

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
PUC Minas Virtual
Pós-graduação *Lato Sensu* em Arquitetura de *Software* Distribuído

Projeto Integrado
Relatório Técnico
Software de Gestão – Módulo de Gestão Financeira

José Ricardo Serathiuk da Silveira

Belo Horizonte
outubro/2022

Projeto Integrado – Arquitetura de Software Distribuído

Sumário

Projeto Integrado – Arquitetura de Software Distribuído	2
1. Introdução	3
2. Cronograma do Trabalho	5
3. Especificação Arquitetural da solução	6
3.1. Restrições Arquiteturais	6
3.2. Requisitos Funcionais	6
3.3. Requisitos Não-funcionais	8
3.4. Mecanismos Arquiteturais	9
4. Modelagem Arquitetural	10
4.1 Diagrama de Contexto	10
4.2 Diagrama de Container	12
4.3 Diagrama de Componentes	14
5. Prova de Conceito (PoC)	17
5.1. Integrações entre Componentes	17
5.2. Código da Aplicação	25
6. Avaliação de Arquitetura (ATAM)	29
6.1. Análises das abordagens arquiteturais	29
6.2. Cenários	29
6.3. Evidências da Avaliação	30
6.4. Resultados Obtidos	40
7. Avaliação Crítica dos Resultados	41
8. Conclusão	42
Referências	43

1. Introdução

O objetivo geral deste projeto é apresentar a descrição do projeto de arquitetura distribuída para um software de gestão (ERP). Essa solução será baseada em um software existente que utiliza uma arquitetura monolítica e vendido no modelo de SaaS (Software as a Service). Esse projeto irá propor um protótipo e versão inicial de um módulo de um sistema ERP utilizando uma arquitetura distribuída. Foi escolhido o módulo de Gestão Financeira, por ser um dos núcleos da gestão das empresas clientes.

Como esse projeto esperasse validar uma arquitetura para a modernização de um software ERP que é vendido como serviço e está no mercado há mais de 15 anos, atendendo os segmentos de mineradoras, concreteiras e construção pesada. Esse ERP monolito foi desenvolvido pensando em uma infraestrutura dedicada de servidores, não se aproveitando de todas as vantagens que Cloud Computing trouxe por exemplo. E a empresa que o desenvolveu está crescendo na faixa de 30% ao ano, e com isso cada vez mais está se tornando um problema escalar comercialmente o ERP atual e em manter o software e mudanças como um todo.

Uma das necessidades atuais é começar a separar a equipe de desenvolvimento em times por módulo e contexto de negócios, para especializar cada vez o desenvolvimento. Citando a Lei de Conway (WIKIPEDIA, 2022) que diz que “organizações que desenvolvem sistemas tendem a produzir sistemas que são cópias de suas estruturas de comunicação”, entendemos que essa mudança cultural dentro da empresa irá afetar a forma com que o software será desenvolvido. Cada time de cada módulo deverá ter que ter certa autonomia e independência, para que atenda de melhor forma o contexto de negócio que está inserido.

Para isso, se espera-se começar a utilizar uma arquitetura distribuída, para lidar com a questão organizacional, e como consequência melhorar a qualidade do software como um todo, desde o desenvolvimento até a sua entrega e a sua melhor adaptação a plataformas de cloud computing.

Foi escolhido apenas 1 módulo, e de forma simplificada para validar a arquitetura inicial, pois hoje o ERP monolito possui 15 módulos, cada um com a média de 10-15 telas. Não seria viável para fim de validação neste trabalho desenvolver uma solução completa de ERP. Será utilizado o Strangler Fig pattern (FOWLER, 2004) para fazer uma migração gradual do monolito para um modelo distribuído.

O módulo escolhido, o de Gestão de Finanças, deverá possibilitar a empresa controlar as finanças, de forma que ele consiga consolidar várias contas bancárias em um único local, possibilitando um melhor acompanhamento de sua saúde financeira e conferir o fluxo de caixa dos gastos. Também irá possibilitar controlar as Contas a Pagar e Receber. O usuário poderá acompanhar e atualizar suas informações financeiras via navegador de internet e via smartphone. Também possibilitará a integração com o ERP monolito (legado) e com os futuros módulos a serem migrados ou desenvolvidos. Por consequência também será migrado o módulo de autorização e autenticação, que será escolhida uma solução de mercado para tal tarefa.

Os objetivos específicos propostos são:

1. Definir e testar um padrão de autenticação e autorização unificado para ser utilizado por todos os serviços;
2. Validar a estrutura e estratégias de comunicação entre os serviços e possíveis problemas que se possa ter com eles, como por exemplo, definição de estratégias para comunicação, transações distribuídas, isolamento e consistência de dados entre serviços;
3. Analisar as complexidades da arquitetura distribuída e seus componentes e verificar se ela é uma alternativa viável para a arquitetura monolítica atual;

2. Cronograma do Trabalho

A seguir é apresentado o cronograma proposto para as etapas deste trabalho.

Datas		Atividade / Tarefa	Produto / Resultado
De	Até		
08/02/2022	09/02/2022	1. Definição do Cronograma do Trabalho	Relatório / Cronograma do Trabalho
10/02/2022	14/02/2022	2. Elaboração da Introdução do Relatório	Relatório / Introdução
15/02/2022	16/02/2022	3. Definição das Restrições Arquiteturais	Relatório / Restrições Arquiteturais
17/02/2022	18/02/2022	4. Definição dos Requisitos Funcionais	Relatório / Requisitos Funcionais
19/02/2022	21/02/2022	5. Definição dos Requisitos Não Funcionais	Relatório / Requisitos Não Funcionais
22/02/2022	23/02/2022	6. Mecanismos Arquiteturais	Relatório / Mecanismos Arquiteturais
23/02/2022	23/02/2022	7. Diagrama de Contexto	Relatório / Diagrama de Contexto
24/02/2022	25/03/2022	8. Revisão do Conteúdo da Etapa 1	Resultado: Melhoria do texto/Conteúdo
26/03/2022	26/03/2022	9. Produção do Vídeo da Etapa 1	Resultado: Vídeo da Etapa 1
28/03/2022	28/03/2022	10. Entrega do Relatório Parcial Etapa 1	Resultado: Relatório Parcial da Etapa 1
29/03/2022	29/03/2022	11. Diagrama de Container	Relatório / Diagrama de Container
30/03/2022	30/03/2022	12. Diagrama de Componentes	Relatório / Diagrama de Componentes
31/03/2022	06/04/2022	14. Wireframes da POC.	Wireframes da POC.
07/04/2022	10/05/2022	15. Código Fonte da POC	Serviço POC: POC funcional
11/05/2022	12/05/2022	16. Vídeo de Apresentação da POC	Resultado: Vídeo de Apresentação da POC
13/05/2022	13/05/2022	17. Entrega do Conteúdo da Etapa 2	Resultado: Entrega do Conteúdo da Etapa 2
15/05/2022	21/05/2022	18. Análise de Abordagens Arquiteturais	Relatório / Análise de Abordagens Arquiteturais
23/05/2022	28/05/2022	19. Cenários	Relatório / Cenários
30/05/2022	04/06/2022	20. Evidências da Avaliação	Relatório / Evidências da Avaliação
06/06/2022	13/06/2022	21. Resultados Obtidos	Relatório / Resultados Obtidos
14/06/2022	20/06/2022	22. Avaliação Crítica dos Resultados	Relatório / Avaliação Crítica dos Resultados
21/06/2022	27/06/2022	23. Conclusão	Relatório / Conclusão
28/07/2022	29/07/2022	24. Produção do Vídeo da Etapa 03	Resultado: Vídeo da Etapa 03
30/07/2022	30/07/2022	25. Entrega da Etapa 03/Entrega Final	Resultado: Entrega do Relatório Final/Etapa 3

Fonte: Elaborada pelo autor. 2022.

3. Especificação Arquitetural da solução

Esta seção apresenta a especificação básica da arquitetura da solução a ser desenvolvida, incluindo diagramas, restrições e requisitos definidos pelo autor, tal que permitem visualizar a macroarquitetura da solução.

3.1. Restrições Arquiteturais

As restrições arquiteturais foram definidas levando em conta o background técnico da equipe que trabalha hoje na versão legada do software ERP.

ID	Descrição
RA01	Os módulos deverão ser desenvolvidos utilizando a linguagem Java, na versão 11 utilizando a stack de tecnologias Spring Boot.
RA02	As APIs devem seguir o padrão RESTFUL
RA03	Deverá ser utilizado o banco de dados PostgreSQL
RA04	Deverá ser utilizar o protocolo OAuth 2.0 para autorização de acesso dos serviços.

Fonte: Elaborada pelo autor. 2022.

3.2. Requisitos Funcionais

Os requisitos funcionais abaixo foram baseados nas operações básicas do módulo de Gestão de Finanças, o suficiente para a validação inicial da arquitetura.

