramentas, surge a necessidade de uma aplicação simples, que facilite a gestão desses livros de qualquer lugar, não apenas de um dispositivo específico.

1.3 Problema

Fazer a gestão desses itens hoje em dia não é uma tarefa tão complicada se você possui um aparelho celular, a quantidade de aplicativos pra isso é razoável, o problema é que muitas vezes temos essa aplicação apenas no celular, limitando o trabalho a um só dispositivo que, como comentando anteriormente, pode atrapalhar na concentração e por consequência na gestão do negócio. Dessa maneira, como construir um sistema de gestão de livros que seja acessível por qualquer dispositivo?

1.4 Objetivo

A seguir serão apresentados o objetivo geral e os específicos, respectivamente.

1.4.1 Objetivo Geral

Desenvolver um software de gestão pessoal de livros, visando pessoas com bibliotecas pessoais ou pequenas livrarias.

1.4.2 Objetivos Específicos

Os objetivos da aplicação serão:

- Salvar livros na base de dados;
- Editar/excluir os livros da base de dados;
- Ser capaz de criar e editar usuários para que apenas pessoas autorizadas tenham acesso aos dados;
- Viabilizar o consumo desses dados via APIs REST;
- Disponibilizar o sistema em aplicação WEB, para que seja possível acessá-las independentemente do dispositivo em questão;

1.5 Justificativa

Segundo a Otimiza, é de grande importância a gestão de empresas porque “traz benefícios para diferentes aspectos do negócio: processos, equipe, resultados e até mesmo na percepção do cliente.”, ou seja, com uma boa gestão conseguimos melhorar até as pequenas partes de um negócio, consequentemente podendo aumentar os lucros, no caso das pequenas empresas e ganho de uma melhor qualidade de vida para casos pessoais, segundo a YRU Organizer (2017).

2. FUNDAMENTAÇÃO TEÓRICA

A seguinte parte do projeto irá explicar os conceitos utilizados para o desenvolvimento do projeto e depois as ferramentas que foram utilizados para isso, desde os padrões de projeto, passando pelas linguagens de programação e frameworks.

2.1 Segmento de mercado

O software proposto neste trabalho tem como norte o mercado de aplicações web. Tal segmente tem muitas vantagens hoje em dia, MadeInWeb elenca alguma delas (2020): “Criptografia no banco de dados, economia (tendo em vista que a aplicação pode ser hospedada em qualquer servidor), a praticidade por não precisar de muitos equipamentos, (apenas um dispositivo que tenha conexão com a internet) e a

integração, que faz com que vários usuários possam utilizar o software ao mesmo tempo”.

Sendo assim, a maior vantagem da “egger-library” é a liberdade que o usuário possui, pois não importa o dispositivo e nem onde esteja fisicamente, ele será capaz de utilizar todas as ferramentas disponibilizadas.

Na seguinte seção, a proposta explicará as ferramentas utilizadas no desenvolvimento da aplicação:

2.2 BANCO DE DADOS

RockContent (2020) define que um banco de dados “é a organização e armazenagem de informações sobre um domínio específico.”

Todos os dados que trafegam na internet não vem do nada, eles precisam ficar hospedados em algum lugar de forma organizada, é aí que entra o banco de dados. Bancos de dados estão mais presentes na nossa vida do que podemos imaginar, principalmente tratando-se de software, afinal sempre que acessamos algum software, as informações nele contidas com certeza estão hospedados em algum lugar. Mas analisar, criar e manter um banco de dados não é algo tão fácil, de fato há muitas opções de infraestrutura e de como modelar um banco de dados, cabe ao desenvolver analisar a situação e decidir qual se encaixa melhor no seu cenário.

Alguns exemplos de banco de dados são: Oracle, SQL Server, MySQL, PostgreSQL, MongoDB etc.

Além dessas opções, temos 2 tipos de banco, no âmbito de infraestrutura:

BANCO DE DADOS RELACIONAL

De acordo com Kondado (2022), “no banco de dados relacional os dados são armazenados com linhas e colunas específicas que definem os dados”.

BANCO DE DADOS NÃO RELACIONAL

“Nesse tipo de banco, os dados são alocados em pastas, fator que possibilita a definição de um esquema personalizado, além de permitir a adição de novas propriedades sem que impacte as outras informações armazenadas.” (Kondado, 2022)

Tendo em vista as informações acima citadas, o banco de dados escolhido para salvar as informações do software “egger-library” foi o MongoDB, devido ao fato de ser um

banco de dados não relacional, ele traz uma melhor performance na consulta e inserção de dados quando há poucas coleções em sua infraestrutura, e como o software em questão tem uma estrutura de banco relativamente simples, ele foi a escolha adequada.

2.3 MongoDB

De acordo com o TreinaWeb (2020), “o MongoDB é um banco de dados opensource, de alta performance e flexível, sendo considerado o principal banco de dados NoSQL (Não Relacional)”. Os bancos de dados NoSQL apresentam algumas vantagens sobre outros tipos, principalmente quando precisamos de escalabilidade, flexibilidade, bom desempenho e facilidade para consultas.

Sendo o propósito da aplicação salvar dados simples, o MongoDB além de ser um banco de dados facilmente integrado a qualquer linguagem de programação, ele também disponobiliza um servidor gratuito para hospedar a estrutura de banco, o que permite que a aplicação não precise se preocupar em ela administrar o banco de alguma forma, a única coisa que ela precisa é ser capaz de se conectar a ele.

O serviço que fornece uma hospedagem de banco de dados gratuito citado acima se chama MongoDB Atlas, que, segundo Codigo Simples (2018), é o serviço de DbaaS (Banco de Dados como Serviço) oferecido pela MongoDB. Basicamente o desenvolvedor só se preocupa em administrar os dados que estarão lá, toda a infraestrutura e manutenção das máquinas, bem como segurança disso tudo fica por conta deles.

2.4 Linguagem de Programação

Segundo Na Pratica (2022), “linguagem de programação é um conjunto de regras e instruções que um programador constrói para gerar programas e softwares que serão processados por um computador, dispositivo móvel ou outro equipamento.” Quando estruturada, essa linguagem forma o código fonte de um software e informa a uma ferramenta quais ações ela deve tomar.

Resumidamente uma linguagem de programação é um idioma no qual o desenvolvedor informa o computador quais ações ele deve tomar. Ela se assemelha a linguagem humana, o que a torna compreensível para seres humanos, é papel do computador compilar os códigos para que a máquina entenda, já que que basicamente o que um computador entende são zeros e uns.

2.5 Java

A linguagem escolhida para o desenvolvimento do egger-library foi Java, devido ser uma das linguagens mais utilizadas do mundo, sua robustez e praticidade que seus frameworks possibilitam.

Segundo a própria documentação da linguagem Java, além de linguagem de programação, também é uma plataforma de computação liberada pela primeira vez pela Sun Mycrosystems em 1995. De um início humilde, ela evoluiu para uma grande participação no mundo digital dos dias atuais, oferecendo a plataforma confiável na qual muitos serviços e aplicativos são desenvolvidos. Produtos e serviços novos e inovadores projetados para o futuro continuam a confiar no Java também.

TecnoBlog (2021) ainda complementa dizendo que “diferente de outras linguagens de programação, o software não é compilado em “código nativo” para ser executado diretamente pelo computador, mas sim em um código intermediário chamado “bytecode”, que então é interpretado e executado pela máquina virtual Java (JVM, na sigla em inglês).
Dessa forma, um sistema ou aplicação criado em Java torna-se muito mais portátil, podendo ser rodado em praticamente qualquer ambiente ou dispositivo no qual o Java Virtual Machine seja instalado.”

2.6 Arquiteura Orientada a Serviços

Para poder escrever um software em qualquer linguagem de programação, é necessário escolher um paradigma que melhor se encaixe na solução proposta. Um paradigma é uma forma de organizar o código para que ele torne legível, otimizado e com uma fácil manutenibilidade, tanto para os desenvolvedores quanto para o computador, que precisa ser otimizado o máximo possível para economizar recursos físicos.

O paradigma escolhido para o desenvolvimento do “egger-library” foi a Arquitetura Orientada a Serviços.

RedHat (2020) explica que a “Arquitetura Orientada a Serviços é um tipo de design de software que torna os componentes reutilizáveis usando interfaces de serviços com uma linguagem de comunicação comum em uma rede. Um serviço é uma unidade ou conjunto de funcionalidades de software independente, que foi desenvolvido para concluir uma tarefa específica, como recuperar determinadas informações ou executar uma operação. Ele contém as integrações de dados e o código necessários para

executar uma função de negócios completa. Esses serviços podem ser acessados remotamente e é possível interagir com eles e atualizálos de maneira independente.”

Em outras palavras, SOA integra os componentes de software que foram implantados e são mantidos separadamente, permitindo que eles se comuniquem e trabalhem juntos para formar aplicações que funcionam em sistemas diferentes.

Essa arquitetura foi escolhida porque o código fica muito bem organizado, cada pacote tem seu objetivo específico, inclusive as seções de configuração. Basicamente a parte central da arquitetura são os pacotes onde ficam os serviços, que são os métodos que vão realizar uma operação, que geralmente consiste em alterar um documento do banco de dados, buscá-los, deletá-los e incluí-los.

2.7 Aplicação Web

Resumidamente, Rock Content (2022) explica que “uma aplicação web trata-se de uma aplicação de software que roda na internet, em vez de funcionar com base em sistemas operacionais.”

Esse é o ponto em que dá a liberdade do usuário e até mesmo ao desenvolvedor também. Com uma aplicação hospedada na web, não é necessário que o dispositivo do usuário seja compatível com a aplicação, pois ela vai trafegar na rede, então basta ter uma conexão na internet que vai funcionar. E a vantagem para o desenvolvedor é que para desenvolver e dar manutenção à aplicação, não é necessário estar no mesmo espaço físico do servidor, basta ter acesso à ele também através da rede.

