**PROPOSTA DE PROJETO**

**DESAFIO 1**

**NOVA ARQUITETURA**

**ESCOPO**

A segunda fase do projeto contempla uma arquitetura totalmente nova, construída do zero e considerando o uso de tecnologias mais adequadas ao contexto da empresa e aos objetivos de negócio a longo prazo.

**INFORMAÇÕES ADICIONAIS**

Para que seja possível implantar uma solução de longo prazo que atenda a todos os objetivos estratégicos da empresa, é necessário que algumas informações adicionais sejam fornecidas. Portanto, todas as propostas que seguem neste documento foram elaboradas de maneira genérica, podendo ser alteradas de acordo com informações adicionais trazidas pela equipe de negócio. Assim, seguem abaixo algumas questões iniciais que precisam de esclarecimento para o início do desenvolvimento do projeto:

1. Considerando a informação de que os acionistas preveem um aumento de 300% na base de usuários, qual o perfil desses 3 milhões de usuários que devem entrar na plataforma nos próximos meses?
2. Qual a real intenção em internacionalizar a API? Há uma ideia de expandir os negócios para clientes de fora do país?
3. Existe algum prazo previamente acordado para a implantação desta nova versão dos serviços?
4. É possível utilizar algum outro formato de integração para a realização de TED/DOC que não dependa de um servidor físico lendo e escrevendo arquivos em pastas?
5. Qual a frequência de uso dos relatórios existentes no sistema? E qual a necessidade de ter dados em real-time nesses relatórios?

**PROPOSTAS**

1. **ARQUITETURA E INFRAESTRUTURA**

Considerando os objetivos de longo prazo da empresa, sugere-se uma arquitetura de microsserviços distribuídos, baseada em eventos. Dessa forma, é possível que cada serviço seja responsável por apenas uma parte da aplicação, com baixo acoplamento, e permitindo que seja desenvolvido, implantado e mantido de forma independente dos demais. A comunicação entre esses serviços pode se dar através de requisições de API REST ou de serviços de Message Broker, facilitando o desenvolvimento de soluções assíncronas.

Uma imagem contendo estacionamento, medidor, ao ar livre, rua

Descrição gerada automaticamente

A infraestrutura a ser utilizada deverá tirar o melhor de uma arquitetura *cloud native*, considerando a escolha da plataforma AWS. Assim, serão utilizados os seguintes serviços:

- AWS RDS (<https://aws.amazon.com/pt/rds/>): Banco de dados relacional totalmente gerenciável e escalável, utilizando a implementação de PostgreSQL.

- AWS Redshift (<https://aws.amazon.com/pt/redshift/>): Data warehouse, onde serão armazenados os dados históricos para execução de relatórios e ferramentas de BI.

- AWS ElasticSearch Service (<https://aws.amazon.com/pt/elasticsearch-service/the-elk-stack/>): Serviço gerenciável com toda a Elastic Stack (ElasticSearch, Logstash, Beats e Kibana), utilizado para armazenamento de logs, monitoramento de serviços e aplicações, alertas e extração de informações de inteligência de negócio.

- AWS EKS (<https://aws.amazon.com/pt/eks/>): Permite o uso de Kubernetes para automação de implantação e escalabilidade de containeres, que será utilizado em conjunto com o Docker.

- AWS MQ (<https://aws.amazon.com/pt/amazon-mq/>): Serviço de mensagens compatível com RabbitMQ totalmente gerenciável e escalável, permitindo a comunicação entre os microsserviços de modo assíncrono. É possível configurar inúmeras regras para a entrega de mensagens, garantindo que todas as necessidades de comunicação sejam atendidas.

Inicialmente, os *endpoints* públicos podem ser divididos entre 3 serviços, sendo eles:

- Autenticação: Métodos de criação, atualização e busca de dados do usuário, login e troca de senha. Devem seguir o padrão JWT com OAuth2. Posteriormente, de acordo com a necessidade, pode ser desenvolvido um método de autenticação em 2 fatores usando um aplicativo, e-mail ou SMS.

- Transações: Transferências, saques e depósitos, agendamentos e folha de pagamento.

Além destes, o backend deverá ter outros serviços que auxiliarão na execução dos serviços principais:

- Tokenizer: Serviço para criptografia de dados do usuário, para que estes não sejam armazenados de forma aberta, visando uma melhor segurança e futura certificação PCI Compliance.

- Processamento de serviços em lote: Folhas de pagamento, transações agendadas e TED/DOC recebidos, executados através de agendamentos de acordo com a necessidade e as obrigações legais da empresa.

Cada serviço será de responsabilidade de um *squad*, que deverá ter a responsabilidade por todo o ciclo de vida do serviço, desde o desenvolvimento até a implantação e manutenção. Neste processo, as equipes terão certo grau de liberdade para sugerirem as tecnologias que podem ser utilizadas no desenvolvimento. Porém, inicialmente, sugere-se o uso da linguagem Python para os serviços de *backend* e o framework ReactJS para o *frontend* da aplicação.

Todos os desenvolvedores deverão seguir boas práticas de desenvolvimento e arquitetura de código. O “*Clean Code*” deve ser utilizado como uma base para o desenvolvimento. O *code review* será realizado pelo arquiteto, e sempre que necessário devem ser feitos comentários sugerindo melhorias de código e correções antes que o código seja integrado ao repositório principal. Considerando os requisitos da aplicação, as aplicações deverão seguir uma arquitetura hexagonal seguindo os princípios *SOLID*.