ID	Descrição	Prioridad e B/M/A	Dificuldade B/M/A
RF01	O usuário poderá manter o Plano de Contas de Fluxo de Caixa da empresa.	A	M
RF02	O usuário poderá manter as contas bancárias da empresa (contas correntes, contas poupança e contas de caixa). A conta deverá ter uma descrição, tipo (corrente, poupança ou caixa) e a qual filial ela pertence.	A	M
RF03	O usuário poderá manter lançamentos financeiros nas contas. O usuário poderá informar a descrição, data, hora, tipo de movimentação (débito ou crédito), valor, a Pessoa de origem/destino do movimento, Filial de Movimento e Conta de Fluxo de Caixa.	A	M
RF04	O usuário poderá efetuar transferências entre contas bancárias.	B	A

	Nela ele irá escolher uma conta origem, uma conta destino, a data e valor a ser transferido.		
RF05	O usuário poderá anexar arquivos aos seus lançamentos, possibilitando guardar comprovantes, NF's e afins no sistema.	B	M
RF06	O usuário poderá manter os Pagar e Receber da empresa. O usuário deverá informar a Filial do Movimento, a Filial de Cobrança, a Conta de Fluxo de Caixa, o Tipo (Pagar ou Receber), o Fornecedor/Cliente, a Data de Emissão, Data de Vencimento e Valor.	A	A
RF07	O sistema irá manter Lançamentos Financeiros para Baixas de Pagar e Receber efetuadas no sistema. Para cada Baixa, será gerado um Lançamento Financeiro. Se removida ou alterada a Baixa, o sistema deverá atualizar o Lançamento Financeiro gerado.	A	A
RF08	O usuário poderá manter baixas (integrais ou totais) de Pagar e Receber.	M	M
RF09	O sistema irá importar os lançamentos financeiro de conciliação bancária (extrato bancário) via integração bancária, para contas configuradas.	M	M
RF10	O usuário poderá enviar para ordens de Pagamento a Fornecedores ou transferências, via integração bancária para contas configuradas. A Ordem de Pagamento/Transferência irá gerar um Pagar ou Receber.	M	A
RF11	O sistema fará baixas automáticas de Ordens de Pagamento a Fornecedores efetuadas.	M	M
RF12	O sistema irá importar os boletos a pagar via integração bancária DDA - Débito Direto Autorizado. O sistema irá gerar um Pagar ou Receber para boletos a pagar	M	M
RF13	O usuário poderá gerar um relatório em XSLX Pendências financeiras. Nele o usuário poderá exibir informações de valores a pagar ou a receber.	M	B
RF14	O usuário poderá gerar um relatório em XLSX de pagamentos e recebimentos. Nele poderá gerar o que já foi pago ou recebido de Fornecedores/Clientes.	M	B
RF15	O usuário final poderá exportar o extrato consolidado dos lançamentos financeiros para Excel (XSLX), utilizando filtros de data, contas bancárias e/ou fluxo de caixa e período.	M	B
RF16	O usuário final poderá visualizar gráficos de evolução financeira por conta bancária e por fluxo de caixa, podendo filtrar por data, conta bancária e/ou fluxo de caixa.	B	B

RF17	O serviço de Pagar e Receber deverá permitir via evento que outros módulos do ERP criem, alterem ou excluam títulos de Pagar e Receber. Também irá enviar eventos de cancelamento, estorno ou rejeições dos Pagar e Receber enviados.	A	A
RF18	O sistema deverá disponibilizar um log de auditoria de Lançamentos Financeiros.	M	M
RF19	O sistema deverá disponibilizar um log de auditoria de Pagar e Receber.	M	M
RF20	O sistema deverá disponibilizar um log de auditoria de Baixas de Pagar e Receber.	M	M

Fonte: Elaborada pelo autor. 2022.

*B=Baixa, M=Média, A=Alta.

3.3. Requisitos Não-funcionais

Os requisitos não-funcionais abaixo foram baseados nos requisitos não-funcionais existentes para o software ERP legado, com a adição de alguns requisitos relativos a softwares distribuídos em geral.

ID	Descrição	Prioridade e B/M/A
RNF01	O sistema deve ser apresentar disponibilidade de funcionamento mensal de 99%.	A
RNF02	O tempo máximo aceitável de duração de qualquer requisição síncrona a API será de 5 segundos.	M
RNF03	A interface frontend deverá ser compatível com os navegadores Google Chrome, Mozilla Firefox, Apple Safari, em versões com até 2 anos.	M
RNF04	O sistema deverá atender as normas legais descritas na LGPD (Lei Geral de Proteção de Dados)	A
RNF05	A API REST deverá ser pública e documentada para integrações externas.	M
RNF06	Todos os logs internos deverão ser acompanhados com a data e hora do evento e identificador de origem do evento (mensagem, usuário, ip, etc)	M
RNF07	O sistema deverá manter um registro dos eventos.	A

Fonte: Elaborada pelo autor. 2022.

*B=Baixa, M=Média, A=Alta.

3.4. Mecanismos Arquiteturais

Abaixo uma visão geral mecânicos que compõem a arquitetura para os novos módulos/funcionalidades do ERP.

Análise	Design	Implementação
Persistência	ORM	JPA / Hibernate / Spring Data JPA
Persistência	Banco de Dados Relacional	Amazon RDS for PostgreSQL
Persistência	Armazenamento de arquivos	Amazon S3
Front end	Framework de interface de usuário	Vue
Front end	UI Library	PrimeVUE
Backend	Regras de Negócio	Java
Backend	Framework de Aplicação	Spring Boot
Backend	Autenticação e Autorização - Cliente	Spring Security
Autenticação e Autorização	Servidor de Autenticação e Autorização	Amazon Cognito
Integração	Eventos/Mensageria	AWS SQS/SNS
Integração	Service Discovery	Netflix Eureka
Integração	API de Integração	API REST usando JSON implementada com Spring MVC REST
Integração	API Gateway	Amazon API Gateway
Monitoramento	Framework de Log	Logback
Monitoramento	Visualização de logs	AWS Cloudwatch Logs
Monitoramento	Alarmes e Métricas	AWS Cloudwatch
Teste de Software	Framework de Testes Unitários - Backend	JUnit
Entrega de Software	Ferramenta de Integração contínua - Backend	AWS CodeBuild
Entrega de Software	Ferramenta de entrega contínua - Backend	AWS Codepipeline / AWS Elastic Beanstalk
Entrega de Software	Ferramenta de Build - Backend	Maven
Entrega de Software	Ferramenta de Integração contínua e entrega contínua – Frontend	AWS Amplify
Entrega de Software	Ferramenta de Build - Frontend	Vue CLI
Versionamento	Versionamento de Código da Aplicação	Git/Github

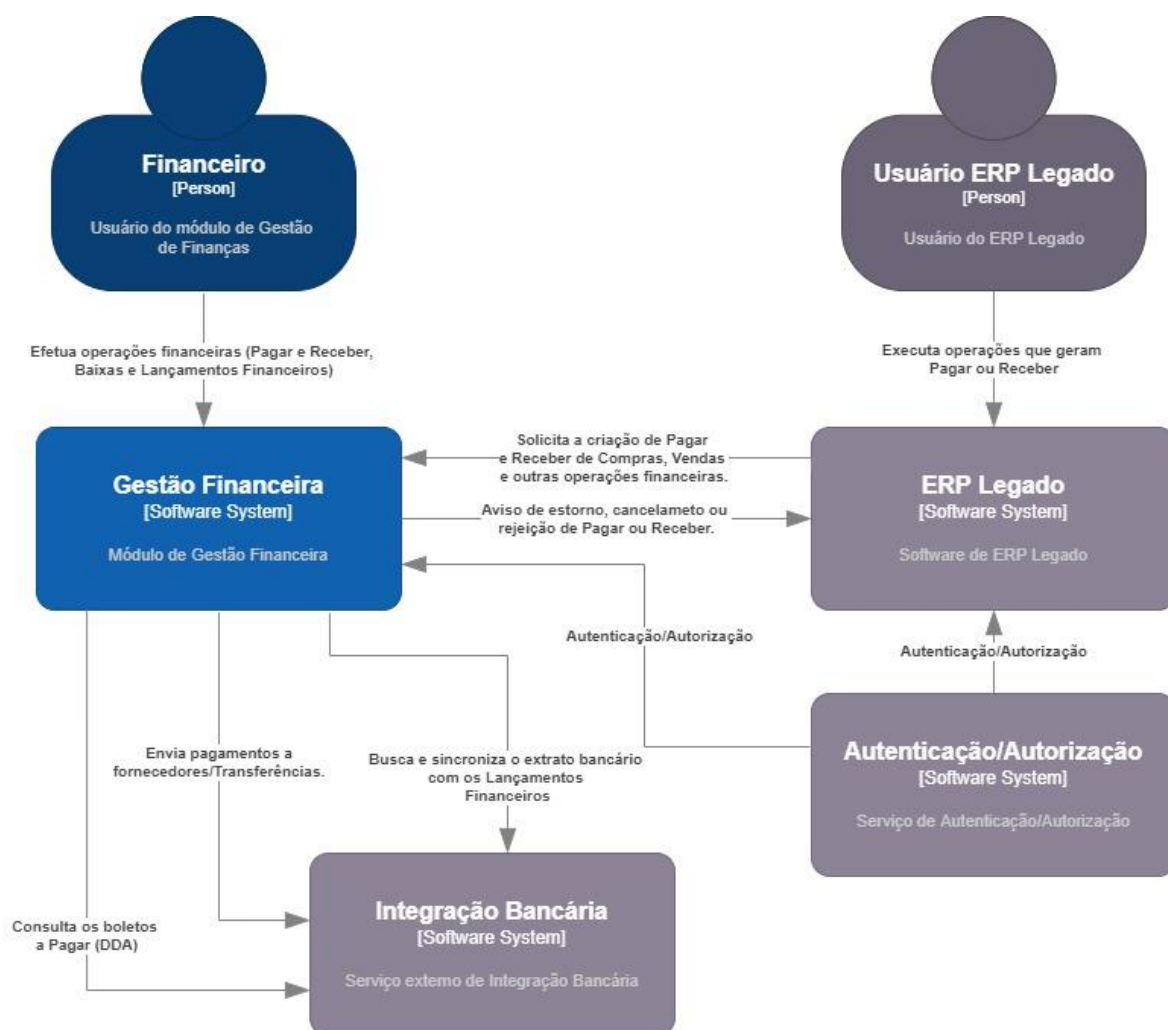
Fonte: Elaborada pelo autor. 2022.

4. Modelagem Arquitetural

Esta seção apresenta a modelagem arquitetural da solução proposta, de forma a permitir seu completo entendimento visando à implementação da prova de conceito (PoC) do módulo de Gestão Financeira na seção 5.

Para esta modelagem arquitetural optou-se por utilizar o modelo C4 para documentação de arquitetura de software. Mais informações a respeito podem ser encontradas aqui: e aqui: <https://www.infoq.com/br/articles/C4-architecture-model/>. Dos quatro nível que compõem o modelo C4 três serão apresentados aqui e somente o Código será apresentado na próxima seção (5).

4.1 Diagrama de Contexto



[Diagrama de Contexto] Sistema ERP

Nova arquitetura do software ERP com Gestão de Finanças Migrado

Figura 1 - Visão Geral da Solução de ERP com o módulo migrado.

A figura 1 mostra a especificação o diagrama geral da solução proposta, com todos seus principais módulos, serviços e pessoas envolvidas. Nele é apresentado a interação entre os papéis e sistemas que compõem a solução completa deseja para o ERP e a interação entre eles.