2.8 HTTP

HTTP é a sigla de Hyper Text Transfer Protocol. Rock Content (2019) explica que HTTP é um protocolo de transferência que possibilita que as pessoas que inserem a URL do seu site na Web possam ver os conteúdos e dados que nele existem. Esse sistema é a base da comunicação que existe em toda a Internet em que os sites e conteúdos que tragam hiperlinks possam ser encontrados mais facilmente pelo público por meio de um clique do mouse ou um toque na tela.

Em outras palavras, o HTTP serve para que haja um padrão entre as comunicações entre a internet, ou seja, é um “idioma” comum que serve para todos os computadores e/ou dispositivos conectados na rede possam se entender.

2.9 Protocolo HTTP

A documentação do Mozilla (2022) explica que" o protocolo HTTP define um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada para um dado recurso. Embora os métodos possam ser descritos como substantivos, eles também são comumente referenciados como HTTP Verbs (Verbos HTTP)." Basicamente isso significa que cada comunicação que é feita via HTTP precisa ter um "verbo" como parte de sua configuração principal. Esse verbo define como a requisição precisa chegar até o servidor e se a entidade que está trafegando na rede vai sofrer alterações ou não.

Os principais verbos HTTP utilizados no "egger-library" são:

1. GET: Esse método deve apenas buscar os dados de algum lugar, sem modificálos.
2. POST: Esse método é capaz de transitar um objeto ou entidade via rede, podendo causar mudanças em um objeto já existente ou criar um novo.
3. PUT: Semelhante ao método PUT, esse método costuma substituir as propriedades de um objeto já existente ou até mesmo modificá-lo ele por completo.
4. DELETE: O método DELETE, como o nomo sugere, serve para apagar um objeto ou entidade específico de algum lugar.

2.10 Ambiente de Desenvolvimento Integrado

Ambientes de Desenvolvimento Integrado, ou IDE (do inglês Integrated Development Environment) são ferramentas que os desenvolvedores utilizam para facilitar o desenvolvimento de softwares. São programas que possuem todas as ferramentas necessárias para a criação de uma nova aplicação. As IDEs não apenas possuem o kit de desenvolvimento da linguagem embutida em si, ela também auxilia na escrita do código, dando dicas e mostrando as possíveis falhas e erros em tempo de desenvolvimento. Essa é uma vantagem de usar Java, que é uma linguagem compilada e intepretada, isso significa que ela nem deixa chegar um erro em tempo de execução, pois com certas falhas a compilação nem chega a acontecer. Nesses momentos, cabe ao desenvolvedor, com a ajuda da IDE, analisar o erro apontado e ajustá-lo.

2.11 IntelliJ

O IntelliJ IDEA é a principal IDE para desenvolvimento Java hoje em dia, segundo a própria JetBrains em sua documentação.

Essa ferramenta traz consigo as principais necessidades de um desenvolvedor em seu trabalho. Com uma interface extremamente amigável, rápida e intuitiva, ela acaba se tornando a principal IDE de muitos desenvolvedores Java hoje em dia. Ela traz uma facilidade em criar pacotes, classes etc. Tem uma poderosa sugestão de melhoria de código, sua navegabilidade entre os arquivos é muito rápida, além de que com poucos cliques e alguns atalhos ela é capaz de criar e compilar um projeto Java rapidamente.

O IntelliJ foi escolhido para o desenvolvimento do “egger-library” visando a facilidade, rapidez e ergonomia do desenvolvedor.

2.12 Ferramentas & Tecnologias

A partir das próximas seções a proposta explicará com mais detalhes a estrutura técnica da aplicação “egger-library”, com ênfase nas tecnologias utilizadas diretamente no código.

2.12.1 Frameworks

Um framework é um conjunto de códigos já escritos, alguns de forma mais genérica e outros mais especializados, que auxiliam o programador no desenvolvimento de uma aplicação. Uma vez que esses frameworks já tem a maior parte de códigos “comuns” já escritos, isso reduz o tempo de desenvolvimento e trabalho do programador, deixando o foco nas regras de negócio. Praticamente toda grande aplicação hoje em dia precisa de frameworks, porque eles são revisados e atualizados constantemente, tornando os seguros e facilitando um padrão de desenvolvimento para toda a web.

2.12.2 Spring Framework

De acordo com o Blog Geek Hunter (2020), “Spring Framework é um framework desenvolvido para a plataforma Java baseado nos padrões de projetos (Design Patterns), inversão de controle e injeção de dependências. É constituído por diversos e completos módulos capazes de dar um boost na aplicação Java.”

Em outras palavras, o Spring Framework é uma ferramenta capaz de criar uma aplicação web utilizando Java, ou seja, através de uma IDE somos capazes

de criar um servidor, com código Java, e através desse servidor trafegar dados através dos métodos HTTP explicados anteriormente.

2.12.3 SpringBoot

Zup (2021) explica que “o SpringBoot é um framework Java opensource que tem como objetivo facilitar esse processo em aplicações Java. Consequentemente, ele traz mais agilidade para o processo de desenvolvimento, uma vez desenvolvedores conseguem reduzir o tempo gasto com as configurações iniciais.”

O SpringBoot serve basicamente para reduzir ainda mais o trabalho do desenvolvedores com configurações que seguidamente seriam as mesmas em diferentes projetos, como configurações de servidores, portas, entre outras configurações internas. Com o SpringBoot todas essas configurações são geradas automaticamente, e o que faltar basta o desenvolvedor utilizar algumas anotações pré configuradas e pronto. Com um clique de “play” na IDE o servidor já sobe em pouquíssimo tempo.

O SpringBoot foi o principal framework utilizado no “egger-library”, devido sua fácil compreensão e instalação, ele torna a aplicação extremamente ágil, de fácil compreensão e leitura, fácil manutenibilidade e a oportunidade de hospedar a aplicação em um servidor Google Cloud.

2.12.4 Testes Unitários

Digite explica que um teste unitário consiste em verificar o comportamento das menores unidades em sua aplicação.

Os testes unitários são essenciais em qualquer aplicação para que haja uma confiabilidade grande nos serviços que ela disponibiliza. O teste unitário é um método dentro da própria aplicação que testa os métodos que servem ao usuário, diretamente ou indiretamente.

Um exemplo no próprio “egger-library” é a regra de negócio que mantém os títulos dos livros e o nome dos usuários em letra maíuscula, essa regra foi imposta para que haja uma padronização no banco de dados e na visualização das informações para os usuários. Para garantir que esse comportamento está correto, o próprio desenvolvedor executa este método antes da aplicação subir em um servidor local, e o teste possui um

critério de aceite que só finaliza o teste com sucesso caso o comportamento seja o esperado.

Isso não só ajuda ao desenvolvedor a encontrar as falhas durante o desenvolvimento, como ajuda a dar uma confiabilidade muito maior no código, tornando a manutenção muito menos onerosa no futuro.

Os testes unitários também auxiliam caso um método de um framework seja modificado, por exemplo, tornando o resultado de um método da aplicação diferente do esperado, nesse caso cabe ao desenvolvedor mudar a versão do framework, mudar o framework ou alterar o método de forma manual para atingir o resultado esperado.

2.12.5 JUnit

Segundo o DevMedia (2009) “o JUnit é um framework que facilita o desenvolvimento e execução de testes unitários em código Java. Ele fornece uma completa API (conjunto de classes) para construir os testes e aplicações gráficas e em modo console para executar os testes criados. Algumas vantagens descritas são: O JUnit pode verificar se cada unidade de código funciona de forma esperada, faciltia a criação, execução automática de testes e a apresentação dos resultados, é orientado a objetos e é gratuito.”

Utilizando o JUnit e o IntelliJ juntos, os testes ficam cada um com um botão de “play” do lado de sua declaração para que possamos executar cada testes separadamente, além de nos dar a opção de visualizar a cobertura de testes.

Como os testes unitários são essenciais em aplicação que visa robustez, segurança, confiabilidade e integridade tanto com os usuários para com os seus dados, “egger-library” conta com uma cobertura de testes de 100%, ou seja, todos os métodos, sejam eles públicos ou não, foram testados e com os critérios de aceite adequados, todos foram executados com sucesso.

2.12.6 Application Programming Interface

Application Programming Interface, ou simplesmente API, de acordo com Amazon Web Services (2023) “é um mecanismo que permite que dois componentes de software se comuniquem usando um conjunto de definições e protocolos.”

Uma API Rest é uma configuração que permite que um método seja disponibilizado pela internet para algum usuário, seja para o setor privado, público ou até pessoal. As APIs estão na nossa rotina diariamente e mal nos damos conta, é um assunto avançado na área de tecnologia, por isso a explicação a seguir deve facilitar o entendimento: Em quase

todo cadastro que fazemos em algum aplicativo hoje em dia, nos é perguntado o nosso endereço, mas antes mesmo de perguntar nome da rua e número, geralmente a aplicação pede a inserção do CEP. Isso se dá porque através de uma API nós enviamos o número do CEP, e ela nos retorna os dados com o nome da rua, bairro, cidade etc. Isso é extrememante facilitador porque seria custoso demais todas as empresas possuírem um banco de dados com as informações de endereço de um país inteiro, e esse custo de alguma forma seria repassado ao usuário final. Então, quando é feita uma requisição numa API de CEP, é cortado um trabalho gigantesco dos desenvolvedores. Então, basicamente, uma API é um método que é disponibilizado para que outros usuários ou até mesmo desenvolvedores possam facilitar o seu trabalho.

Dito isso, “egger-library” tem um conjunto de APIs que permitem ao usuário, através de requisições HTTP, ler, modificar e excluir as entidades salvas no banco de dados.

2.12.7 Swagger

De acordo com GR1D (2022), “o Swagger trata-se de uma aplicação opensource que auxilia desenvolvedores nos processos de definir, criar, documentar e consumir APIs REST.”

O Swagger foi aplicado no “egger-library” devido a facilidade com que ele monta a visualização das APIs contidas no projeto.

Essa ferramenta facilita muito o trabalho do desenvolvedor, principalmente em tempo de desenvolver a aplicação, porque o intuito do Swagger é montar um site listando todos as APIs que estão no código, fazendo com que o desenvolvedor não precise se lembrar do caminho completo das URLs de cada API, basta apenas adicionar algumas anotações no código, que a ferramenta reconhece e monta o site. Esse framework também auxilia quando a estrutura já está disponível, pois também é possível acessá-lo através do servidor onde a aplicação web está hospedada.

