PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Arquitetura de Software Distribuído

Bruno Eustáquio Enes Cardoso
Michael Inácio
Michael Pacheco
Raphael Garnica
Victor Quaresma

SISTEMA PARA CONVITE DE CASAMENTOS ONLINE

Belo Horizonte
2022

Bruno Eustáquio Enes Cardoso

Michael Inácio

Michael Pacheco

Raphael Garnica

Victor Quaresma

SISTEMA PARA CONVITE DE CASAMENTOS ONLINE

Projeto Integrado de Arquitetura de Software Distribuído como requisito parcial à obtenção do título de especialista.

Belo Horizonte

2022

RESUMO

Com a popularização do uso da internet e seus sistemas on-line, muitos usuários têm mudado seus costumes e hábitos. Como resultado do avanço tecnológico, hoje temos acesso quase ilimitado à internet, principalmente por meio de smartphones e tablets. Após a pandemia de Covid-19, esta revolução tornou-se ainda mais visível e experimentou uma aceleração significativa. Muitas pessoas se acostumaram a resolver tudo pela internet, de forma prática e instantânea. Em um casamento tradicional, por exemplo, o casal normalmente convida seus parentes e amigos mais próximos, mas muitos não poderão comparecer devido à distância e ao custo da viagem do casal. Como resultado, foi investigada a possibilidade de desenvolver um sistema que facilite o planejamento e gestão de casamentos no meio virtual, com o objetivo de reduzir custos e trazer mais comodidade aos usuários.

Palavras-chave: software, casamento, convite, microsservicos, backend, frontend.

SUMÁRIO

INTRODUÇÃO

No mundo globalizado, a internet assume um papel cada vez mais importante no dia-a-dia, seja para usuários comuns, pequenas empresas ou até mesmo grandes corporações. As pessoas ganharam mais liberdade para expressar suas opiniões, compartilhar informações e conteúdos diversos, criando assim a possibilidade de formação de vínculos entre usuários interessados no mesmo tema. Estas possibilidades têm proporcionado inovação e interatividade, sendo muitas vezes utilizadas como um tipo de auxílio direto.

Pesquisa realizada pela Agência Brasil (2020), divulgada na segunda-feira, 23 de agosto de 2021, aponta que, em 2022 o Brasil possuía mais de 152 milhões de usuários conectados à internet. O número era 7% maior do que no ano anterior. O estudo abrangeu pessoas com acesso à internet em qualquer ambiente (domicílios, ambiente de trabalho, lan houses, escolas, locais públicos e outras localidades).

Com o aumento do uso da internet, alguns sistemas e sites mudaram o foco da maioria das visitas dos usuários à rede, além do uso comum do correio eletrônico (e-mail) e de buscadores como o Google. Um setor que tem se beneficiado do avanço tecnológico é a indústria de eventos, principalmente casamentos.

Segundo Ladeira (2015), os casamentos mesmo os tradicionais não têm conseguido fugir muito dessa onda tecnológica, muitos casais têm aderido tecnologias em suas cerimônias e se beneficiado tanto na economia de tempo, dinheiro, esforço e paciência durante os preparativos.

Com o objetivo de contribuir para uma área que está a ganhar muita atenção na sociedade atual, o presente projeto visa criar uma plataforma que permitirá aos casais que pretendem casar organizar a sua lista de convidados, gerir os seus convidados e ter uma cerimónia de casamento, tudo isso economizando dinheiro e reduzindo o uso de papel. Além disso, um ponto importante seria o uso do *QRCode* nos convites virtuais para evitar filas e permitir que os convidados acessem as dependências do evento por meio de seus smartphones, garantindo um acesso mais rápido.

1. Objetivos do trabalho

Apresentar a arquitetura do sistema Invitation que ajuda casais de noivos a se comunicarem com os seus convidados, permitindo assim o envio de convites e recebimento de presentes por meio da plataforma.

Os objetivos específicos são:

- Explicar a arquitetura do sistema e seu funcionamento, bem como os seus principais componentes;
- Desenvolver o protótipo *frontend* em *ReactJs* e *backend* em *Java*;
- Desenvolver interface responsiva para o sistema;
- Elencar os principais requisitos do sistema;

2. Descrição geral da solução

A solução desenvolvida irá criar a camada de *frontend* que fornecerá uma estrutura de componentes personalizáveis para uma aparência mais moderna e intuitiva ao usuário, além de ser responsivo que facilitará usuários que utilizem PC`s, notebooks, smartphones e tablets, possam ter uma experiência adequada a realidade de seu dispositivo. O *frontend* se comunicará com o *backend* através de *API*s expostas no *backend* em *REST*, utilizando um *gateway*, que permitirá que os cadastros de convidados, listagem, confirmações dos convidados ao casamento, contribuição dos convidados e visualização dos recursos sejam feitos de forma efetiva.