4.2 Diagrama de Container

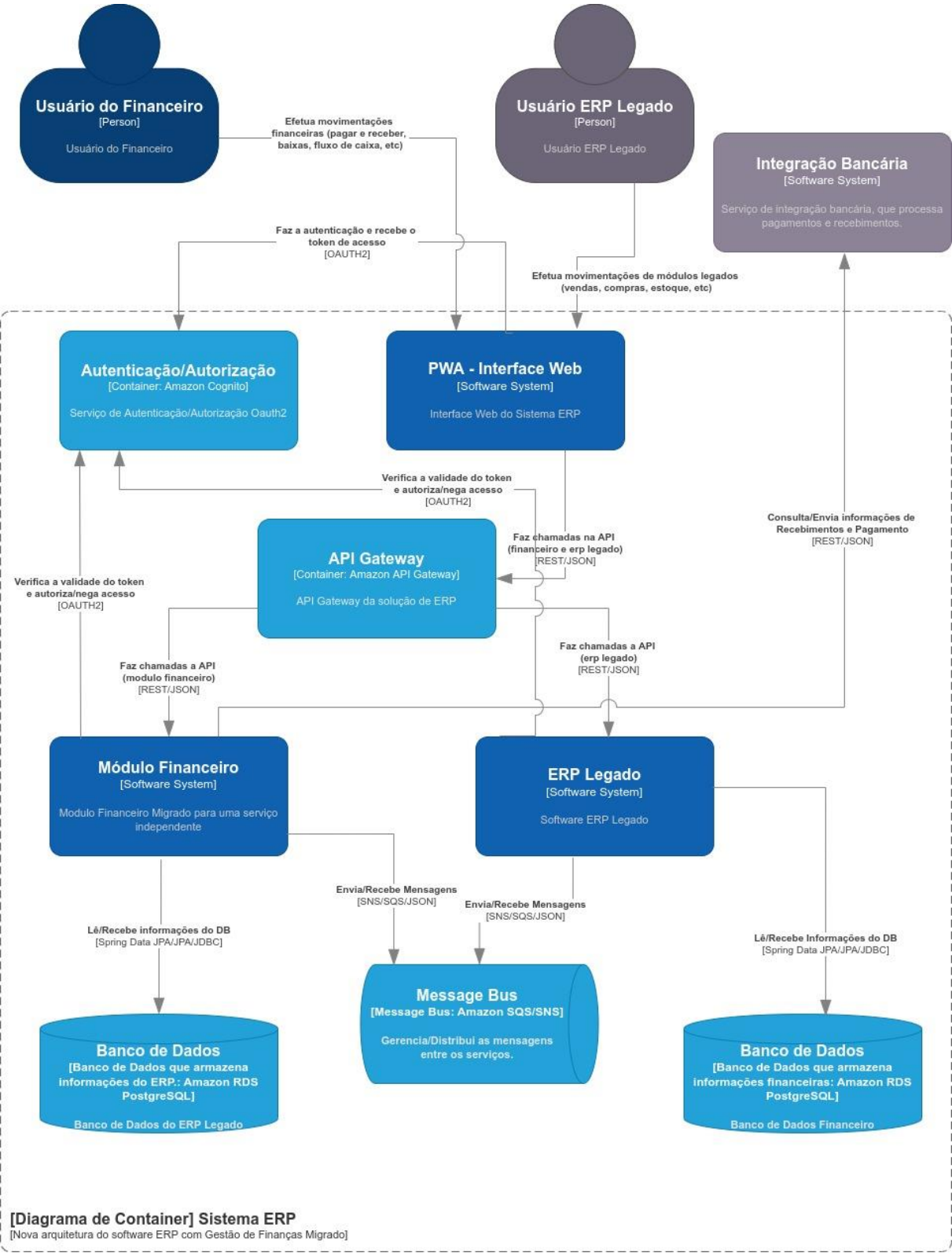


Figura 2 - Diagrama de container

A figura 2 apresenta os containers do Sistema ERP e suas interações com o módulo financeiro. Nesse diagrama é mostrado como ficará a interação entre o ERP Legado, o Módulo Financeiro e a Integração Bancária.

Existem 2 papéis, de utilizar, o usuário do ERP Legado e o usuário do módulo Financeiro. No dia a dia pode acontecer do mesmo usuário assumir ambos os papéis, mas para fins de explicação, foi escolhido detalhar como figuras distintas.

O usuário faz o login na aplicação frontend PWA. A aplicação frontend, em sua tela de login, requisita um token de acesso para o serviço o OAuth2, que é fornecido pelo Amazon Cognito. Com o token de acesso disponível, a aplicação web é autorizada a fazer requisições.

As requisições são feitas através do API Gateway, utilizando a tecnologia Amazon API Gateway. Dependendo da requisição, o API Gateway redireciona as requisições para o módulo financeiro, para o ERP Legado e no futuro para possíveis outros módulos migrados.

Tanto o módulo financeiro quanto o ERP possuem bancos de dados independentes. A comunicação entre o ERP Legado e o módulo Financeiro será feita via mensageria, utilizando as tecnologias Amazon SQS (Fila) e Amazon SNS (tópicos). E para requisições web dos usuários, as aplicações, de forma independente, fazem a checagem do token de acesso com o servidor OAuth2 (Amazon Cognito).

Não será feita a autenticação/autorização diretamente no API Gateway, devido características internas e tecnológicas do ERP Legado. Possivelmente essa será uma implementação em um segundo momento, quando o ERP Legado estiver todo migrado.

O módulo financeiro se comunica via interface REST com o Integrador Bancário, que é um serviço externo de uma empresa parceira. O Integrador Bancário possui uma interface de pagamentos e recebimentos.

4.3 Diagrama de Componentes

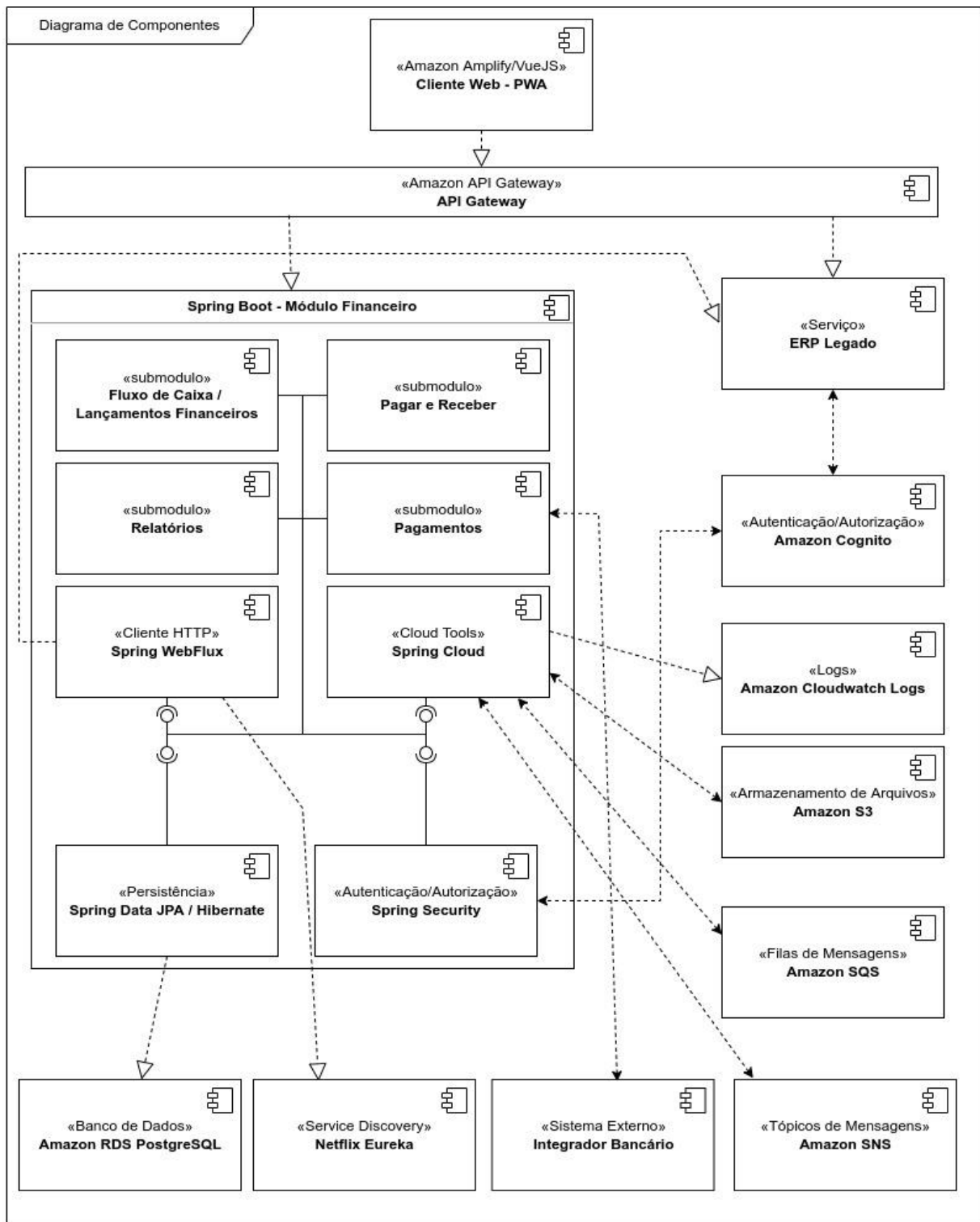


Figura 3 - Diagrama de componentes

A figura 3 mostra o diagrama de componentes da solução, seguindo o padrão UML. Através dele é mostrada com mais detalhes a base tecnológica da aplicação. Os componentes da solução:

- **Cliente Web/PWA:** É uma aplicação desenvolvida em Vue 3, utilizando a biblioteca de componentes PrimeVue e também a biblioteca AWS Amplify para a integração com o serviço Amazon Cognito (servidor OAuth2).
- **API Gateway:** É utilizado o serviço Amazon API Gateway como solução de API Gateway, que é utilizado para unificar a API dos serviços ERP Legado e Módulo Financeiro em um único endpoint.
- **ERP Legado:** O software ERP Legado que está sendo migrado e refatorado.
- **Integrador Bancário:** serviço externo de uma empresa parceira responsável pela integração bancária, de pagamentos e recebimentos. Ela disponibiliza uma API REST para fazer essa integração.
- **Amazon RDS PostgreSQL:** o servidor de banco de dados do módulo financeiro. É utilizado o serviço de bancos de dados gerenciado da Amazon, o Amazon RDS. É utilizado a engine de banco de dados PostgreSQL.
- **Netflix Eureka:** Serviço de Service Discovery.
- **Amazon Cognito:** servidor OAuth2. Por ele é feita a autenticação e autorização dos usuários da solução de ERP.
- **Amazon S3:** é um serviço de armazenamento de objetos (arquivos) da Amazon.
- **Amazon SQS:** O *Amazon Simple Queue Service* (SQS) é um serviço de filas de mensagens gerenciado.
- **Amazon SNS:** É um serviço de notificação da Amazon. Com ele é possível criar tópicos de mensagens. Utilizado para utilização do padrão pub/sub de mensagens.
- **Amazon Cloudwatch Logs:** utilizado para o armazenamento de logs dos serviços.
- **Módulo Financeiro:** O módulo financeiro é um módulo do ERP Legado que está sendo refatorado/migrado para um serviço independente, e é a solução proposta nesse trabalho. Utiliza como base a framework Spring Boot.