2.12.8 Sonarlint

De acordo com ILEGRA (2020), “o Sonarlint é um detector, em tempo real, de de ´”códigos fedidos” (que geram dificuldade de manutenção), bugs e vulnerabilidades. Para essas categorias temos também o nível de severidade dos problemas, que são: Minor, Major, Critical e Blocker.”

O Sonarlint é um plugin que foi adicionado no IntelliJ no desenvolvimento do

“egger-library” porque ela auxilia no desenvolvimento de um código enxuto, coeso e sem falhas. Ele fica monitorando todas as linhas que o desenvolvedor digita, e quando encontra algo que não está de acordo com os padrões definidos pelo plugin, ele avisa no console da IDE, dessa forma cabe ao desenvolvedor corrigir a possível falha ou código mal escrito.

É de extrema importância dar atenção a este plugin, porque ele é como se fosse uma versão avançada das sugestões de escrita do IntelliJ. A própria IDE costuma dar dicas muitas vezes de código mais simples ou falhas que não deixem o arquivo compilar. Já o Sonarlint consegue prever falhas futuras, como por exemplo, se o desenvolvedor optar por utilizar um framework com uma versão descontinuada, ele vai avisar com um alerta. Esse alerta não impede o desenvolvedor de continuar o seu trabalho daquela maneira, porém se ele der atenção, vai perceber que se alterar o seu código, muito provavelmente ele vai ganhar em coerência, organização e performance. Outro exemplo é que o Sonarlint é capaz de identificar se um método pode ou não lançar uma exceção (uma espécie de falha no Java em tempo de execução), avisando ao desenvolvedor que ele pode tratar essa falha para a aplicação não causar um erro estranho ao usuário final.

2.12.9 Maven

DevMedia (2012) explica que em seu cerne, “o Maven é uma ferramenta de gerenciamento e automação de construção (build) de projetos. Entretanto, por fornecer diversas funcionalidades adicionais através do uso de plugins estimular o emprego de melhores práticas de organização, desenvolvimento e manuntenção de projetos, é muito mais do que apenas uma ferramenta auxiliar.”

O Maven junto com o framework Spring, torna um projeto como o “egger-library” uma aplicação extremamente robusta. Porque o Maven, em poucas palavras, permite ao desenvolvedor adicionar tecnologias em sem projeto apenas adicionando a dependência, ou seja, apenas explicita qual tecnologia deseja e sua versão. No momento de build do projeto, por se tratar de uma aplicação web, ele se conecta na internet e relaliza o download da versão escolhida. Isso facilita muito, por exemplo, no caso do desenvolvedor quiser ganhar tempo no desenvolvimento. Há uma dependência chamada “Lombok”, que ao ser utilizada ela diminui a quantidade de código genérico escrito, ou seja, o desenvolvedor não vai digitar o que geralmente precisa nos seus projetos, mas terá as mesmas ferramentas, através de anotações que o Lombok disponibiliza. Outra coisa é que o Lombok, também através de anotação, permite ao programador mostrar as informações no console do IntelliJ conforme a aplicação roda. Isso é muito importante

porque mostra ao desenvolvedor como a aplicação está funcionando e se ela está se comportando da maneira esperada.

2.12.10 Google Cloud

De acordo com Safetec (2022), “o Google Cloud Platform é a plataforma de computação nuvem do Google que serve como servidor para outros negócios realizarem migração de VM (Máquinas Virtuais). Além de criar aplicativos e sites com o melhor desempenho, segurança e facilidade de acesso.”

Essa é uma das ferramentas mais importantes do “egger-library”, porque sem ela não seria possível disponibilizar a aplicação na internet. O Google Cloud disponibiliza que possamos entregar a aplicação em um servidor hospedado pela própria Google, e também permite ao desenvolvedor realizar a gestão da aplicação. Inclusive é possível ter monitoria do software, para saber se ele está sendo muito requisitado ou se está com um número de requisições aceitável para a robustez do servidor em que o Google o hospeda. Assim como o banco de dados MongoDB através do MongoDB Atlas, também é o Google Cloud que permite que a aplicação web seja acessível através de qualquer dispositivo, desde que o mesmo tenha uma conexão com a internet. Sem essa ferramentas, até seria possível porém seria muito mais trabalhoso e custoso, tendo em vista que o “egger-library” utiliza das versões gratuitas destas ferramentas.

2.12.11 HTML

Segundo a HomeHost (2019), o “HTML foi criado pelo britânico Tim Berners-Lee, o acrônimo HTML significa HyperText Markup Language, traduzindo ao português: Linguagem de marcação de Hipertexto. O HTML é o componente básico da web, ele permite inserir o conteúdo e estabelecer a estrutura básica de um website. Portanto ele serve para dar significado e organizar as informações de uma página web. Sem isso, o navegador não saberia exibir textos como elementos ou carregar imagens e outros conteúdos.”

O HTML foi utilizado no “egger-library” para que a visualização das informações no banco de dados e tratadas na aplicação web, sejam mostradas de forma visualmente agradável ao usuário final. Sem essa ferramenta, o usuário poderia apenas utilizar o Swagger para acessar as APIs, porém o Swagger é uma ferramenta apenas para desenvolvedores, e não para usuários, afinal ali ele mostra inclusive APIs que são usadas indiretamente pela

aplicação, como a busca de usuários. Dessa maneira, toda essa estrutura por trás da aplicação funciona normalmente enquanto o usuário final tem na sua tela apenas o que o desenvolvedor julgar necessário.

2.12.12 CSS

De acordo com Hostinger (2022), “o CSS é chamado de linguagem Cascading Style Sheet e é usado para estilizar elementos escritos em uma linguagem de marcação como o HTML. O CSS separa o conteúdo da representação visual do site. Pense na decoração da sua página. Utilizando o CSS é possível alterar a cor do texto e do fundo, fonte e espaçamento entre parágrafos.”

O CSS é uma das ferramentas mais importantes do “egger-library”, porque sem ela, toda a visualização não ficaria agradável ao usuário, e ele provavelmente não acessaria mais a aplicação devido a dificuldade que isso acarreta. Portanto o CSS serve para embelezar toda a estrutura HTML, tornando a navegação rápida, fluida, dinâmica e intuitiva.

2.12.13 JavaScript

Kenzie (2021) explica que “o JavaScript é uma linguagem de programação de alto nível criada, a princípio, para ser executada em navegadores e manipular comportamenos de páginas web.”
Com seus scripts é possível incluir, em uma página estática, elementos dinâmicos como mapas, formulários, operações numéricas, animações, infográficos interativos e muito mais.

O JavaScript (ou simplesmente JS), foi aplicado no desenvolvimento do “eggerlibrary” para que o desenvolvimento da parte visual do projeto fosse facilitado. A aplicação do JS também permite a utilização de alguns frameworks que facilitam ainda mais o trabalho no desenvolvedor no que diz respeito a parte visual de projetos.

2.12.14 Angular

Segundo TreinaWeb (2020), “Angular é um framework JS de código aberto mantido pela Google para a construção de SPA (sigla para Single Page Applications ou Aplicações de Página Única). Resumidamente, uma SPA é basicamente uma aplicação web construída em uma só página, na qual a interação e a navegação entre as sessões de uma página ocorrem de maneira a qual não é ncessária recarregar a página em cada umad essas mudanças.”

Esse framework foi utilizado no “egger-library” principalmente para economizar as requisições que o usuário precisaria fazer na página. Sendo assim, a parte visual do projeto precisa ser carregada apenas uma vez, e enquanto o usuário navegar pelas telas, a única conexão à internet que será necessária será para a chamada das APIs da aplicação web. Isso facilita muito porque a navegabilidade do site fica muito fluida e rápida. E também ajuda na economia de dados móveis, caso o usuário esteja utilizando um dispositivo móvel sem estar conectado em uma rede wifi, por exemplo. Como a chamada das APIs apenas trafegam texto na rede, então a aplicação torna-se bastante leve.

2.12.15 Bootstrap

De acordo com o TecnoBlog (2021), o “Bootstrap é uma estrutura de desenvolvimento web de código aberto e gratuita. Foi projetada para facilitar o processo de criação de sites responsivos para dispositivos móveis, fornecendo uma coleção de sintaxe para designs de modelo.”

O framework é usado para tornar o site responsivo. O que significa isso? Ao abrir um site no PC, no laptop, tablet ou abrir em qualquer smartphone, o tamanho da tela do site feito com bootstrap se ajusta automaticamente e o conteúdo se apresenta melhor.

Esse framework utilizado no “egger-library” é um dos mais interessantes, porque ele já traz muitas configurações pré estabelecidas, no que diz respeito a design de menus, navegação, cabeçalho, rodapés, entre outras seções de um site. Como a aplicação se propõe a estar disponível em qualquer dispositivo, a visualização precisa se adequar ao tamanho da tela. Fazer isso manualmente é possível mas muito custoso, tendo isso em mente foi escolhido o bootstrap para que o visual da aplicação torne-se amigável ao usuário final.