Na camada de *backend*, por sua vez, serão criados vários microsserviços, cada um com sua responsabilidade, os serviços terão a comunicação direta com bancos de dados para armazenamento de dados da aplicação, e poderá ser publicado na nuvem que será responsável por todo controle da máquina memória e *CPU*, além disso neste ambiente será possível efetuar o orquestramento e *load balancer* da aplicação. Essa camada terá a função de armazenar os dados dos noivos, convidados e também o convite de casamento que será importado na ferramenta. Este mesmo serviço se encarregará de comunicar com os diversos gateways disponíveis no mercado de pagamento via cartão de crédito/débito ou transferências bancárias, visando facilitar que convidados ou parentes possam contribuir.

2.1. Apresentação do problema

Tendo em mente o crescente número de usuários que estão utilizando a tecnologia para organizar um casamento, o projeto busca solucionar problemas comuns no planejamento de casamentos, como a sustentabilidade na redução do uso de papel por meios virtuais.

Segundo Olsen (2021), para a produção de uma tonelada de papel, são necessárias 17 árvores e 115 mil litros de água, ou seja, trocar o uso do papel para o meio digital promove a redução do gasto de recursos naturais. Portanto, o sistema tem a proposta de deixar o processo de envio de convite de maneira virtual, os convidados não precisam possuir o convite em mãos para visualizá-lo e nem mesmo levá-lo consigo na cerimônia para confirmar o seu ingresso nas dependências da festa, sem a necessidade que o segurança busque o nome da pessoa na lista. Outra facilidade que a plataforma busca em resolver é a possibilidade do casal realizar tudo em só lugar, bem como receber quantias transferidas por convidados e até mesmo se comunicar com fornecedores que trabalham especificamente para organização da cerimônia.

2.2. Descrição geral do software (Escopo)

O objetivo deste software é fornecer a usuários a possibilidade de cadastrar convidados em uma lista digital para ter acesso presencialmente em uma cerimônia de casamento, permitindo assim a visualização do convite de forma virtual e dinâmica pelos convidados e para ter acesso aos eventos por meio do *QRCode* gerado pela aplicação.

Outra característica importante é a possibilidade dos convidados transferirem fundos pela plataforma ou até mesmo pagarem com cartão de crédito/débito por meio de *gateways* de pagamento integrados ao sistema para envio de valores aos noivos. Vale destacar também que o sistema poderá se integrar com fornecedores especializados em planejamento de casamentos, buffets, músicos e empresas interessadas.

3. Definição conceitual da solução através de Requisitos funcionais

- **Cadastro de convidados:** O sistema permitirá que o usuário possa cadastrar informações do convidado na plataforma, para o mesmo ser notificado para aceitar e visualizar os dados do evento como endereço, data e *QRCode*.

- **Exibição de dashboards:** O sistema permitirá que o usuário visualize através de gráficos, a quantidade de convidados que aceitaram e confirmaram a presença no evento.

- **Importação do convite:** O sistema permitirá que o usuário realize a importação do arquivo contendo o convite do casamento e pré-visualize o mesmo na aplicação, para ser enviada aos convidados.

- **Contribuição com os noivos:** O sistema permitirá que os convidados contribuam com algum valor para ajudar os noivos a mobiliar a sua nova casa ou uma contribuição para a festa, esses valores poderão ser transferidos diretamente do sistema.

4. Modelagem e projeto arquitetural

A Imagem 1 mostra como será a arquitetura geral do sistema, várias tecnologias serão utilizadas proporcionando assim um sistema agnóstico ao provedor de nuvem, podendo ser implementado como por exemplo na *AWS*, *Azure* ou *Google Cloud Provider*.

Também serão utilizados alguns *gateways* de pagamentos de parceiros para que o sistema não precise controlar os vários meios de pagamentos existentes.

É importante salientar que o sistema é escalável de acordo com as configurações realizadas no provedor *cloud* selecionado ou até mesmo em ambientes *on-premise*.