- **Fluxo de Caixa:** É um submodulo dentro do Módulo financeiro, responsável por todo controle de fluxo de caixa da empresa.
- **Pagar e Receber:** É um submódulo dentro do módulo financeiro, responsável pelo controle de Pagar e Receber e suas baixas.
- **Relatórios:** Submódulo de relatórios do módulo financeiro.
- **Pagamentos:** É o submódulo responsável por pagamentos.
- **Spring Web Flux:** é uma biblioteca do Spring Boot responsável por consultas REST. É utilizado principalmente pelo módulo de Pagamentos, para a comunicação com o Integrador Bancário.
- **Spring Cloud:** uma biblioteca para o Spring Boot, utilizada para auxiliar a utilização de serviços cloud, como os da Amazon Web Services. Nessa solução, é utilizado para a comunicação com o Amazon SQS e SNS, e também com a comunicação com o serviço Amazon S3. Também é utilizado para disponibilizar o serviço de Service Discovery utilizando o Netflix Eureka para o Spring Web Flux.
- **Spring Data JPA/Hibernate:** O Spring Data JPA é um facilitador para a criação de repositórios JPA. O JPA, sigla para Java Persistence API , é uma API padrão para Java que faz a interface com o banco de dados. É utilizado a biblioteca Hibernate como implementação da API JPA.

5. Prova de Conceito (PoC)

Nessa sessão será detalhada a prova de conceito arquitetural. Para que o objetivo deste trabalho fosse atendido, foram desenvolvidas algumas simulações e foram feitas algumas simplificações negociais, pois o objetivo do trabalho não é validar os requisitos negociais da aplicação, mas sim sua arquitetura.

5.1. Integrações entre Componentes

O objetivo central deste trabalho é a migração do módulo financeiro de um sistema de gestão. Como não é possível a utilização do sistema real para os propósitos deste trabalho foram desenvolvidos alguns artefatos para ajudar a atingi-los.

Primeiramente, foi desenvolvido um subprojeto chamado “ERP Legado”. Esse projeto possui apenas algumas funções essenciais para a validação do módulo financeiro. Foi desenvolvida a tela de Filiais e Pessoas, e também uma tela de “Venda”, que seria uma simplificação de uma operação existente no ERP Legado, apenas para validar a criação de Pagar e Receber por solicitação de algum outro módulo.

Também foi desenvolvido um subprojeto chamado “Integração Bancária”, que tem como objetivo simular o sistema externo de Integração Bancária. Nele foi criada uma API simplificada das operações de Recebimentos e Pagamentos, e algumas funcionalidades para configurar o comportamento desse aplicativo, como por exemplo, se o mesmo vai rejeitar ou aceitar os Pagamentos e Recebimentos.

Foi desenvolvido o módulo Financeiro, com algumas funcionalidades essenciais para as validações das requisições, como cadastros de Contas Bancárias, Contas Fluxo de Caixa. Também foram desenvolvidas funcionalidades como Lançamentos de Fluxo de Caixa, Pagar e Receber, Baixas de Pagar e Receber e Ordem de Pagamento.

Foi desenvolvido um protótipo completo da prova de conceito, e o mesmo pode ser acessado em <https://serathiuk.dev> (login: **demopucmg** e senha: **DemoPUCMG2022***), até o final da avaliação deste trabalho. Também é possível criar novos usuários pela tela de login caso seja necessário.

O protótipo utiliza grande parte da arquitetura proposta em ambiente real da Amazon Web Services. No repositório de código-fonte também possui os componentes caso se deseje fazer a simulação em ambiente local (na própria máquina). Para isso existe uma configuração de testes e orientações como utilizar o Docker para rodar o projeto, utilizando PostgreSQL e Localstack (ambiente de simulação da AWS) localmente.

5.1.1. Requisito funcional RF10 – Enviar Ordens de Pagamentos

O objetivo desse requisito é validar o envio de Ordens de Pagamento para um serviço externo de integrador. Ao mesmo tempo validar um padrão de integração distribuída enviando dados em lote, com processamento assíncrono, para um serviço externo utilizando serviços REST.

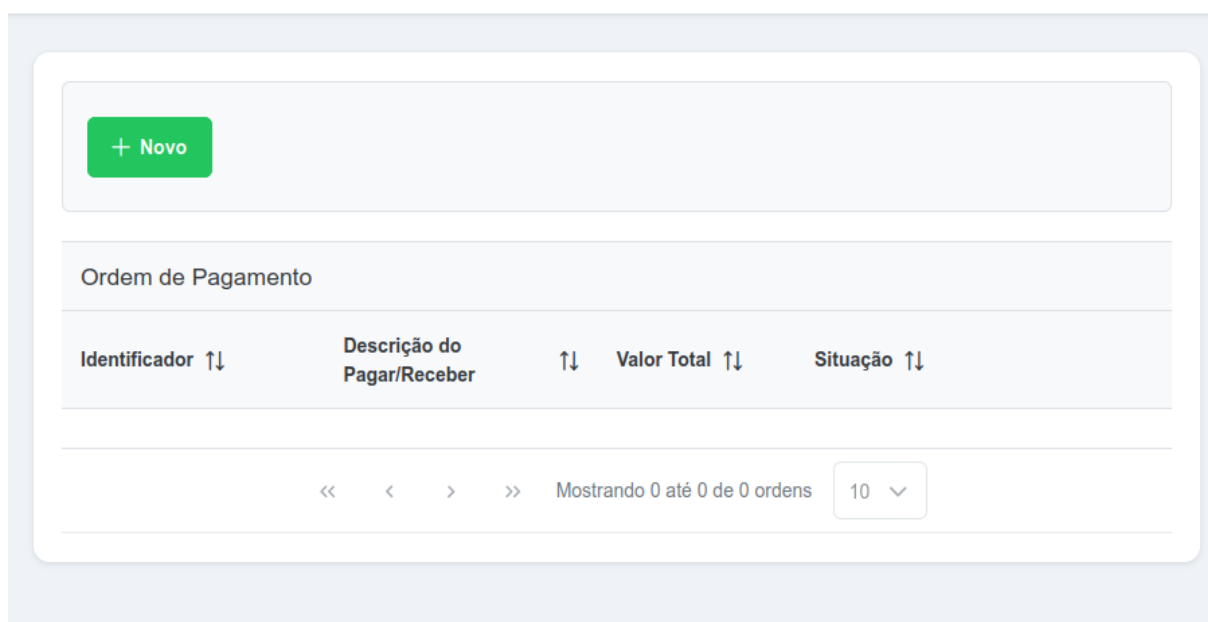



Figura 4 - Tela de Ordens de Pagamentos

A partir dessa tela, é possível cadastrar novas Ordens de Pagamento para Contas a Pagar pendentes:



The image shows a modal window titled "Ordem de Pagamento" with a close button (X) in the top right corner. The modal contains four input fields:

- Pagar/Receber:** A dropdown menu with the selected value "TESTE PUC MG" and a downward arrow.
- Filial de Pagamento:** A dropdown menu with the selected value "Matriz" and a downward arrow.
- Conta Bancária:** A dropdown menu with the selected value "Conta Corrente Banco 1" and a downward arrow.
- Valor Total:** A text input field containing "R\$ 1.000,00".

At the bottom of the modal, there are two buttons: "X Cancelar" and "✓ Salvar".

Figura 5 - Cadastro de Ordens de Pagamentos

Depois de cadastrada, a Ordem de Pagamento, ficará o status NAO_ENVIADO, até que a tarefa agendada do módulo financeiro envie a Ordem de Pagamento para o Integrador Bancário.

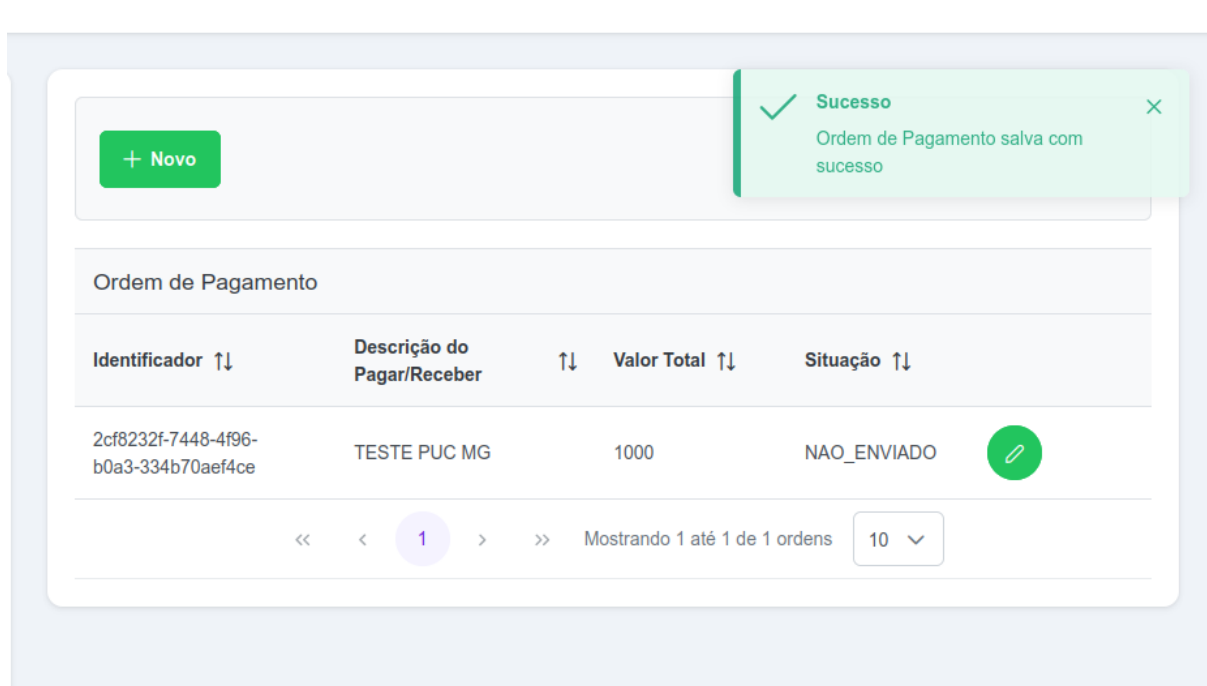


Figura 6 - Ordem de Pagamento criada com sucesso

Após enviar o status ficará como EM_PROCESSAMENTO, confirmando assim o envio da Ordem de Pagamento para o Integrador Bancário.