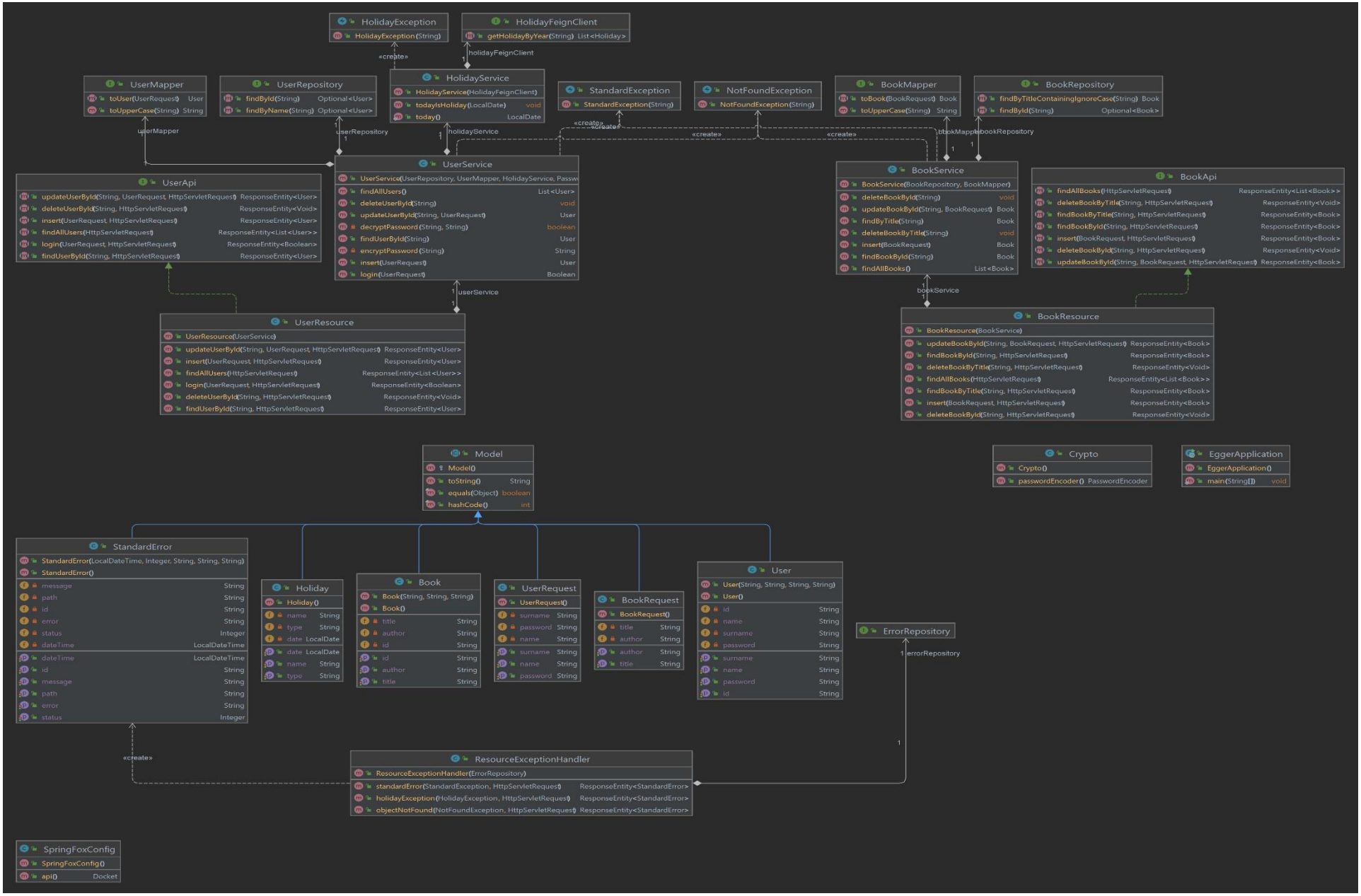
2.12.16 Diagrama de Classes (UML)

Segundo Lucidchart (2023), “a Linguagem de Modelagem Unificada (UML) ajuda o desenvolvedor a modelar sistemas de diversas maneiras. Um dos tipos mais populares na UML é o Diagrama de Classes. Bastante usado por engenheiros de software para documentar arquiteturas de software, os diagramas de classe são um tipo de diagrama de estrutura porque descrevem o que deve estar presente no sistema a ser modelado.”

Foi utilizado a UML no “egger-library” devido ao fato de que quando o desenvolvedor enxerga visualmente as classes e atributos do projeto, a estrutura do código começa a ficar mais clara, dessa maneira facilitando e diminuindo o tempo no desenvolvimento da

proposta. Ná pagina seguinte será apresentada a figura que representa a UML do “egger-library”.

Diagrama de classes de “egger-library” gerado pelo próprio IntelliJ:

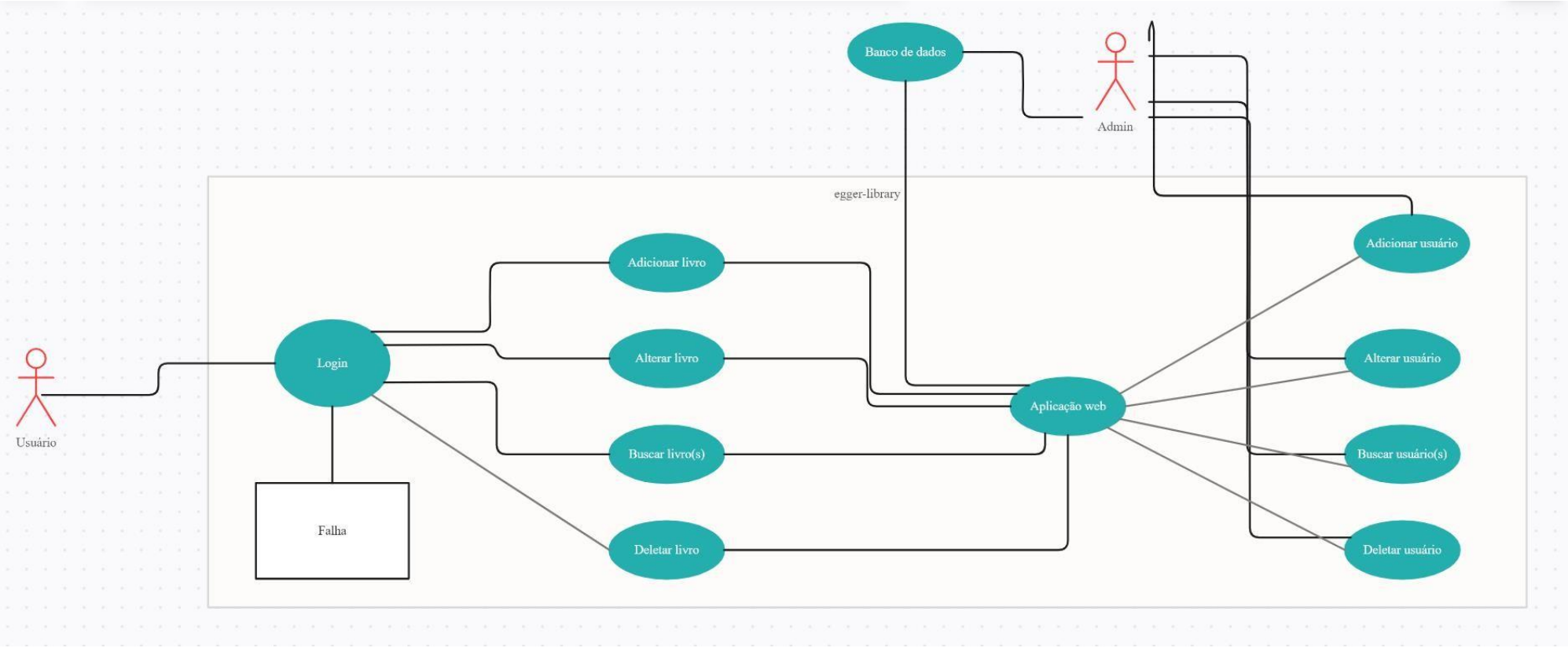

HolidayException
HolidayException(String)
HolidayFeignClient
getHolidayByYear(String) List<Holiday>
«create»
holidayFeignClient
UserMapper
toUser(UserRequest) User
toUpperCase(String) String
UserRepository
findById(String) Optional<User>
findByName(String) Optional<User>
HolidayService
HolidayService(HolidayFeignClient)
todayIsHoliday(LocalDate) void
today() LocalDate
StandardException
StandardException(String)
NotFoundException
NotFoundException(String)
BookMapper
toBook(BookRequest) Book
toUpperCase(String) String
BookRepository
findByTitleContainingIgnoreCase(String) Book
findById(String) Optional<Book>
userMapper
userRepository
holidayService
bookMapper
bookRepository
UserApi
updateUserById(String, UserRequest, HttpServletRequest) ResponseEntity<User>
deleteUserById(String, HttpServletRequest) ResponseEntity<Void>
insert(UserRequest, HttpServletRequest) ResponseEntity<User>
findAllUsers(HttpServletRequest) ResponseEntity<List<User>>
login(UserRequest, HttpServletRequest) ResponseEntity<Boolean>
findUserById(String, HttpServletRequest) ResponseEntity<User>
UserService
UserService(UserRepository, UserMapper, HolidayService, Passw
findAllUsers() List<User>
deleteUserById(String) void
updateUserById(String, UserRequest) User
decryptPassword(String, String) boolean
findUserById(String) User
encryptPassword(String) String
insert(UserRequest) User
login(UserRequest) Boolean
BookService
BookService(BookRepository, BookMapper)
deleteBookById(String) void
updateBookById(String, BookRequest) Book
findByTitle(String) Book
deleteBookByTitle(String) void
insert(BookRequest) Book
findBookById(String) Book
findAllBooks() List<Book>
BookApi
findAllBooks(HttpServletRequest) ResponseEntity<List<Book>>
deleteBookByTitle(String, HttpServletRequest) ResponseEntity<Void>
findBookByTitle(String, HttpServletRequest) ResponseEntity<Book>
findBookById(String, HttpServletRequest) ResponseEntity<Book>
insert(BookRequest, HttpServletRequest) ResponseEntity<Book>
deleteBookById(String, HttpServletRequest) ResponseEntity<Void>
updateBookById(String, BookRequest, HttpServletRequest) ResponseEntity<Book>
userService
bookService
UserResource
UserResource(UserService)
updateUserById(String, UserRequest, HttpServletRequest) ResponseEntity<User>
insert(UserRequest, HttpServletRequest) ResponseEntity<User>
findAllUsers(HttpServletRequest) ResponseEntity<List<User>>
login(UserRequest, HttpServletRequest) ResponseEntity<Boolean>
deleteUserById(String, HttpServletRequest) ResponseEntity<Void>
findUserById(String, HttpServletRequest) ResponseEntity<User>
BookResource
BookResource(BookService)
updateBookById(String, BookRequest, HttpServletRequest) ResponseEntity<Book>
findBookById(String, HttpServletRequest) ResponseEntity<Book>
deleteBookByTitle(String, HttpServletRequest) ResponseEntity<Void>
findAllBooks(HttpServletRequest) ResponseEntity<List<Book>>
findBookByTitle(String, HttpServletRequest) ResponseEntity<Book>
insert(BookRequest, HttpServletRequest) ResponseEntity<Book>
deleteBookById(String, HttpServletRequest) ResponseEntity<Void>
Model
Model()
toString() String
equals(Object) boolean
hashCode() int
Crypto
Crypto()
passwordEncoder() PasswordEncoder
EggerApplication
EggerApplication()
main(String[]) void
StandardError
StandardError(LocalDateTime, Integer, String, String, String)
StandardError()
message String
path String
id String
error String
status Integer
dateTime LocalDateTime
dateTime LocalDateTime
id String
message String
path String
error String
status Integer
Holiday
Holiday()
name String
type String
date LocalDate
date LocalDate
name String
type String
Book
Book(String, String, String)
Book()
title String
author String
id String
id String
author String
title String
UserRequest
UserRequest()
surname String
password String
name String
surname String
name String
password String
BookRequest
BookRequest()
title String
author String
author String
title String
User
User(String, String, String, String)
User()
id String
name String
surname String
password String
surname String
name String
password String
id String
ErrorRepository
errorRepository
«create»
ResourceExceptionHandler
ResourceExceptionHandler(ErrorRepository)
standardError(StandardException, HttpServletRequest) ResponseEntity<StandardError>
holidayException(HolidayException, HttpServletRequest) ResponseEntity<StandardError>
objectNotFound(NotFoundException, HttpServletRequest) ResponseEntity<StandardError>
SpringFoxConfig
SpringFoxConfig()
api() Docket