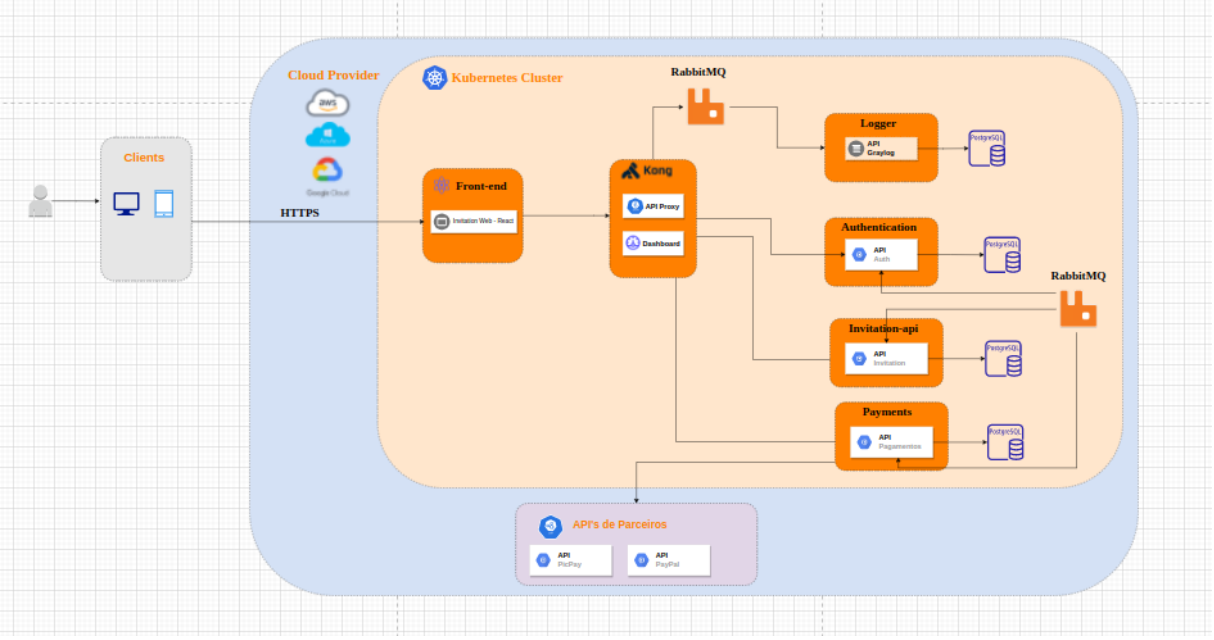

Imagem 1. Arquitetura do sistema.

4.1. *Kubernetes*

Para o controle e gerenciamento dos microsservicos da aplicação, será utilizado o *Kubernetes* que é uma plataforma de código aberto, portável e extensiva para o gerenciamento de cargas de trabalho e serviços distribuídos em contêineres, que facilita tanto a configuração declarativa quanto a automação. Ele possui um ecossistema grande, e de rápido crescimento. O Google tornou *Kubernetes* um projeto de código aberto em 2014. O *Kubernetes* combina mais de 15 anos de experiência do Google executando cargas de trabalho produtivas em escala, com as melhores ideias e práticas da comunidade.

- **Balanceamento de carga:** O *Kubernetes* pode expor um contêiner usando o nome *DNS* ou seu próprio endereço IP. Se o tráfego para um contêiner for alto, o *Kubernetes* pode balancear a carga e distribuir o tráfego de rede para que a implantação seja estável.

- **Orquestração de armazenamento:** O *Kubernetes* permite a construção automática de um sistema de armazenamento, como armazenamentos locais, provedores de nuvem pública entre outros.

- **Lançamentos e reversões automatizadas:** O usuário pode descrever o estado desejado para seus contêineres implantados usando o *Kubernetes*, e ele pode alterar o estado real para o estado desejado em um ritmo controlado. Por exemplo, é possível automatizar o *Kubernetes* para criar novos contêineres para uma implantação, remover os contêineres existentes e transferir todos os recursos para o novo contêiner.

- **Empacotamento binário automático:** É possível fornecer ao *Kubernetes* um cluster de nós que pode ser usado para executar tarefas nos contêineres. O usuário informa ao *Kubernetes* de quanta *CPU* e memória (*RAM*) cada contêiner precisa. O *Kubernetes* pode encaixar contêineres em nós para fazer o melhor uso de recursos.

- **Autocorreção:** O *Kubernetes* reinicia os contêineres que falham, substitui os contêineres, elimina os contêineres que não respondem à verificação de integridade definida pelo usuário e não os anuncia aos clientes até que estejam prontos para servir.

- **Gerenciamento de configuração e de segredos:** O *Kubernetes* permite armazenar e gerenciar informações confidenciais, como senhas, tokens *OAuth* e chaves *SSH*.[1]

[1] KURBENETES. **O que é Kubernetes.** Disponível em: <https://kubernetes.io/pt-br/docs/concepts/overview/what-is-kubernetes/>. Acesso em: 03 de Maio de 2022.