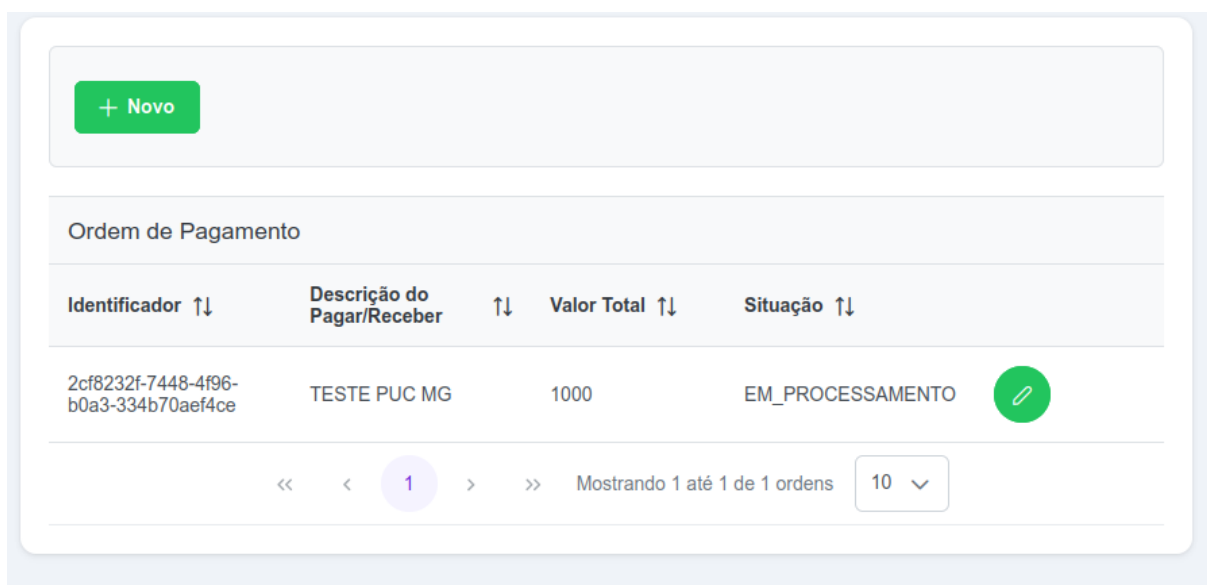
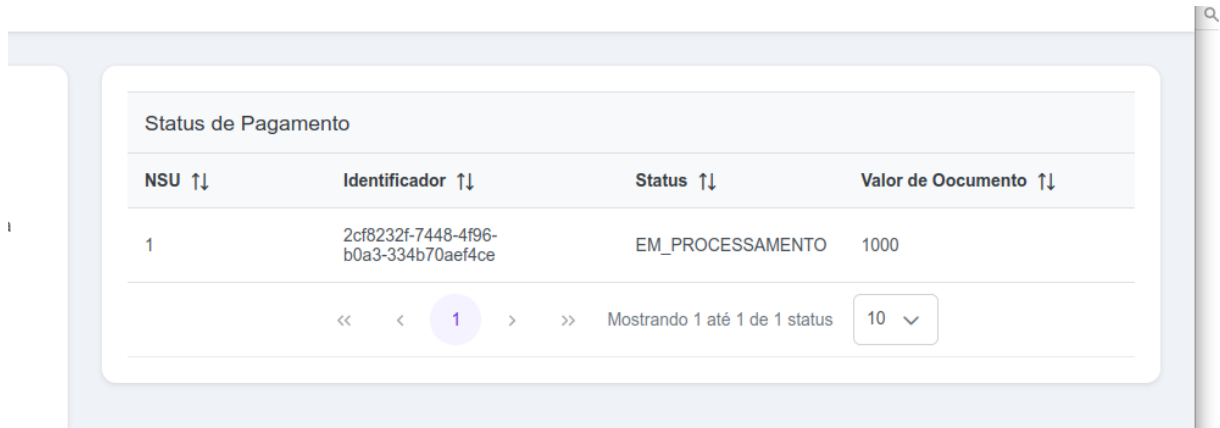


Figura 7 - Ordem de Pagamento enviada com sucesso



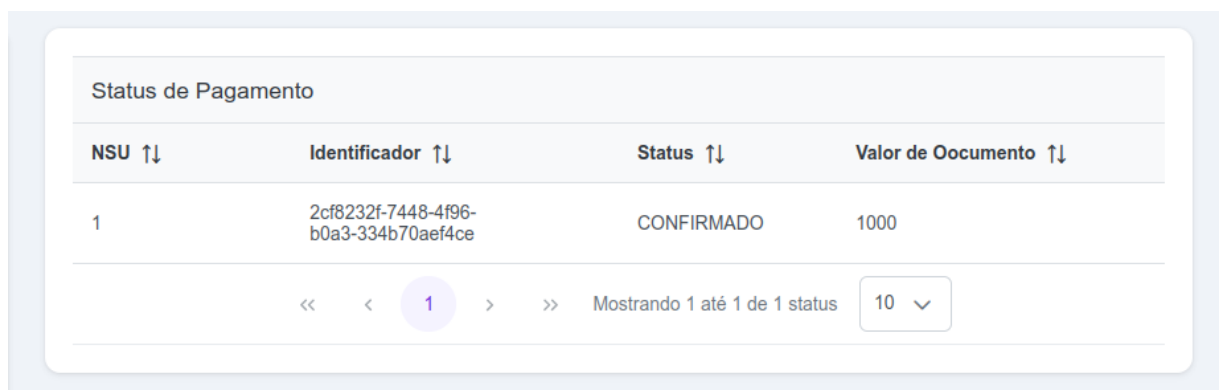
Status de Pagamento			
NSU ↑↓	Identificador ↑↓	Status ↑↓	Valor de Documento ↑↓
1	2cf8232f-7448-4f96-b0a3-334b70aef4ce	EM_PROCESSAMENTO	1000

<< < 1 > >> Mostrando 1 até 1 de 1 status 10 ▾

Figura 8 - Foi recebido com sucesso pela Integração Bancária

5.1.2. Requisito funcional RF11 – Baixas automáticas de Conta a Pagar em respostas de Ordens de Pagamentos

Após o envio de uma Ordem de Pagamento, conforme do RF10, a Integração Bancária irá se comunicar com o Agente Financeiro (Bancos) para enviar a intenção de pagamento. Após a Integração Bancária tiver a resposta, irá mudar seu status para confirmado. O módulo financeiro consulta em lote a confirmação dos pagamentos. E quando confirmado pela Integração Bancária, o Módulo Financeiro deverá gerar uma baixa para a Conta a Pagar vinculada a Ordem de Pagamento.



Status de Pagamento			
NSU ↑↓	Identificador ↑↓	Status ↑↓	Valor de Documento ↑↓
1	2cf8232f-7448-4f96-b0a3-334b70aef4ce	CONFIRMADO	1000

<< < 1 > >> Mostrando 1 até 1 de 1 status 10 ▾

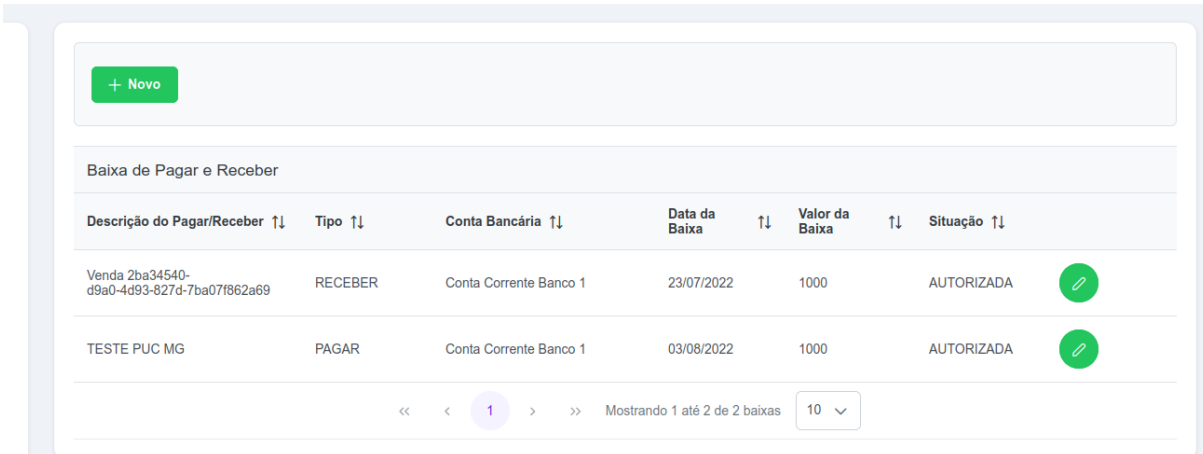
Figura 9 - Confirmado na Integração Bancária





Venda 2ba34540-d9a0-4d93-827d-7ba07f862a69	21/07/2022	223232	RECEBER	QUITADO	
TESTE PUC MG	01/08/2022	1000	PAGAR	QUITADO	

<< < 1 > >> Mostrando 1 até 6 de 6 lançamentos 10 ▾

Figura 10 - Pagar e Receber alterado para quitado



Baixa de Pagar e Receber						
Descrição do Pagar/Receber ↑↓	Tipo ↑↓	Conta Bancária ↑↓	Data da Baixa ↑↓	Valor da Baixa ↑↓	Situação ↑↓	
Venda 2ba34540-d9a0-4d93-827d-7ba07f862a69	RECEBER	Conta Corrente Banco 1	23/07/2022	1000	AUTORIZADA	
TESTE PUC MG	PAGAR	Conta Corrente Banco 1	03/08/2022	1000	AUTORIZADA	

<< < 1 > >> Mostrando 1 até 2 de 2 baixas 10 ▾

Figura 11 - Baixa gerada automaticamente

5.1.3. Requisito funcional RF17 – Possibilitar o recebimento de eventos de criação de Pagar e Receber

Para possibilitar o teste de criação de Pagar e Receber através de outros módulos, ou até mesmo através do ERP Legado, foi criada a funcionalidade de Simulação de Venda. Essa funcionalidade nada mais que simula de forma simplista uma Venda enviada pelo ERP Legado. O ERP Legado envia via mensagem (Amazon SQS) a solicitação de criação de um novo Pagar e Receber. E como resposta, o listener de Criação de Pagar e Receber reenvia uma mensagem, através de um tópico do Amazon SNS, notificando sobre a situação da criação do Pagar e Receber, se foi criado ou não.