2.12.17 Diagrama de Caso de Uso

DevMedia (2012) descreve um Diagrama de Caso de Uso como um “diagrama que documenta o que o sistema faz do ponto de vista do usuário”. Em outras palavras, ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema. Nesse diagrama não nos aprofundamos em detalhes técnicos que dizem como o sistema faz.

Essa ferramenta é bastante importante para demonstrar o fluxo que o usuário pode passsar ao acessar o “egger-library”, mostrando como e por onde ele deve passar para chegar ao seu objetivo final. Na página seguinte será mostrada a figura que represetanda o diagrama de caso de uso do projeto.

Diagrama de caso de uso de “egger-library”:



2.13 Desenvolvimento

O desenvolvimento de “egger-library” correu tranquilamente com o auxílio das ferramentas e tecnologias acima citadas e descritas. O tempo de desenvolvimento ocorreu por volta de duas semanas, sendo a maior parte do tempo aplicado na escrita e revisão de código, seguindo da configuração dos ambientes para deixar a aplicação minimamente funcional.

3. METODOLOGIA

“Metodologia é o estudo analítico e crítico dos métodos de investigação e de prova. A metodologia não é, senão, uma reflexão sobre a atividade científica que está sendo desenvolvida para obter, em determinado momento, um retrato dessa atividade – retrato esse que deferirá de acordo com a Ciência sobre a qual estamos refletindo. “ (DENCKER; VIÁ, 2001)

3.1. Pesquisa Aplicada

A pesquisa aplicada é, segundo Via Carreira (2021), “uma atividade em que o conhecimento, previamente adquirido através de outros estudos, é usado para coletar, selecionar e processar fatos e dados. Assim, torna-se possível fazer a comprovação de resultados e gerar algum tipo de impacto.”

O projeto “egger-library” utilizou da pesquisa aplicada para poder ter mais estofo no que diz respeito a suas funcionalidades. Dessa maneira, o projeto não se torna apenas uma teoria, e sim algo que algo que agrega valor a algum usuário.

3.2 Cronograma de Atividades

O cronograma de atividades do “egger-library” foi baseado na figura ilustrada abaixo que foi inspirada no Projeto Aplicado, Luis Gomes(2018):

1.1 - Área de atuação 1.2 - Linguagem de Programação 1.3 - Banco de Dados	**1 - DEFINIÇÃO**
2 - CONHECIMENTO	2.1 - Plataformas de Gestão 2.2 - Regras de Negócio
3.1 - Diagrama de Casos de Uso 3.2 - Moldes Estruturais UML	**3 - MODELAGEM**
4 - DESENVOLVIMENTO	4.1 - Arquitetura 4.2 - Código 4.3 - Validações 4.4 - Telas
5.1 - Análise	**5 - RESULTADO**

Fonte: O prório autor (2023)

1. DEFINIÇÃO

Nesta seção será apresentada de uma forma mais ampla e genérica, como o projeto “egger-library” foi construído.

1.1 Área de Atuação

Este tópico define o ponto focal do projeto; define o que, quem e como ele ira atender suas expectativas.

1.2 Linguagem de Programação

Foi escolhida a linguagem Java por se tratar da linguagem padrão da instituição de ensino FAQI e por familiaridade do desenvolvedor que projetou.

1.3 Banco de Dados

Foi escolhido o banco de dados não relacional MongoDB para facilitar e otimizar as consultas dos dados salvos e ter maior poder de manutenibilidade futura, tendo em vista que essa estrutura permite a adição ou remoção de atributos sem afetar a estrutura existente.

2. CONHECIMENTO

Nesta etapa foi escolhido o escopo das funcionalidades e as foram definidas as regras para que “egger-library” pudesse funcionar corretamente.

2.1 Plataformas de Gestão

Esse tema foi escolhido por se tratar de um tema extremamente importante na vida atual das pessoas para ajudá-las a terem um melhor controle de sua organização.

2.2 Regras de Negócio

Nesta seção foi definido como de fato os métodos da aplicação funcionam.

3. MODELAGEM

Neste ponto foram feitas representações visuais do esqueleto da estrutura do projeto “egger-library”.

3.1 Diagrama de Caso de Uso

Aqui nesta seção foi feito uma figura que representa as ações do usuário perante a interface disponibilizada por “egger-library”.

3.2 Moldes Estruturais UML

Neste ponto foi representado visualmente o código Java de “eggerlibrary”, através da UML para referencias as classes.

4. ARQUITETURA

Esta seção serviu para montar a estrutura do código em si, como a divisão dos atributos por classes e divisão de classes por pacotes.

4.1 Arquitetura

Foi definido que no “egger-library” seria usado a Orientação a Objetos e Arquitetura Orientada a Serviços.

4.2 Codificação

Esta seção representa a escrita do código fonte em Java do projeto.

4.3 Validações

Foi definido que “egger-library” terá todos os seus métodos, visíveis ou não ao usuário final, cobertos com testes unitários, garantindo assim a certeza de que o projeto está funcionando como deveria.

4.4 Telas

Na última seção do desenvolvimento foram criadas as telas do projeto para que a visualização ficasse mais amigável.

5. RESULTADO

A seção de resultado visa demonstrar como o projeto se comportou em seus testes.

5.1 Análise

Nesta seção foi analisado o resultado de “egger-library”, visando descobrir seus pontos fracos e fortes e procurar melhorias para melhorar a experiência do usuário.

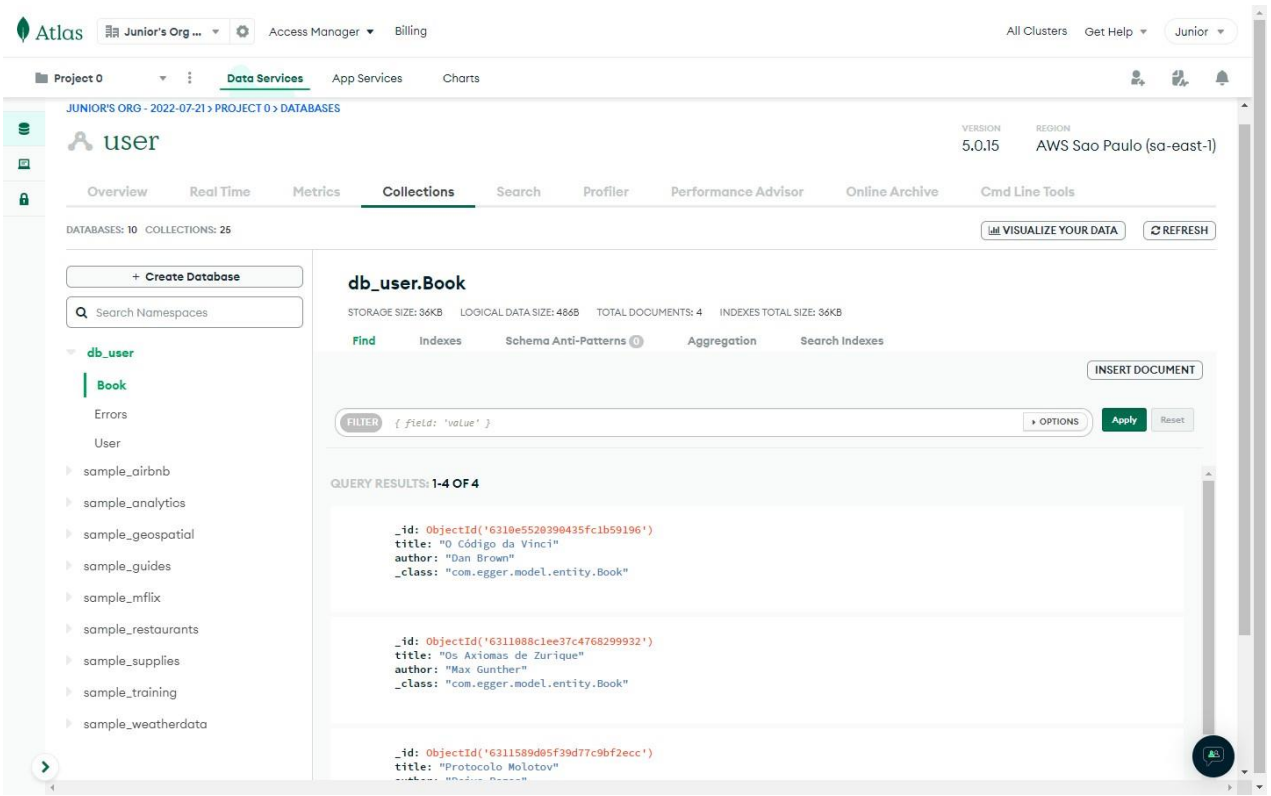
3.4 DESENVOLVIMENTO DO PROJETO

A etapa de desenvolvimento do projeto, por sua vez, visa explicar o passo a passo da criação do projeto, desde sua concepção, até as telas finais, detalhando os passos mais importantes.