1. **PROCESSOS**

Para obter melhores resultados com a nova arquitetura, é necessário também fazer alterações nos processos de desenvolvimento, implantação e manutenção dos sistemas. A cultura de DevOps é uma boa alternativa para tal, trazendo a integração entre as equipes de desenvolvimento e operação/infraestrutura com o objetivo de entregar software de maior qualidade, com menos riscos e maior alinhamento com o negócio.

* 1. Gerenciamento de código-fonte

Para a gestão e versionamento do código fonte, deverá ser utilizada a plataforma Bitbucket (<https://bitbucket.org/product/>). Por meio desta, é possível gerenciar as versões, criar páginas de documentação, criar e revisar *pull requests* e gerenciar pipelines de integração contínua. O modelo de branches a ser seguido deverá ser o GitFlow.

* 1. Testes unitários

Todos os serviços deverão possuir uma boa cobertura de testes unitários. Estes testes deverão ser criados juntamente de cada tarefa de desenvolvimento pelo próprio engenheiro responsável. A cada *pull*, os testes serão automaticamente executados e o repositório somente permitirá o *commit* se os testes passarem com sucesso. Visto que é uma arquitetura totalmente nova, se espera uma cobertura de, no mínimo, 85% de código.

* 1. CI / CD

Considerando a alta cobertura de testes unitários e o modelo de branches utilizado, o próximo passo é a configuração de pipelines de integração / entrega contínua. Para tal, será utilizada uma instância do Jenkins (<https://www.jenkins.io>) integrada ao BitBucket. Com isso, é possível configurar gatilhos de deploy a cada *pull* nas branchs de desenvolvimento ou produção, com pipelines específicos para cada ambiente.

1. **MONITORAÇÃO**

Em um ambiente de alta complexidade e criticidade, é importante que todos os serviços sejam monitorados de forma efetiva, desde a saúde dos servidores até os erros de navegador que o cliente encontrou. Para a monitoração do ambiente, será utilizada a Elastic Stack (<https://www.elastic.co/pt/elastic-stack>), que possui uma ampla gama de funcionalidades que se integram de forma transparente nas aplicações de cada ambiente, permitindo a configuração de alertas, integração com ferramentas de chat e extração de inúmeras informações sobre o ecossistema da empresa de forma inteligente. Todos estes dados são extraídos, processados de forma assíncrona, armazenados em uma instância de ElasticSearch e posteriormente exibidos em um painel de controle intuitivo com diversas visualizações possíveis chamado Kibana.

* 1. Logs

O componente Logstash é um dos pontos de entrada de dados de monitoração das aplicações. Por meio deste, todos os serviços podem gravar logs de forma simples, e estes serão processados e enviados para uma instância central do ElasticSearch que armazenará os logs para posterior visualização e análise. O Logstash possui diversas integrações, podendo ser instrumentado desde o próprio código da aplicação até os logs de containeres.

* 1. APM

A Elastic Stack possui inúmeros agentes de APM (application performance management) para várias linguagens de programação. Estes agentes acompanham as métricas de cada aplicação em seu uso, desde o browser do cliente até a execução de queries no banco de dados pelos serviços. Todo o consumo de recursos, tempos e respostas do servidor são armazenados, enviados para o ElasticSearch para posterior análise através do painel do Kibana.

* 1. Serviços

Também há integrações com grande parte dos serviços de nuvem que serão utilizados, como RDS, SQS, entre outros. Essa integração se dá através de plugins que podem ser instalados e configurados diretamente no serviço de monitoração, sem impacto nas aplicações em execução. Caso algum outro serviço venha a ser integrado e este não possua plugins disponíveis, ainda é possível realizar a integração através de chamadas HTTP que enviem dados em um formato JSON pré-definido.

* 1. Alertas

É possível configurar gatilhos que enviem alertas para os usuários responsáveis na aplicação. Estes alertas podem ser configurados de acordo com a criticidade, serviço a serviço, de modo bem granular. E os envios podem ser através de e-mails (sugerido para alertas menos críticos) e também através do Slack (<https://slack.com>), que é uma das ferramentas de chat mais utilizadas atualmente no mercado. Por meio deste, pode-se configurar canais específicos com determinados grupos de usuários que irão receber cada alerta, inclusive disparando notificações nos dispositivos móveis dos colaboradores que estiverem em plantão em casos de extrema urgência.

1. **MÉTRICAS**

Além de todas as métricas de saúde e consumo da aplicação elencadas na seção de Monitoração, também são necessárias métricas de negócio para uma melhor compreensão do ecossistema. Para tal, também deverá ser utilizada a Elastic Stack, onde serão criados dashboards de visualização específicos voltados para a área de negócio. Inicialmente, sugere-se as seguintes métricas:

1. Usuários ativos (total do mês, por dia do mês, por dia da semana, por horário, quantidade de ações executadas na plataforma)
2. Transações executadas (tipo de transação, tempo de execução, taxa de erros, custo e receita)
3. Acessos simultâneos por período
4. Movimentação de caixa
   1. Por tipo de transação (analisar transações que dão mais retorno e dedicar mais investimentos)
   2. Por usuário (categorizar usuários e oferecer serviços de acordo com as necessidades e retorno previsto)
5. Erros de software (respostas dos serviços e falhas no frontend)
6. Erros de infraestrutura (falhas de comunicação e downtime)
7. Utilização de recursos e custo do ambiente na AWS