4.2. *Kong*

Já no gerenciamento de requisições nas chamadas nas *APIs* do sistema, utilizaremos o *Kong* que é uma plataforma de *API* escalável e de código aberto (também conhecida como *API Gateway* ou *API Middleware*). O *Kong* foi originalmente desenvolvido pela Kong Inc. (anteriormente conhecida como *Mashape*) para proteger, gerenciar e estender mais de 15.000 microsserviços para seu *API Marketplace*, que gera bilhões de solicitações por mês.[2]

O *gateway* roda na frente de qualquer *API RESTful* e pode ser estendido por meio de módulos e plugins. Ele foi projetado para ser executado em arquiteturas descentralizadas, incluindo implantações de nuvem híbrida e multi-nuvem.
Com a ferramenta é possível:

- Automatizar o fluxo de trabalho e as práticas modernas de *GitOps*.
- Descentralização de aplicativos/serviços em microsserviços.
- Criação de um ecossistema *API`s*.
- Identificação de anomalias e ameaças relacionadas à *API*.

O *Kong Gateway* pode ser executado nativamente no *Kubernetes* com seu controlador de entrada personalizado, gráfico *Helm* e operador. Um controlador de entrada do *Kubernetes* é um proxy que expõe serviços do *Kubernetes* de aplicativos (por exemplo, *Deployments*, *ReplicaSets*) executados em um cluster *Kubernetes* para aplicativos cliente executados fora do cluster.[3]

4.3. Microsserviços e *RabbitMQ*

O sistema será dividido em vários microsserviços, cada um com sua responsabilidade e acesso a sua base de dados, toda a comunicação necessária entre os microsserviços será realizada utilizando *RabbitMQ*.

O *RabbitMQ* é um servidor de mensageria de código aberto desenvolvido em *Erlang*, implementado para suportar mensagens em um protocolo denominado *Advanced Message Queuing Protocol* (AMQP). Ele possibilita lidar com o tráfego de mensagens de forma rápida e confiável, além de ser compatível com diversas linguagens de programação, possuir interface de administração nativa e ser multiplataforma.

---

[2] DOCKER HUB. **Kong.** Disponível em: <https://hub.docker.com/_/kong>. Acesso em: 03 de Maio de 2022.
[3] KONG. **Kong gateway.** Disponível em: <https://docs.konghq.com/gateway/>. Acesso em: 03 de Maio de 2022.

Essa comunicação se faz importante quando precisamos de que alguma confirmação de pagamento seja entregue a outro microsserviço. Foi analisado a utilização de dois tipos de mensageria, *RabbitMQ* e *Kafka*, duas tecnologías bastantes utilizadas em aplicações distribuídas, realizando pesquisas sobre as duas, percebemos que o *RabbitMQ* iria atender as necessidades do projeto, sendo o *Kafka* um pouco mais robusto do que precisa nesse sistema (DE CASTRO ARRUDA, Filipe Jessé; MARTINS, Daves Márcio Silva, 2021).

Para a arquitetura atual, será criado quatro microsserviços, *Logger, Authentication, Invitation-api e payments* descritas abaixo:

- ***Logger*:** Esse microsserviços será responsável por registrar as requisições recebidas pelo *gateway Kong*, as informações serão publicadas em um fila *RabbitMQ* utilizando um *plugin* para comunicação do *kong* com o protocolo *AMQP* e gravadas no banco de dados da *API*;

- ***Authentication:*** *API* responsável por verificar o login e senha do usuário e gerar o token JWT que dará acesso para o usuário aos outros recursos disponíveis. A *API* terá sua própria base de dados para realizar as validações e se comunicará com o *Kong* para receber os dados de login;

- ***Invitation-api:*** *API* responsável pelos recursos gerais do sistema, como convidados, convite, *QRCode*, confirmações pela parte do convidado ao casamento, entre outros. Os endpoints são protegidos e necessita de um *JWT* no cabeçalho das requisições. O *Kong* irá receber as requisições, realizar as validações de segurança pela *API Authentication* e caso sucesso, dar continuidade a requisição para a *API Invitation-api*;

- ***Payments:*** *API* responsável por receber as requisições direcionadas as contribuições dos convidados, nela será feita validações sobre dados do casamento e dos usuários que receberão as contribuições, ou seja, os dados das contas do usuário, caso esteja ok, será realizada uma requisição as *API's* dos parceiros cadastrados responsáveis pelos pagamentos.

5. Prova de Conceito (POC) / protótipo arquitetural

Foi desenvolvida uma POC (*Proof of Concept*) para avaliarmos a criação do projeto em questão. A POC foi dividida em três sistemas, banco de dados utilizando *PostgreSQL*, *frontend* utilizando a biblioteca *React* e o *backend* utilizando a linguagem *Java*. Para os três

sistemas foi utilizado o Docker simulando um ambiente isolado para cada aplicação. O código se encontra no github https://github.com/gitmichaelpap/PI-Invitation.