O primeiro passo do desenvolvimento de “egger-library” foi entender a necessidade de sua existência, ou seja, entender o porquê dele ser necessário atualmente. Foi entendido durante este processo que seria necessário porque uma aplicação web que está disponível vinte e quatro horas por dia para o usuário independentemente do dispositivo não é algo tão comum como deveria ser, além de ajudar na organização pessoal e até mesmo pessoal para que a qualidade de vida do usuário, seja pessoal ou profissional, seja melhor.

Logo após isso, foi decidido que a gestão de livros é um ótimo exemplo para desenvolver algo do gênero, porque são objetos fáceis de tratar e com poucos dados, sendo uma forma barata de usar os recursos de rede a favor do usuário.

A primeira parte prática do projeto então foi criar o banco de dados. Foi utilizado o MongoDB Atlas, para que ficasse disponível de forma gratuita para o usuário. A figura seguinte demonstra a ferramenta do MongoDB Atlas no navegador.



Captura de tela do MongoDB Atlas (2023)

Com o banco de dados configurado, partimos então para o diagrama de caso de uso, para definir o fluxo do que o usuário esteja autorizado a usufruir, a seguinte figura demonstra tal fluxo:

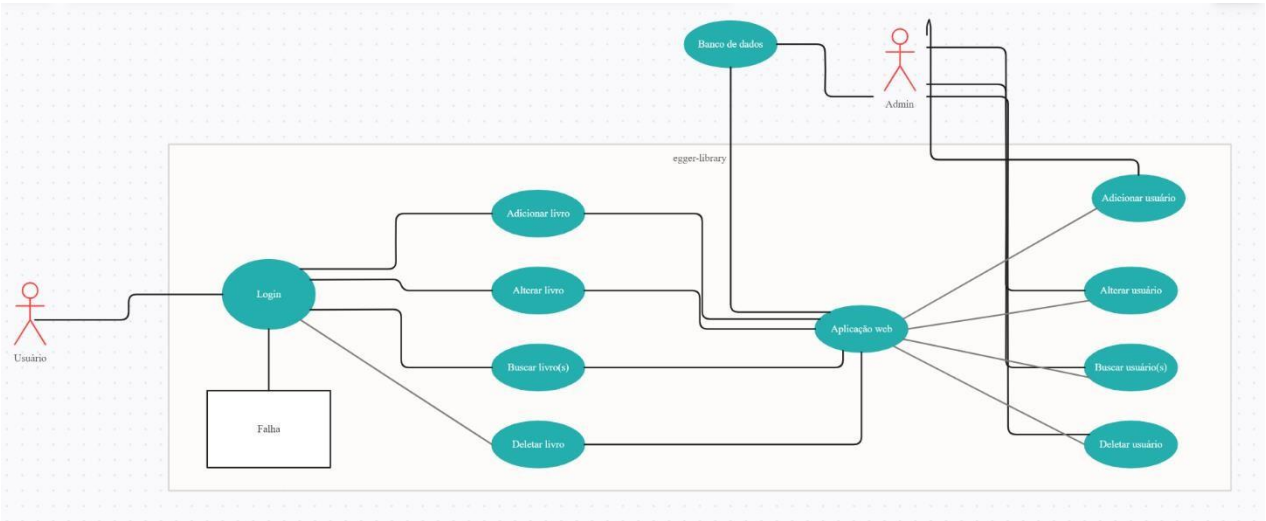
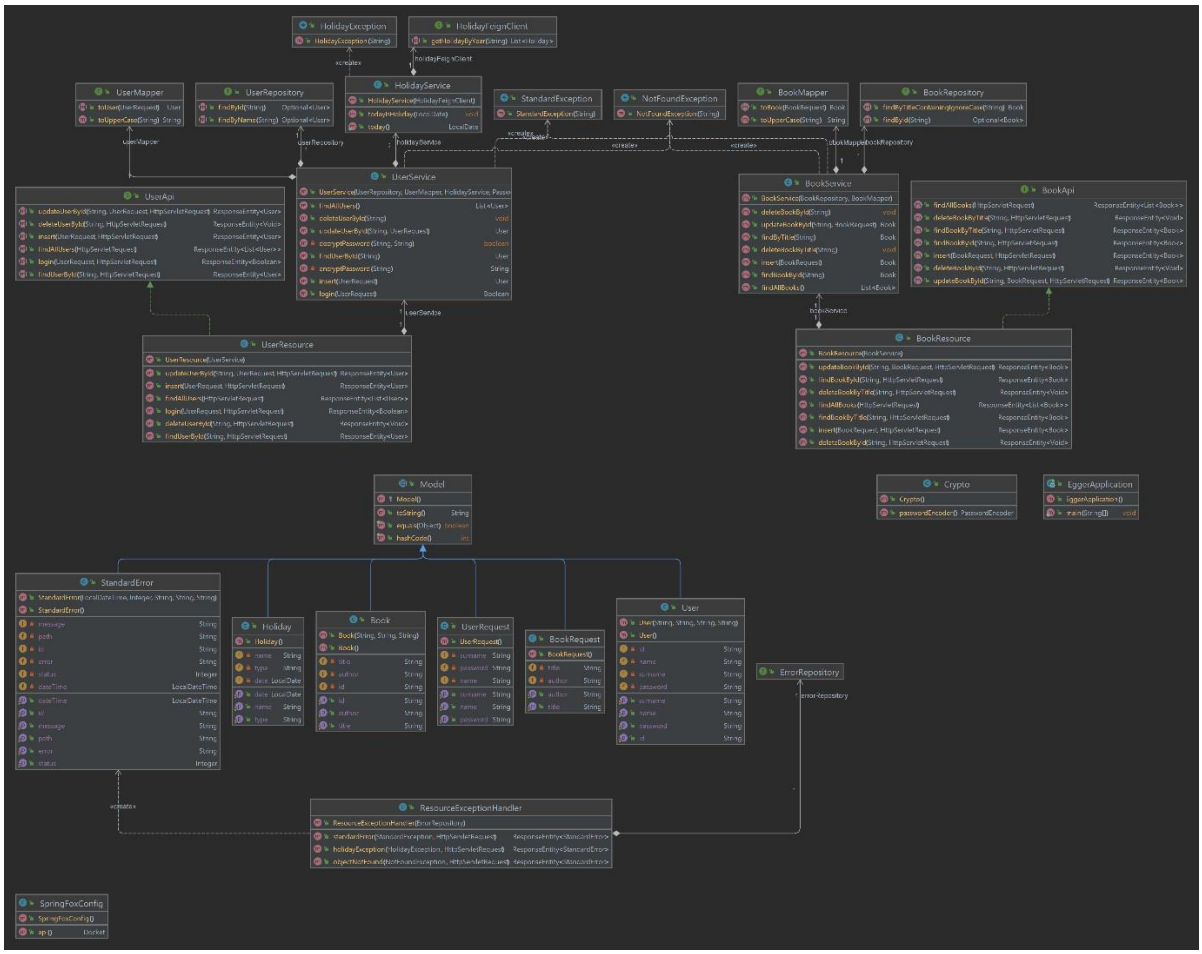


Diagrama de caso de uso de “egger-library” (2023)

Após o fluxo do usuário estar completo, chegamos no momento em que devemos estrutura o código de uma forma visual, para que o desenvolvedor consiga se guiar no momento em que estiver trabalhando, a figura seguinte representa a UML.



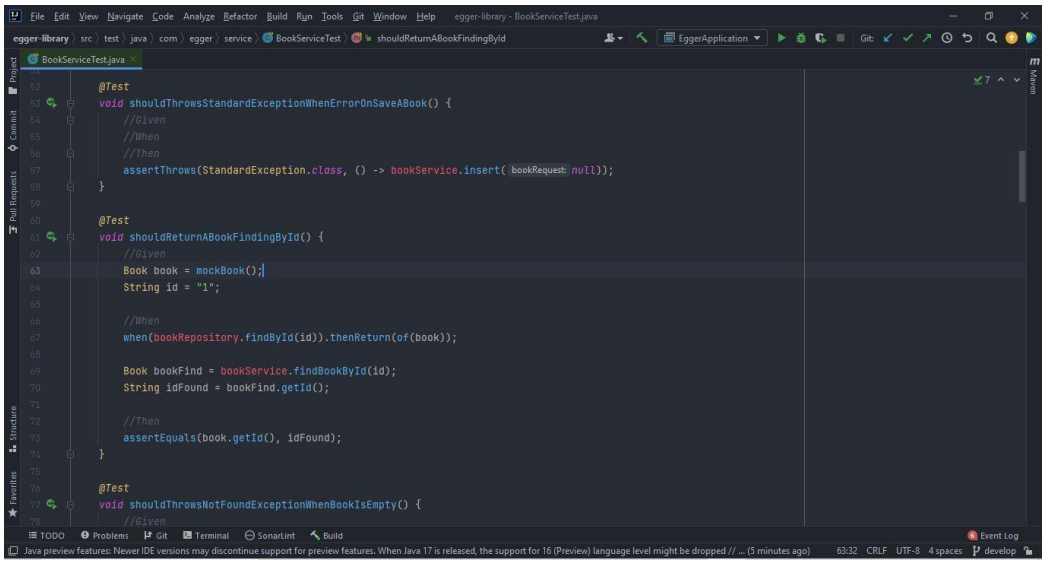
UML Completa de “egger-library” (2023)