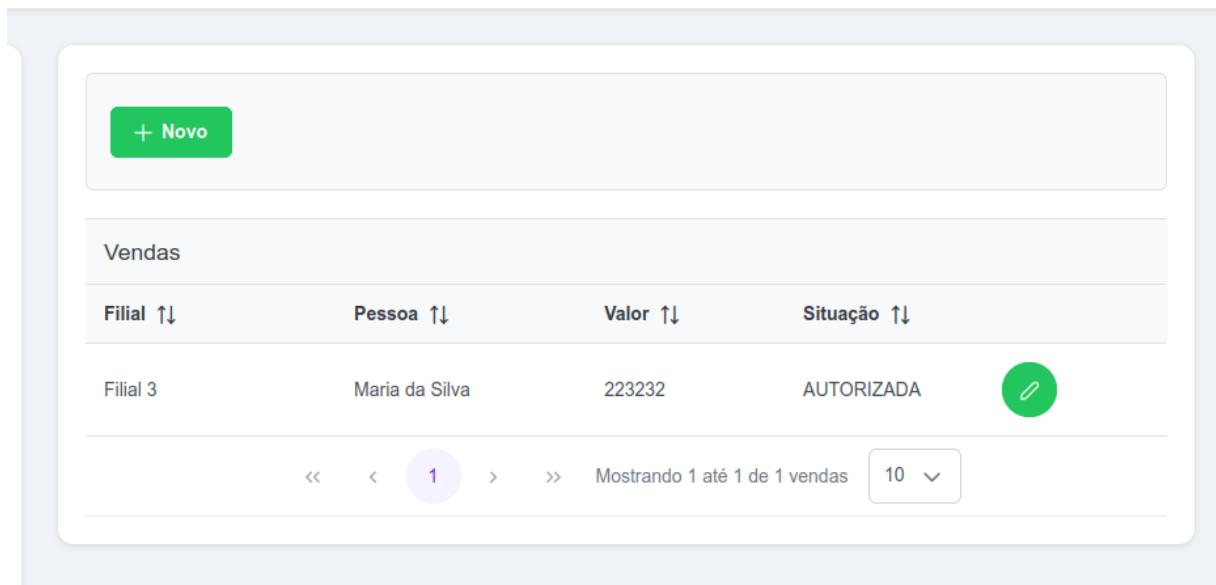


Figura 12 - Tela de Vendas do ERP Legado

A partir da tela de Venda é cadastrada uma nova Venda. Após o cadastro de uma nova venda é enviada uma mensagem via fila Amazon SQS, solicitando a criação para o módulo Financeiro. O financeiro executa o processo de criação e envia a resposta para o tópico Amazon SNS. Com a resposta o ERP Legado atualiza a situação da Venda para Autorizada.

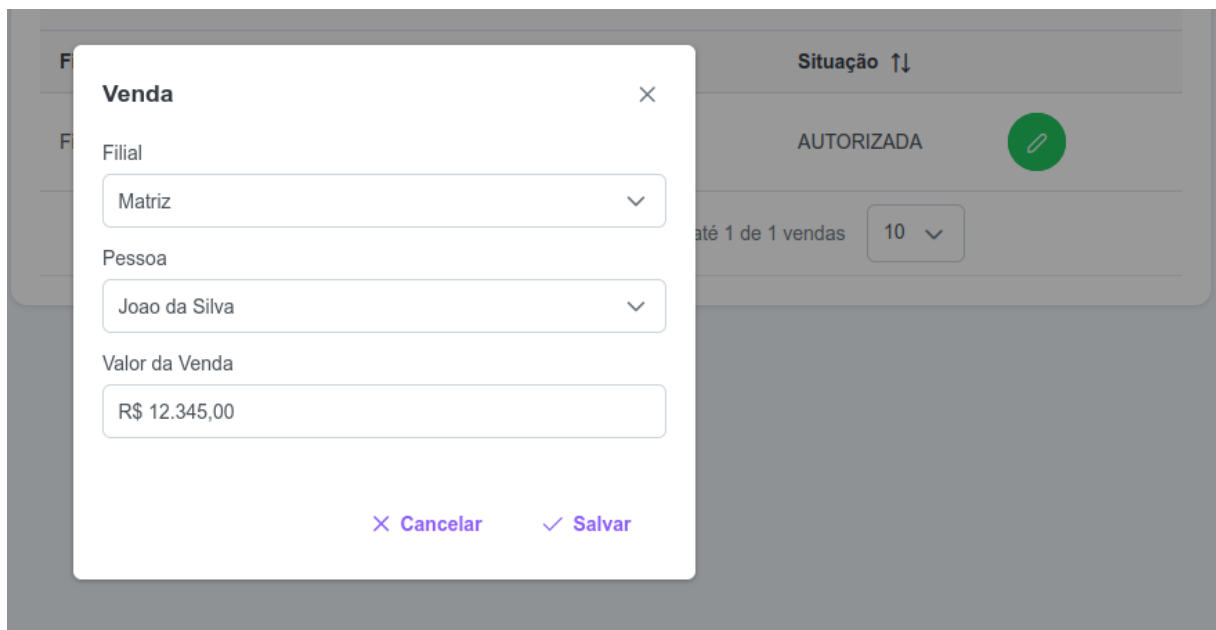


Figura 13 - Cadastro de uma nova Venda

+ Novo

Sucesso

Venda salvo com sucesso

X

Vendas			
Filial ↑↓	Pessoa ↑↓	Valor ↑↓	Situação ↑↓
Filial 3	Maria da Silva	223232	AUTORIZADA
Matriz	Joao da Silva	12345	PENDENTE

<<

<

1

>

>>

Mostrando 1 até 2 de 2 vendas

10 ▾

Figura 14 - A venda após de salva fica como pendente

+ Novo

Vendas			
Filial ↑↓	Pessoa ↑↓	Valor ↑↓	Situação ↑↓
Filial 3	Maria da Silva	223232	AUTORIZADA
Matriz	Joao da Silva	12345	AUTORIZADA

<<

<

1

>

>>

Mostrando 1 até 2 de 2 vendas

10 ▾

Figura 15 - Após a confirmação por parte do módulo financeiro, a Venda altera o status para AUTORIZADA

Pagar e Receber X

Filial de Movimento
Matriz

Filial de Cobrança
Matriz

Pessoa
Joao da Silva

Descrição do Lançamento
Venda 21db45be-91b0-49fa-9476-eda6e15a3bed

Tipo de Operação
☐ Pagar
 ☒ Receber

Data da Emissão
2022-08-03

Data da Vencimento
2022-09-02

Valor da Operação
R\$ 12.345,00

Conta Fluxo de Caixa
VENDAS

X Cancelar

Figura 16 - A venda gerada no Pagar e Receber

5.2. Código da Aplicação

Nessa sessão será explicado a nível de código o funcionamento dos requisitos escolhidos. O código fonte completo da aplicação pode ser acessado no endereço: <https://github.com/serathiuk/pocgestaofinanceira>