Na base de dados foram criadas duas tabelas, *user* e *guest*, a primeira irá conter os dados do casamento e dos noivos e também os dados de login para acessar o sistema, já a segunda tabela, *guest*, temos os dados dos convidados do casamento como o nome, data que o mesmo confirmou se irá ao casamento, *QRCode* do convite e o relacionamento com a tabela *user*, para o sistema saber a qual casamento aquele convidado pertence.

Para o *backend* foi desenvolvido uma *API Rest* para que os usuários possam realizar o login, utilizando a autorização *Bearer* e tambem alguns endpoints para ter acesso aos dados do casamento, como lista de convidados, adicionar novos convidados, listar convidados. Todos os endpoints (exceto o de login) precisar ser passado no Header da requisição o token *JWT* gerado pela própria *API* (endpoint de login), caso não seja passado, a *API* retorna o *HTTP* Status 401. Essa abordagem de a mesma *API* realizar o login, gerar o token e ter acesso aos recursos foi utilizada somente para a POC, quando o sistema estiver completo será utilizado uma *API* somente para autenticação.

No front, além da tela de login, que dará acesso aos dados do casamento, teremos uma tela de home, onde temos um resumo de quantidade de convidados e quantidade de confirmações, ainda teremos um menu para cadastrar, editar ou excluir um convidados e um outro menu para mostrar o convite que será enviado ao convidado.

5.1. Implementação

Para implementar essa POC, podemos destacar a utilização das seguintes tecnologias:

- **ReactJS**: Uma biblioteca *JavaScript* para criação de interface de usuário, a escolha se deu pelo conhecimento e facilidade do time de desenvolvedores de *frontend* da equipe.

- **Spring**: Um *framework Java* criado com o objetivo de facilitar o desenvolvimento de aplicações explorando os conceitos de inversão de controle e injeção de dependências, com ele é possível criar uma *API Rest* em poucos segundos utilizando o próprio site https://start.spring.io/. Também foi escolhido pelo conhecimento do time de desenvolvimento backend e pela facilidade na criação de *API's*, mas isso não interfere na utilização de outras linguagens ou *frameworks* quando for desenvolvidos as outras *API's* do sistema como um todo (SILVA, Alice Fernandes, 2019).

- **PostgreSQL:** Um servidor de banco de dados para o armazenamento seguro de informações, desenvolvido no Berkeley Computer Science Department da Universidade da Califórnia. Essa ferramenta tem código aberto, implementa a sintaxe de linguagem *SQL* e roda nos sistemas *Unix*, *Mac OS X, Solaris e Windows*. Foi escolhida por ser de código aberto e pela facilidade de utilizar a mesma com o docker e sua escalabilidade. Também foi pensado se a aplicação iria necessitar de um banco mais completo, como o *Oracle* (banco de dados pago) e decidido que o *PostgreSQL* seria o ideal para a aplicação em um primeiro momento.

- **Docker:** o *Docker* é uma forma de virtualizar aplicações no conceito de "containers", trazendo da *web* ou de seu repositório interno uma imagem completa, incluindo todas as dependências necessárias para executar sua aplicação. A sua escolha foi exatamente pelo seu container todas as dependências necessárias para rodar a aplicação em si. Na POC foi criado um container para o *frontend* (dependências do *ReactJS* entre outras), container para o banco de dados (dependências do *PostgreSQL*) e um container para o backend (dependências do *Spring / Java*) (NETTO, Hylson Vescovi,2018);

- **Docker Compose:** *Docker compose* é um orquestrador de containers da *Docker*, ele irá nos auxiliar a orquestrar os três container que temos na aplicação, *backend*, frontend e banco de dados além de facilitar como vamos subir as nossas aplicações (mais detalhes no tópico 5.3).

5.2. Arquitetura

A imagem 2 ilustra como foi realizado a arquitetura da POC, como explicado anteriormente, as três aplicações. *frontend*, *backend* e banco de dados estão em cada um em seu container, onde o usuário acessa o página *Web* e seus recursos, a aplicação *Web*, que está rodando em um contêiner realiza a comunicação com o container docker onde o *backend* está sendo executado, a aplicação *backend* aplica as regras de negócios e caso precise, acessar a aplicação do banco de dados que está sendo executado em um terceiro container docker.