Logo após a definição da modelagem da estrutura ntão iniciou-se a etapa de desenvolvimento do backend da Aplicação Web, para que o usuário não se conecte diretamente ao banco de dados, evitando o envio de dados com estrutura errada. As seguintes imagens demonstram alguns trechos de código.

```java
public Book insert(BookRequest bookRequest) {
    if (bookRequest != null) {
        return bookRepository.save(bookMapper.toBook(bookRequest));
    }
    throw new StandardException("O OBJETO NÃO PODE SER NULO!");
}

public Book findBookById(String id) {
    Optional<Book> book = bookRepository.findById(id);
    if (book.isEmpty()) {
        throw new NotFoundException(BOOK_NOT_FOUND);
    }
    return book.get();
}

public Book findByTitle(String title) {
    Book book = bookRepository.findByTitleContainingIgnoreCase(title);
    if (book == null) {
        throw new NotFoundException(BOOK_NOT_FOUND);
    }
    return book;
}

public List<Book> findAllBooks() {
    List<Book> books = bookRepository.findAll();
    if (books.isEmpty()) {
        throw new StandardException(EMPTY_LIST);
```

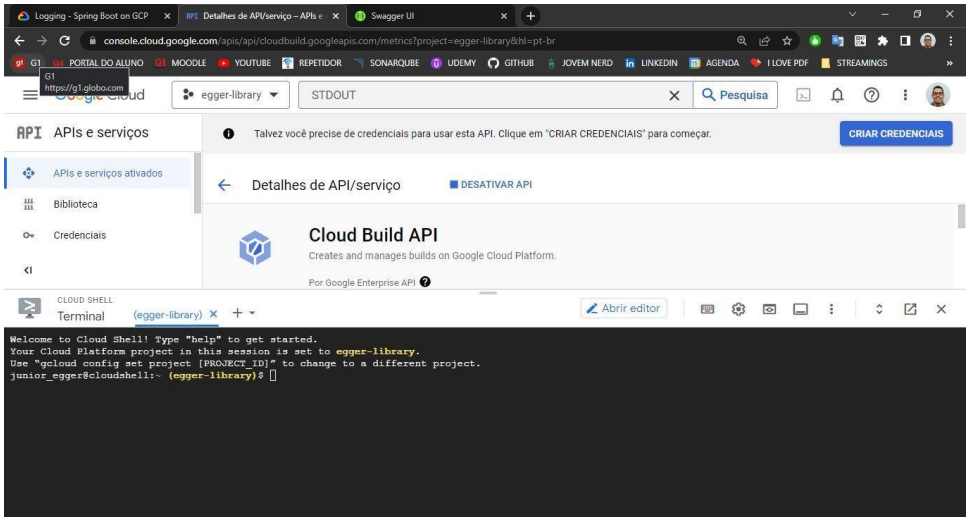
A figura acima demonstra a classe de serviço dos livros.

Após a codificação da UML em métodos, chegou o momento em que a aplicação passou pela primeira etapa de validação, os testes unitários. Com os testes desenvolvidos com a semântica correta, é garantido que cada componente do código funcione de maneira adequada. O critério criado pelo próprio autor do projeto diz que nenhuma funcionalidade deve ser aplicada ao projeto sem que o teste a esteja cobrindo. A seguir temos algumas imagens dos testes unitários da aplicação.



A figura acima demonstra a estrutura dos testes unitários da classe de serviço dos books.

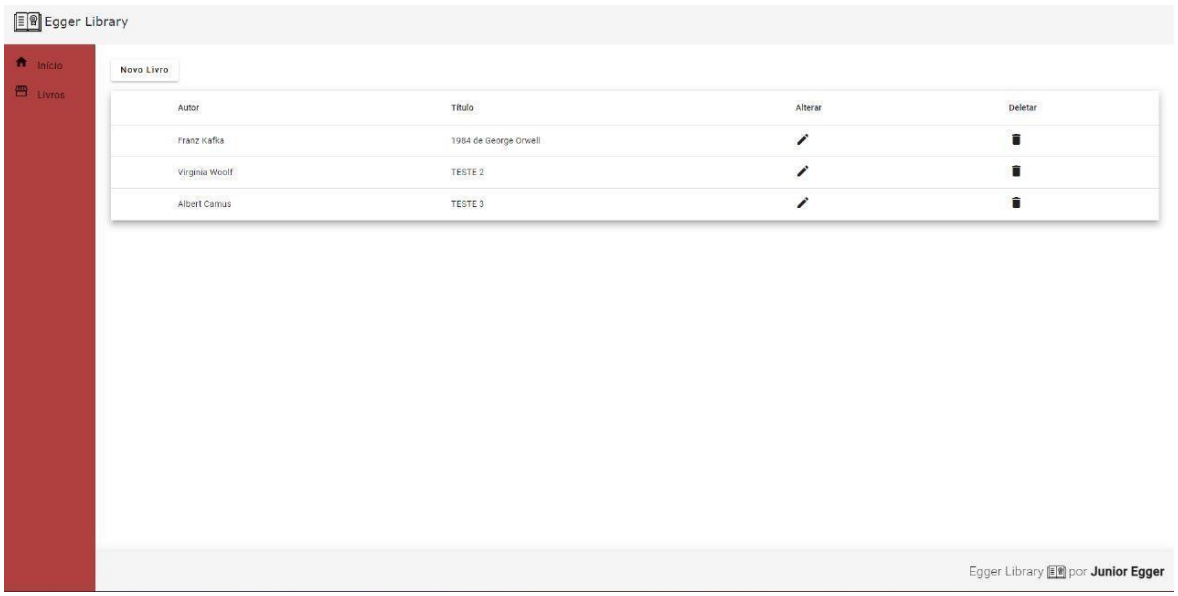
Com os testes prontos, dessa chegou o momento de disponibilizar as APIs criadas na web. Foi escolhido o serviço do Google Cloud pela sua robustez, escalabilidade de gratuidade. A seguir temos algumas imagens da ferramenta do Google Cloud.



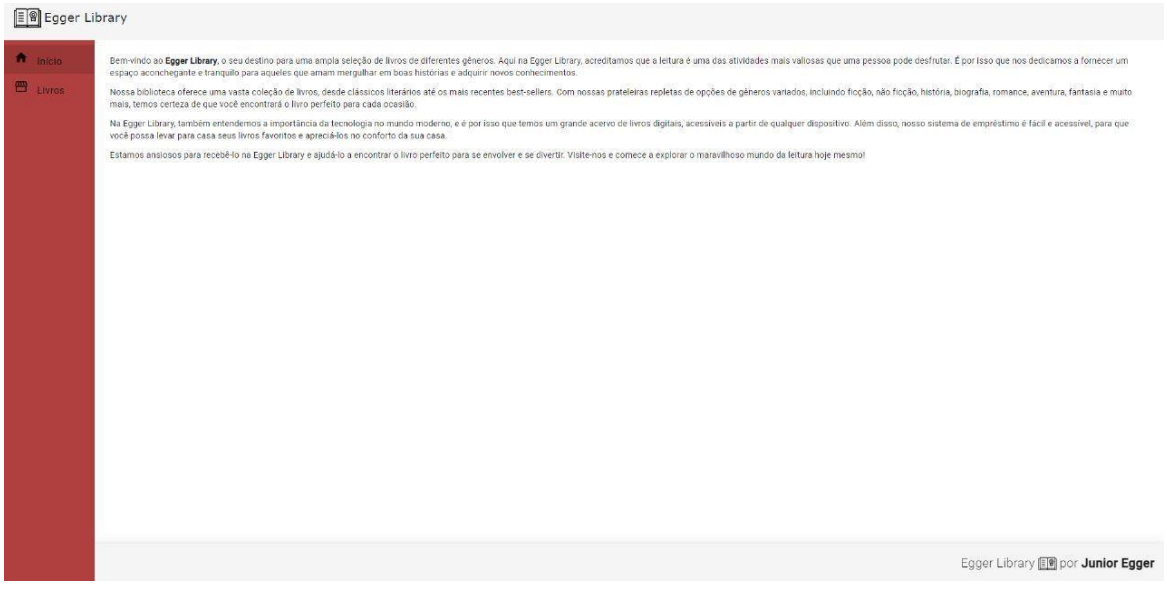
A figura acima demonstra a interface do Google Cloud Platform, onde podemos realizar a entrega da aplicação web na internet.

Com a aplicação web e já podendo ser consumida diretamente através de seu endereço, começamos o desenvolvemos da parte visual do projeto “egger-library”, chamado “frontend”.

Primeiro foi decidido que o projeto usaria os frameworks Angular e Bootstrap para facilitar o trabalho do desenvolvedor com as telas da aplicação. Depois foi começado de fato o desenvolvimento das telas utilizando HTML, CSS e os frameworks citados anteriormente. Finalmente, temos a seguir algumas imagens de como a parte visual ficará para os usuários:



A figura acima demonstra a tela de listagem dos livros.



A figura acima mostra a tela inicial de “egger-library”, onde o usuário vai começar sua navegação.

4. RESULTADOS

Com isso tudo acima descrito, a ideia é que “egger-library” seja um sistema de rápido aprendizado e de fácil manuseio por qualquer usuário, independentemente de sua familiaridade com tecnologia.

Em um primeiro momento, dependendo da quantidade de itens que o usuário possa vir a ter em seu acervo pessoal, talvez seja demorada a configuração do projeto como um tudo. Mas uma vez que estejam todos os livros cadastrados, a manutenção dos mesmos será muito facilitada, sendo este o principal propósito de “egger-library”.

5. CONSIDERAÇÕES FINAIS

O resultado final obtido com “egger-library” foi extremamente satisfatório, porque foi possível chegar no propósito desejado no início do projeto. Foi possível construir um projeto amplo e robusto, com soluções simples mas que entregam muito bem ao usuário o que ele deseja.

Foram usadas várias ferramentas diferentes no projeto, o que tornou um desafio para o desenvolvedor a conexão entre elas, mas foi tudo conectado perfeitamente, dessa maneira cada uma tem a sua própria responsabilidade e não precisa cuidar do que não é de seu escopo.