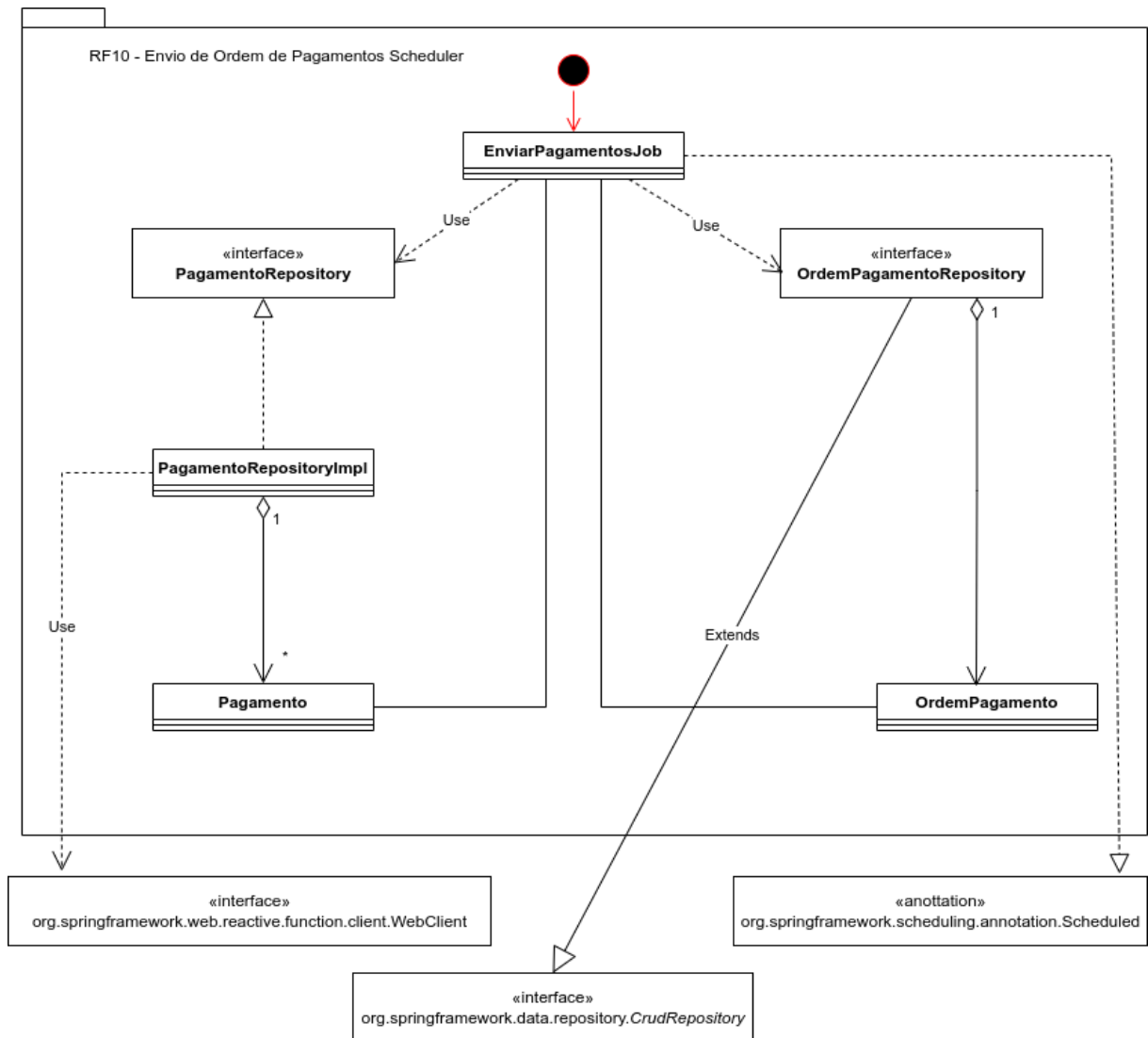


Figura 17 - RF10 – Enviar Ordens de Pagamento para a Integração Bancária

Na figura 17 é detalhado o funcionamento do requisito RF10. A classe `EnviarPagamentosJob` é uma task agendada do Spring Boot, configurada com a anotação `Scheduled`. São consultadas no banco de dados todas as Ordens de Pagamento pendentes através da interface `OrdemPagamentoRepository`, que utiliza o `CrudRepository` do Spring Data JPA para gerar uma implementação de um Repository em tempo de execução. Após a consulta dos dados, são enviadas as Ordens de Pagamento para o serviço de Integração Bancária através da classe `PagamentoRepository` e sua implementação `PagamentoRepositoryImpl`, que faz as requisições REST utilizando a classe `WebClient` do Spring WebFlux. Se enviado com sucesso, a Ordem de Pagamento terá o status atualizado para `Em Processamento`.

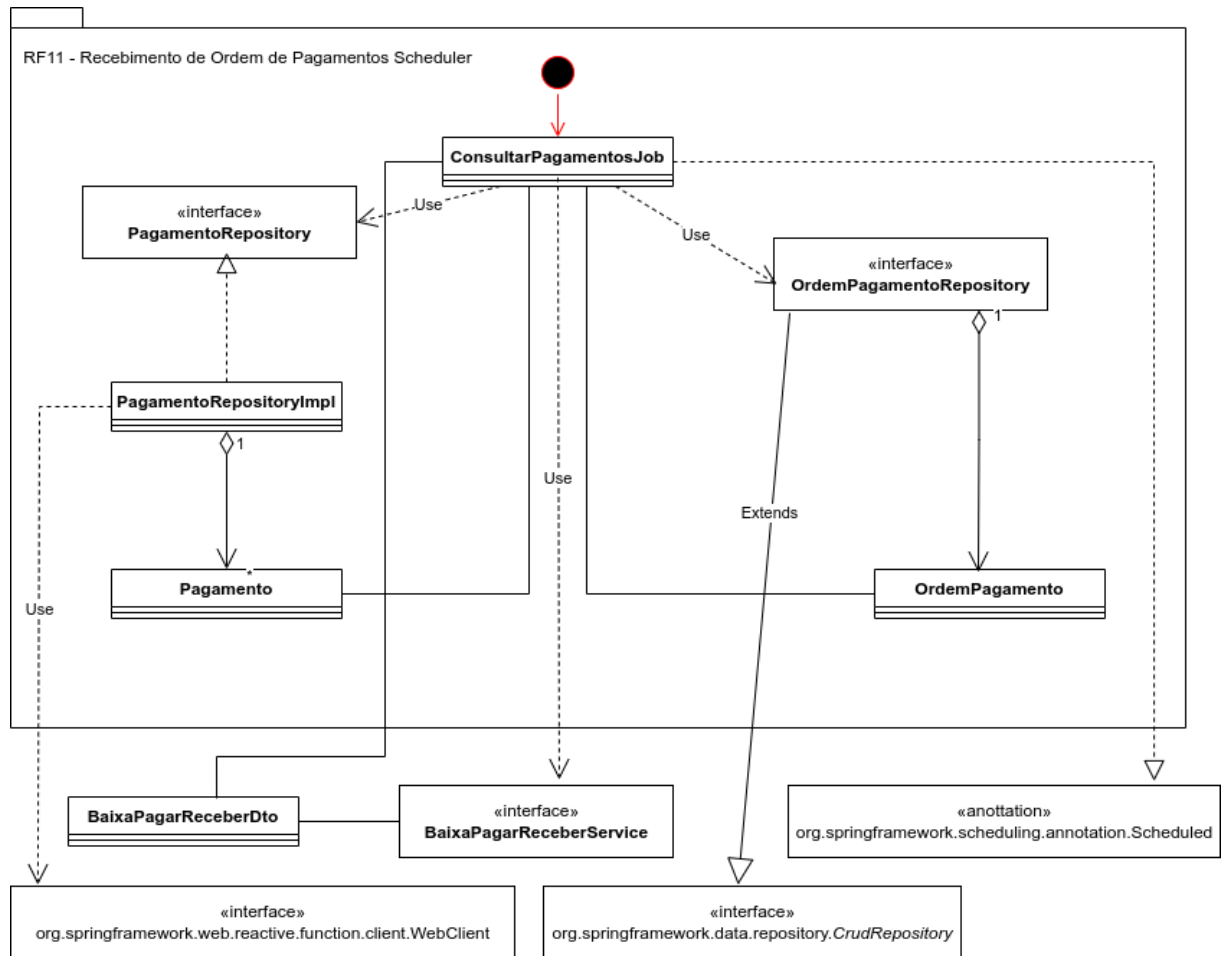


Figura 18 - RF11 – Receber a situação das Ordens de Pagamento da Integração Bancária

A figura 18 é mostrado o funcionamento do requisito RF11, a consulta de Ordens de Pagamentos enviadas. A classe ConsultarPagamentosJob é uma task agendada do Spring Boot, configurada com a anotação Scheduled. São consultadas no banco de dados todas as Ordens de Pagamento em processamento através da interface OrdemPagamentoRepository, que utiliza o CrudRepository do Spring Data JPA para gerar uma implementação de um Repository em tempo de execução. Após a consulta dos dados no banco, são consultadas a situação das Ordens de Pagamento no Integração Bancária através da classe PagamentoRepository e sua implementação PagamentoRepositoryImpl, que faz as requisições REST utilizando a classe WebClient do Spring WebFlux. Se a consulta é retornada com Rejeição, a situação da Ordem de Pagamento é alterada para rejeitada. Se a consulta é retornada com sucesso, é alterada a situação da Ordem de Pagamento para Autorizada e é gerada a Baixa da Conta a Pagar vinculada a Ordem de Pagamento.