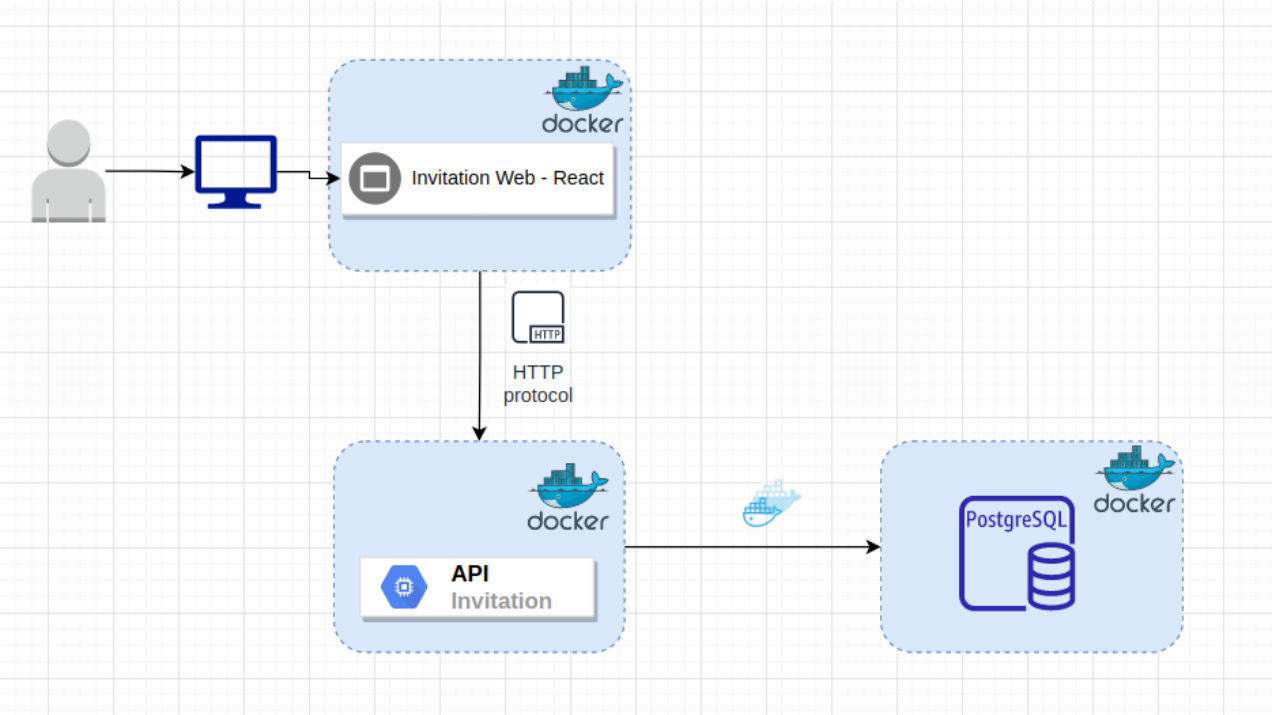

Imagem 2. Arquitetura POC.

5.3. Implantação

Para subirmos as aplicações *backend*, *frontend* e o banco de dados vamos precisar termos em nossas máquinas os seguintes tecnologias:

- *Java* 11 ou superior: https://www.oracle.com/br/java/technologies/javase/jdk11-archive-downloads.html;
- *Maven*: https://maven.apache.org/download.cgi;
- *Docker*: https://docs.docker.com/desktop/windows/install/;
- *Docker Compose*: https://docs.docker.com/compose/install/.

No repositório do *github*, temos duas aplicações, o *frontend* (invitation), e o *backend* (*invitation-api*), cada aplicações contêm um readme.md com as instruções para realizar a implementação local.

Para o *frontend*, na raiz da aplicação existe um arquivo, *docker-compose.yml,* que contêm a imagem que vamos utilizar do *NodeJS*, 14, executando o comando *docker-compose up -d*, o docker irá criar um container com a dependência do *NodeJS* e executar a aplicação que poderá ser acessada pela url http://localhost:3000/ .

Já no *backend*, onde temos tanto a *API* quando o banco de dados, primeiramente na raiz do projeto executamos o comando *mvn clean install*, no qual irá gerar o .jar da aplicação

e também irá gerar uma imagem docker da nossa aplicação, essa imagem é gerada utilizando o *plugin com.spotify.dockerfile-maven-plugin* fornecido pela repositório maven, o qual o Spring está utilizando para obter suas dependências.

O *swagger* da *API* poderá ser acessado no link http://<host:porta>/swagger-ui.html. A mesma se encontra publicada no *Heroku*, https://api-invitation-pi.herokuapp.com/ .