Um ponto que não foi possível implementar no projeto devido ao escasso tempo de desenvolvimento foi um sistema de login e senha de usuários e a gestão de acessos com isso. Até foram criados alguns métodos que recebem e cripografam senha, mas as ferramentas necessárias para que este sistema funcionasse corretamente demandariam uma pré análise e um desenvolvimento muito grande, não se encaixando no tempo do projeto.

Outro aspecto que não foi possível ter implementado no projeto foi a instalação do frontend da aplicação no Github Pages devido ao grande tamanho dos arquivos de código, mas esse problema é facilmente corrigido escolhendo outro

sistema de hospedagem de sites, pelo escasso tempo do projeto, não foi possível implementar alguma dessas outras soluções.

Um ponto de melhoria observado para uma futura implementação no projeto é o desenvolvimento de uma aplicação para dispositivos móveis de “egger-library”, uma vez que o projeto foi originalmente criado para ser consumido em um desktop e responsivo para navegadores de celular, seria uma boa opção aos usuários poderem usar um aplicativo para poderem ter uma forma ainda mais fluida em sua navegação.

Referências Bibliográficas

SIMILAR WEB. **Simular Web.** Plataformas mais utilizadas. Brasil: SimilarWeb, 2023. Disponível em: https://www.similarweb.com/pt/platforms/. Acesso em: 3 mar. 2023.

VIVER BEM. **Viver Bem.** Uso Excessivo do Celular. Brasil: Unimed, 2022. Disponível em: https://viverbem.unimedbh.com.br/prevencao-e-controle/usoexcessivo-do-celular/#:~:text=Tem%20dificuldades%20de%20concentra%C3%A7%C3%A3o%3F,at%C3%A9%20em%20aspectos%20mais%20importantes. Acesso em: 3 mar. 2023.

FÁTIMA LIMA. **Otimiza Pro.** Importância da Gestão. Brasil: Otimiza, 2021. Disponível em: https://otimiza.pro/gestao-empresarial/. Acesso em: 3 mar. 2023.

JULIANA FARIA. **YRU.** Organização faz bem para a saúde. Brasil: YRU, 2017. Disponível em: https://otimiza.pro/gestao-empresarial/. Acesso em: 3 mar. 2023.

MADEINWEB. **MadeinWeb.** Vantagens Aplicação Web. Brasil: MadeinWeb, 2020. Disponível em: https://madeinweb.com.br/por-que-investir-em-umsistema-web/#:~:text=Vantagens%20de%20um%20sistema%20web&text=Porque%20melhora%20a%20usabilidade%20e,lugar%20e%20%C3%A0%20qualquer%20 hora. Acesso em: 3 mar. 2023.

IVAN DE SOUZA. **Rock Content.** Banco de Dados. Brasil: Rock Content, 2020. Disponível em: https://rockcontent.com/br/blog/banco-de-dados/. Acesso em: 4 mar. 2023.

TEAM, Kondado. Banco de Dados: O que é e quais são os principais tipos?. *In:* Kondado. **Kondado.** Brasil, 13 set. 2022. Disponível em: https://kondado.com.br/blog/blog/2022/09/13/banco-de-dados-o-que-e-e-quaissao-os-principais-tipos/. Acesso em: 4 mar. 2023.

MARYLENE GUEDES. **Treina Web.** Mongo DB. Brasil: Treina Web, 2020. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-mongodb. Acesso em: 4 mar. 2023.

LEANDRO DOMINGUES. **Codigo Simples.** MongoDB Atlas. Brasil: Codigo Simples, 2018. Disponível em: https://codigosimples.net/2018/04/23/conhecendo-o-mongodb-atlas-o-dbaasda-mongodb/. Acesso em: 4 mar. 2023.

TATYANE MENDES. **Na Pratica.** Linguagem de Programação. Brasil: Na Pratica, 2023. Disponível em: https://www.napratica.org.br/linguagem-deprogramacao/. Acesso em: 4 mar. 2023.

ORACLE. **Java.** Java. Brasil: Oracle, 2022. Disponível em: https://www.java.com/pt-BR/download/help/whatis_java.html. Acesso em: 4 mar. 2023.

DIEGO MELO. **TecnoBlog.** O que é Java. Brasil: TecnoBlog, 2021. Disponível em: https://tecnoblog.net/responde/o-que-e-java-guia-para-iniciantes/. Acesso em: 4 mar. 2023.

REDHAT. **RedHat.** Arquitetura Orientada a Serviços. Brasil: RedHat, 2020. Disponível em: https://www.redhat.com/pt-br/topics/cloud-native-apps/what-isservice-oriented-architecture. Acesso em: 4 mar. 2023.

GABRIEL SACRAMENTO. **Rock Content.** Aplicação Web. Brasil: Rock Content, 2022. Disponível em: https://rockcontent.com/br/talent-blog/aplicacaoweb/. Acesso em: 4 mar. 2023.

IVAN DE SOUZA. **Rock Content.** Protocolo HTTP. Brasil: Rock Content, 2019. Disponível em: https://rockcontent.com/br/blog/http/. Acesso em: 4 mar. 2023.

MDN WEB DOCS. **developer mozilla.** Verbos HTTP. : Mozilla, 2022. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods. Acesso em: 4 mar. 2023.

JETBRAINS. **JetBrains.** IntelliJ. Brasil: JetBrains, 2023. Disponível em: https://www.jetbrains.com/pt-br/idea/features/. Acesso em: 4 mar. 2023.

PABLO HENRIQUE AGUIAR CAVALCANTE. **Geek Hunter.** Spring Framework. Brasil: Geek Hunter, 2021. Disponível em: https://blog.geekhunter.com.br/springframework/#:~:text=9%20Conclus%C3%A3o-,O%20que%20%C3%A9%20o%20Spring%20Framework%3F,um%20boost%20na%20aplica%C3%A7%C3%A3o%20Java. Acesso em: 4 mar. 2023.

BÁRBARA ROSSALI. **Zup.** SpringBoot. Brasil: Zup, 2021. Disponível em: https://www.zup.com.br/blog/springboot#:~:text=O%20Spring%20Boot%20%C3%A9%20um,gasto%20com%20as %20configura%C3%A7%C3%B5es%20iniciais. Acesso em: 4 mar. 2023.

NIMBLE. **NimbleWork.** Testes Unitários. Brasil: Nimble, 2023. Disponível em: https://www.nimblework.com/pt-br/agile/testes-unitarios/. Acesso em: 4 mar. 2023.

MANOEL. **DevMedia.** JUnit. Brasil: DevMedia, 2009. Disponível em: https://www.devmedia.com.br/junit-tutorial/1432. Acesso em: 4 mar. 2023.

AMAZON AWS. **Amazon Aws.** API. Brasil: Amazon, 2023. Disponível em: https://aws.amazon.com/pt/what-is/api/. Acesso em: 4 mar. 2023.

ADMINGR1D. **gr1d.** Swagger. Brasil: gr1d, 2022. Disponível em: https://gr1d.io/2022/04/15/swagger/. Acesso em: 4 mar. 2023.

ISADORA GIONGO. **Ilegra.** SonarLint. Brasil: Ilegra, 2023. Disponível em: https://ilegra.com/blog/do-zero-sonarlint-para-que-serve-esse-plugin-e-porque-voce-nao-vai-mais-viver-sem-ele/#:~:text=O%20Sonarlint%20%C3%A9%20um%20detector,%2C%20Major%2C%20Critical%20e%20Blocker. Acesso em: 4 mar. 2023.

RODRIGO. **DevMedia.** Maven. Brasil: DevMedia, 2013. Disponível em: https://www.devmedia.com.br/introducao-ao-maven/25128#1. Acesso em: 4 mar. 2023.

SAFETEC INFORMATICA. **Safetec.** Google Cloud Platform. Brasil: Safetec, 2022. Disponível em: https://safetec.com.br/cloud-computing/oque-e-google-cloud-platform/. Acesso em: 4 mar. 2023.

RAFAEL MARQUES. **Homehost.** HTML. Brasil: Homehost, 2019. Disponível em: https://www.homehost.com.br/blog/tutoriais/o-que-e-html/. Acesso em: 4 mar. 2023.

ARIANE G.. **Hostinger.** CSS. Brasil: Hostinger, 2022. Disponível em: https://www.hostinger.com.br/tutoriais/o-que-e-css-guia-basico-de-css. Acesso em: 4 mar. 2023.

UGO ROVEDA. **Kenzie.** JavaScript. Brasil: Kenzie, 2021. Disponível em: https://kenzie.com.br/blog/javascript/. Acesso em: 4 mar. 2023.

MARYLENE GUEDES. **Treina Web.** Angular. Brasil: Treina Web, 2020. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-o-angular-epara-que-serve. Acesso em: 4 mar. 2023.

TECNOBLOG. **TecnoBlog.** Bootstrap. Brasil: TecnoBlog, 2020. Disponível em: https://tecnoblog.net/responde/o-que-e-bootstrap/#h-definindo-obootstrap. Acesso em: 4 mar. 2023.

LUCIDCHART. **LucidChart.** Diagrama de Classes. Brasil: LucidChart, 2023. Disponível em: https://www.lucidchart.com/pages/pt/o-que-ediagrama-de-classe-uml. Acesso em: 4 mar. 2023.

LEANDRO. **DevMedia.** Diagrama de Caso de Uso. Brasil: DevMedia, 2012. Disponível em: https://www.devmedia.com.br/o-que-e-uml-e-diagramas-decaso-de-uso-introducao-pratica-a-uml/23408. Acesso em: 4 mar. 2023.

METODOLOGIA CIENTIFICA. **Metodologia Cientifica.** Metodologia. Brasil: Metodologia Cientifica, 2023. Disponível em: https://www.metodologiacientifica.org/. Acesso em: 9 mar. 2023.

ISABELLA MORETTI. **Via Carreira.** PESQUISA APLICADA. *[S.l.].* Via Carreira, 2021. Disponível em: https://viacarreira.com/pesquisa-aplicada/. Acesso em: 9 mar. 2023.