No repositório *git*, https://github.com/gitmichaelpap/PI-Invitation, existem duas *branches*, *develop*, onde tanto a api quanto o *frontend* podem ser executados localmente, como exemplificado acima utilizando os containers *docker*.

Já a *branch heroku* está com os dados do *heroku*, onde a *API* foi publicada e o banco de dados já está sendo executado, para o código da *API*, existe um *docker-compose*, caminho: *docker/api/*, onde somente será executado a própria *API* apontando para o banco de dados que está sendo executado no *heroku*. Para o *frontend*, quando executado na *branch feature/heroku*, o mesmo já está realizando suas requisições para a *API* publicada no *heroku*.

O *frontend* também foi publicado utilizando a plataforma *heroku* para fins de testes da POC desenvolvida, url da publicação: https://ui-invitation-pi.herokuapp.com/.

5.4. Casos de Uso

Na POC desenvolvida foram contemplados o desenvolvimento do frontend, utilizando ReactJS, um microserviço, *API Rest* utilizando *Spring/Java* com uma documentação de seus endpoints utilizando o *Swagger*, a persistência de dados ficou por conta do *PostgreSQL*, além da utilização do *docker* para subirmos cada aplicação em seu container separadamente.

Quando o backend é iniciado, o mesmo faz a utilização de migrations, a qual na POC irá criar as tabelas user e guest e adicionar um novo usuário, login admin@gmail.com. e senha: **admin**. Além dos casos de uso:

- **Login:** O login e a senha do usuário é verificada na base de dados, para essa verificação, é enviado os dados para a *API* e assim realizado a verificação, a senha comparada utilizando a biblioteca *BCrypt*, logo, na base de dados as senhas são criptografadas. Caso o usuário e senha estejam corretos, é retornado para o *frontend*, os dados do usuário logado e o *token JWT* gerado pela biblioteca *Jsonwebtoken* com uma chave secreta previamente cadastrada nas variáveis de ambiente do sistema;

- **CRUD Convidados:** Foi criado os recursos para cadastrar, deletar, editar e listar os convidados de acordo com o usuário logado, assim o usuário não conseguirá ver os

convidados de outro casamentos/usuário. Todos os recursos criados foram utilizando *Rest*(cadastrados na documentação da *API*, *swagger*) e protegidos, assim, se faz necessário para toda a requisição desses recursos passar no cabeçalho da mesma, um token *JWT* para validação, essa validação ocorre utilizando a chave secreta cadastrada nas variáveis de ambiente do sistema, a mesma utilizada para gerar o token no login;

- **Acesso convite:** Esse recurso permite ao usuário ter acesso ao convite que será enviado para os convidados cadastrados na sua lista de convidados;

- **Novo usuário:** Recurso para criação de um novo usuário com os dados do casamento e com isso, ter acesso ao sistema e adicionar os seus convidados.

6. Architecture Decision Records (ADRs)

Title	Modificação linguagem desenvolvimento *backend* em *NodeJS*.
Status	Aceito
Context	Foi decidido em conjunto utilizar como linguagem de desenvolvimento na camada de *backend* o *NodeJS* ao invés do *DotNet*, por ser uma plataforma robusta e muito utilizada atualmente por grandes empresas. Um ponto importante a ser ressaltado é que a indicação da linguagem se deve por desenvolvedores no projeto possuírem experiência prévia na linguagem, o que facilitará no desenvolvimento do projeto com mais agilidade.
Decision	Utilização do *NodeJS* como camada *backend* para comunicações via *API* com *frontend* em *ReactJS*.

Consequences	● Agilidade no desenvolvimento. ● Facilidade na integração com o *frontend* em *ReactJS*.

Title	Criar microsserviço em *Java* POC
Status	Aceito
Context	Foi decidido em conjunto utilizar na camada de *backend* a arquitetura de microsserviço em *Java*, por ser muito utilizado atualmente e pela facilidade em subir em uma aplicação em nuvem.
Decision	Utilização java como camada *backend* para comunicações via *API* com *frontend* em React.
Consequences	● Agilidade no desenvolvimento. ● Facilidade em fazer deploy na nuvem. ● Facilidade na integração com o *frontend* via *APIs* em *REST*.

Title	Utilização de biblioteca *MUI* no *ReactJS*
Status	Aceito
Context	Foi decidido em conjunto utilizar na camada de *frontend,* a biblioteca *MUI*, que fornece uma biblioteca robusta, personalizável e acessível de componentes básicos e avançados, permitindo que o desenvolvedor

	construa o design mais rapidamente.
Decision	A biblioteca possui uma documentação muito bem explicativa o que facilitou a adoção da mesma no projeto, e por possuir muitos componentes muito comuns de serem utilizados na página *Web*, além de deixar por padrão uma página responsiva.
Consequences	● Agilidade no desenvolvimento de componentes no *React*. ● Design responsivo. ● Documentação de fácil entendimento.

7. RESULTADOS

Nessa arquitetura foi possível observar a utilização de microsserviços, o que é importante para separamos responsabilidades, conseguimos criar um sistema que seja agnóstico a onde será implementado, não ficando dependente a um provedor de nuvem, o que não impeça que seja utilizado algum serviço que os mesmos forneçam.

Algumas melhorias poderão ser realizadas para que o sistema seja mais completo, como a melhor utilização dos *logs* cadastrados, cadastro de novos logs, oriundos de todos os microsserviços existentes. Na parte do login, poderá ser realizada uma implementação para que possa ser utilizado o login do Google, facilitaria o cadastro dos dados do usuário e também a implementação da autenticação com *MFA*, dando uma maior segurança ao sistema.

CONCLUSÃO

Com essa documentação foi possível concluir a importância de uma documentação quando formos iniciar um novo projeto, a mesma nos auxiliou a tomar as melhores decisões em relação a escolha de tecnologias a serem utilizadas e a melhor maneira de implementar a POC que desse uma visão geral do que seria o sistema.

Os objetivos de criar uma arquitetura inicial, escolher as linguagens e *frameworks* que serão utilizados foram atendidos, além de podermos conhecer novas tecnologias através dos estudos feitos para atender os requisitos do sistema.

A principal dificuldade encontrada no decorrer do projeto foi a escolha das tecnologias que seriam utilizadas, pois cada desenvolvedor do time tinha um conhecimento maior em linguagens diferentes, assim a divisão em microsserviços foi bastante útil, cada serviço poderia ter uma linguagem de programação.

Outro ponto de decisão importante foi em qual provedor de *cloud* o sistema iria ficar hospedado devido ao conhecimento diversificado do time e com isso a decisão foi criar uma arquitetura que fosse agnóstica ao provedor.

REFERÊNCIAS

1 KURBENETES. **O que é Kubernetes.** Disponível em: <https://kubernetes.io/pt-br/docs/concepts/overview/what-is-kubernetes/>. Acesso em: 03 de Mai. 2022.

2 DOCKER HUB. **Kong.** Disponível em: <https://hub.docker.com/_/kong>. Acesso em: 03 de Mai. 2022.

3 KONG. **Kong gateway.** Disponível em: <https://docs.konghq.com/gateway/>. Acesso em: 03 de Mai. 2022.

AGENCIA BRASIL. **Brasil tem 152 milhões de pessoas com acesso à internet**. Disponível em:<https://agenciabrasil.ebc.com.br/geral/noticia/2021-08/brasil-tem-152-milhoes-de-pessoas-com-acesso-internet#:~:text=Pesquisa%20promovida%20pelo%20Comit%C3%AA%20Gestor,anos%20t%C3%AAm%20internet%20em%20casa>. Acesso em: 06 de Abr. 2022.

LADEIRA, Kamilla. **Tire proveito da tecnologia no casamento**. 2015. Disponível em:<http://aboutlove.com.br/blog/tecnologia-nos-casamentos>. Acesso em: 06 de Abr. 2022.

OLSEN, Natasha. **Reduzir o uso do papel poupa tempo, dinheiro e recursos naturais.** 2021**.** Disponível em: <https://ciclovivo.com.br/vida-sustentavel/minimalismo/reduzir-papel-poupa-tempo-dinheiro-recursos-naturais/>. Acesso em: 06 de Abr. 2022.

NETTO, Hylson Vescovi. **VIRTUALIZAÇÃO EM NÍVEL DE SISTEMA USANDO O DOCKER**. Anais da Mostra de Ensino, Pesquisa, Extensão e Cidadania (MEPEC), v. 3, p. 120-120, 2018.

SILVA, Alice Fernandes et al. **Gerador de código para uma API REST com base no framework Spring Boot**. 2019.

DE CASTRO ARRUDA, Filipe Jessé; MARTINS, Daves Márcio Silva. **Análise Comparativa entre sistemas de mensageria: Apache Kafka vs RabbitMQ**. Seminários de Trabalho de Conclusão de Curso do Bacharelado em Sistemas de Informação, v. 6, n. 1, 2021.

APÊNDICES

Repositório *GitHub*: https://github.com/gitmichaelpap/PI-Invitation.

IU invitation Hekoku: https://ui-invitation-pi.herokuapp.com/.

API Invitatio: https://api-invitation-pi.herokuapp.com/.

***API* Invitation *Swagger*:** https://api-invitation-pi.herokuapp.com/swagger-ui/index.html#/