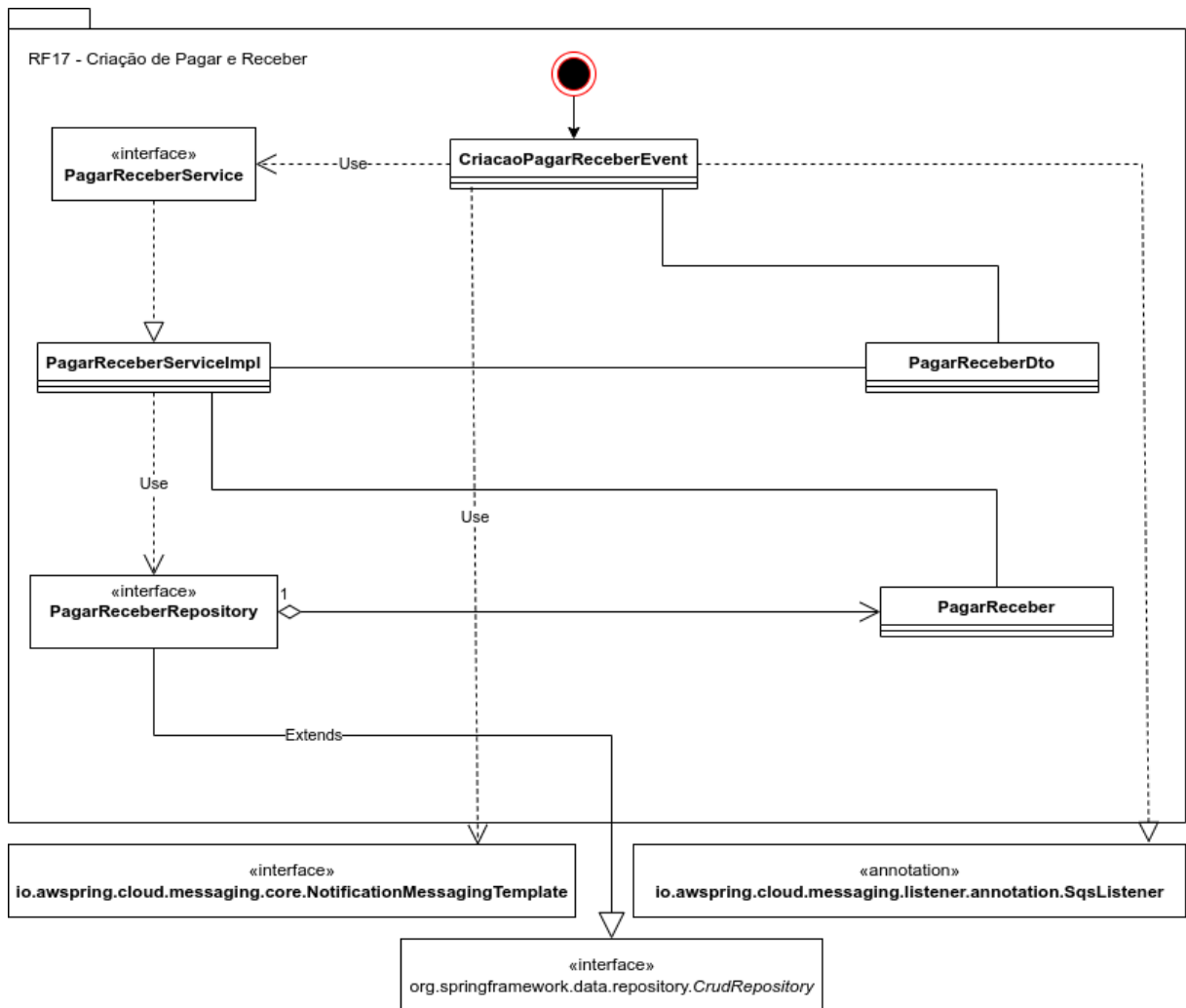


Figura 19 - RF17 – Criar Pagar e Receber via Evento

A classe `CriacaoPagarReceberEvent` é um listener de uma fila SQS, e é anotada com a annotation `SqsListener` do Spring Cloud. Essa classe recebe a mensagem no formato JSON, que segue a estrutura do `PagarReceberDto`. Com o `PagarReceberDto` preenchido, é utilizada a interface `PagarReceberService`, e sua implementação, a classe `PagarReceberServiceImpl`, para iniciar o processo de salvar. O `PagarReceberService` é a classe de negócios do Pagar e Receber. O `PagarReceberDto` é convertido para a classe de modulo `PagarReceber`, e é persistida no banco de dados com a interface que `PagarReceberRepository`, uma interface de implementação do Spring Data JPA. Após a persistencia, o fluxo volta para a `CriacaoPagarReceberEvent`, onde é enviada a resposta de sucesso ou falha via tópico do Amazon SNS utilizando a classe `NotificationMessagingTemplate` do Spring Cloud. Com isso é criado (ou não) o Pagar e Receber e notificado todos os serviços que são inscritos no tópico SNS correspondente.

6. Avaliação de Arquitetura (ATAM)

A avaliação da arquitetura desenvolvida neste trabalho é abordada nesta seção visando avaliar se ela atende ao que foi solicitado pelo cliente, segundo o método ATAM.

6.1. Análises das abordagens arquiteturais

Atributos de Qualidade	Cenários	Importância	Complexidade
Disponibilidade	Cenário 1: O sistema deve ser apresentar disponibilidade de funcionamento mensal de 99%.	A	M
Desempenho	Cenário 2: O tempo máximo aceitável de duração de qualquer requisição síncrona a API será de 5 segundos.	M	B
Usabilidade	Cenário 3: A interface frontend deverá ser compatível com os navegadores Google Chrome, Mozilla Firefox, Apple Safari, em versões com até 2 anos.	M	B
Conformidade	Cenário 4: A API REST deverá ser pública e documentada para integrações externas.	M	B
	Cenário 5: O sistema deverá atender as normas legais descritas na LGPD (Lei Geral de Proteção de Dados)	A	A
Rastreabilidade	Cenário 6: Todos os logs internos deverão ser acompanhados com a data e hora do evento e identificador de origem do evento (mensagem, usuário, ip, etc)	M	M
	Cenário 7: O sistema deverá manter um registro dos eventos	A	M

6.2. Cenários

Cenário 1 – Disponibilidade. O sistema deverá estar disponível para o cliente em 99% do tempo durante o mês. Pelo software de ERP ser um software de missão

crítica, onde toda a rotina da empresa é mantida, o software deverá ter disponibilidade alta, e evitando-se manutenções fora de horário comercial.

Cenário 2 – Desempenho: Qualquer requisição síncrona deverá ter no máximo 5 segundos. Para requisições fora desse cenário, deverão ser utilizados processamentos assíncronos.

Cenário 3 – Usabilidade: A interface com o cliente deverá ser compatível com os principais navegadores do mercado, e suas versões lançadas pelo menos há 2 anos.

Cenário 4 – Conformidade: Para facilitar integrações tanto internas quanto externas, todas as API's devem seguir o padrão REST, com documentação visível do seu funcionamento.



Cenário 5 – Conformidade: O sistema deverá seguir as regras definidas na LGPD, trazendo sigilo para os dados dos usuários pessoa física.

Cenário 6 – Rastreabilidade – Todos os logs internos deverão descrever os erros, execuções e informações relevantes durante a execução do sistema, conforme sua origem e conforme o horário que o evento ocorreu.

Cenário 7 – Rastreabilidade – O sistema deverá manter as mensagens enviadas entre os serviços, para fins de auditoria ou até mesmo de rastreabilidade de possíveis problemas futuros.

6.3. Evidências da Avaliação

Atributo de Qualidade:	Disponibilidade
Requisito de Qualidade:	O sistema deve ser apresentar disponibilidade de funcionamento mensal de 99%.
Preocupação:	
O sistema deve ter como resposta a uma requisição uma saída de fácil leitura por outro componente.	
Cenário(s):	

Cenário 1																			
Ambiente:																			
Sistema em operação normal.																			
Estímulo:																			
Chamadas na API REST.																			
Mecanismo:																			
Utilização normal do sistema e monitoramento via AWS Cloudwatch																			
Medida de resposta:																			
Mecanismo de autoscaling e monitoramento do AWS Beanstalk.																			
<div><div>Elastic Beanstalk > Environments > SerathiukFinanceiro</div><div><div><div>SerathiukFinanceiro</div><div>serathiukfinanceiro.us-east-1.elasticbeanstalk.com (e-ubzihc7u7g)</div><div>Application name: SerathiukERPLegado</div></div><div><div>Refresh</div><div>Actions</div></div></div><div><div><div><div>Health</div><div></div><div>Ok</div><div>Causes</div></div><div><div>Running version</div><div>financeiro-20</div><div>Upload and deploy</div></div><div><div>Platform</div><div></div><div>Corretto 11 running on 64bit Amazon Linux 2/3.4.0</div><div>Change</div></div></div><div><div>Recent events</div><div>Show all</div><div><div>< 1 ></div><table><thead><tr><th>Time</th><th>Type</th><th>Details</th></tr></thead><tbody><tr><td>2022-10-05 19:13:46 UTC-0300</td><td>INFO</td><td>Environment health has transitioned from Info to Ok. Configuration update completed 51 seconds ago and took 12 minutes.</td></tr><tr><td>2022-10-05 19:12:56 UTC-0300</td><td>INFO</td><td>Environment update completed successfully.</td></tr><tr><td>2022-10-05 19:12:56 UTC-0300</td><td>INFO</td><td>Successfully deployed new configuration to environment.</td></tr><tr><td>2022-10-05 19:11:46 UTC-0300</td><td>INFO</td><td>Removed instance [i-0cfe4b2534ac4bee8] from your environment.</td></tr><tr><td>2022-10-05 19:09:29 UTC-0300</td><td>INFO</td><td>Deployment succeeded. Terminating old instances and temporary Auto Scaling group.</td></tr></tbody></table></div></div></div></div>		Time	Type	Details	2022-10-05 19:13:46 UTC-0300	INFO	Environment health has transitioned from Info to Ok. Configuration update completed 51 seconds ago and took 12 minutes.	2022-10-05 19:12:56 UTC-0300	INFO	Environment update completed successfully.	2022-10-05 19:12:56 UTC-0300	INFO	Successfully deployed new configuration to environment.	2022-10-05 19:11:46 UTC-0300	INFO	Removed instance [i-0cfe4b2534ac4bee8] from your environment.	2022-10-05 19:09:29 UTC-0300	INFO	Deployment succeeded. Terminating old instances and temporary Auto Scaling group.
Time	Type	Details																	
2022-10-05 19:13:46 UTC-0300	INFO	Environment health has transitioned from Info to Ok. Configuration update completed 51 seconds ago and took 12 minutes.																	
2022-10-05 19:12:56 UTC-0300	INFO	Environment update completed successfully.																	
2022-10-05 19:12:56 UTC-0300	INFO	Successfully deployed new configuration to environment.																	
2022-10-05 19:11:46 UTC-0300	INFO	Removed instance [i-0cfe4b2534ac4bee8] from your environment.																	
2022-10-05 19:09:29 UTC-0300	INFO	Deployment succeeded. Terminating old instances and temporary Auto Scaling group.																	
Considerações sobre a arquitetura:																			
Riscos:	Pode haver problemas que fogem do nosso controle, como problemas internos																		


	na infraestrutura cloud, já que foi pensado nessa solução em um ambiente muito região.
Pontos de Sensibilidade:	Problemas na cloud pública ou nas rotas de comunicação com a mesma.
Tradeoff:	Complexidade e custo de uma solução multi-cloud ou multi região.

Atributo de Qualidade:	Desempenho
Requisito de Qualidade:	O tempo máximo aceitável de duração de qualquer requisição síncrona a API será de 5 segundos.
Preocupação:	
O sistema deverá manter requisições curtas, para que problemas de conexões não causem problemas de usabilidade e confiabilidade do sistema.	
Cenário(s):	
Cenário 2	
Ambiente:	
Sistema em operação.	
Estímulo:	
Chamadas na API REST.	
Mecanismo:	
Utilização normal do sistema e monitoramento via AWS Cloudwatch	
Medida de resposta:	
Mecanismo de autoscaling e monitoramento do AWS Beanstalk.	


Elastic Beanstalk > Environments > SerathiukFinanceiro

SerathiukFinanceiro
[serathiukfinanceiro.us-east-1.elasticbeanstalk.com](#) (e-ubzihc7u7g)
 Application name: **SerathiukERPLegado**

[Refresh](#)
[Actions ▼](#)

Health

 Ok
[Causes](#)

Running version
 financeiro-20
[Upload and deploy](#)

Platform

 Corretto 11 running on 64bit
 Amazon Linux 2/3.4.0
[Change](#)

Recent events
[Show all](#)

Time	Type	Details
2022-10-05 19:13:46 UTC-0300	INFO	Environment health has transitioned from Info to Ok. Configuration update completed 51 seconds ago and took 12 minutes.
2022-10-05 19:12:56 UTC-0300	INFO	Environment update completed successfully.
2022-10-05 19:12:56 UTC-0300	INFO	Successfully deployed new configuration to environment.
2022-10-05 19:11:46 UTC-0300	INFO	Removed instance [i-0cfe4b2534ac4bee8] from your environment.
2022-10-05 19:09:29 UTC-0300	INFO	Deployment succeeded. Terminating old instances and temporary Auto Scaling group.

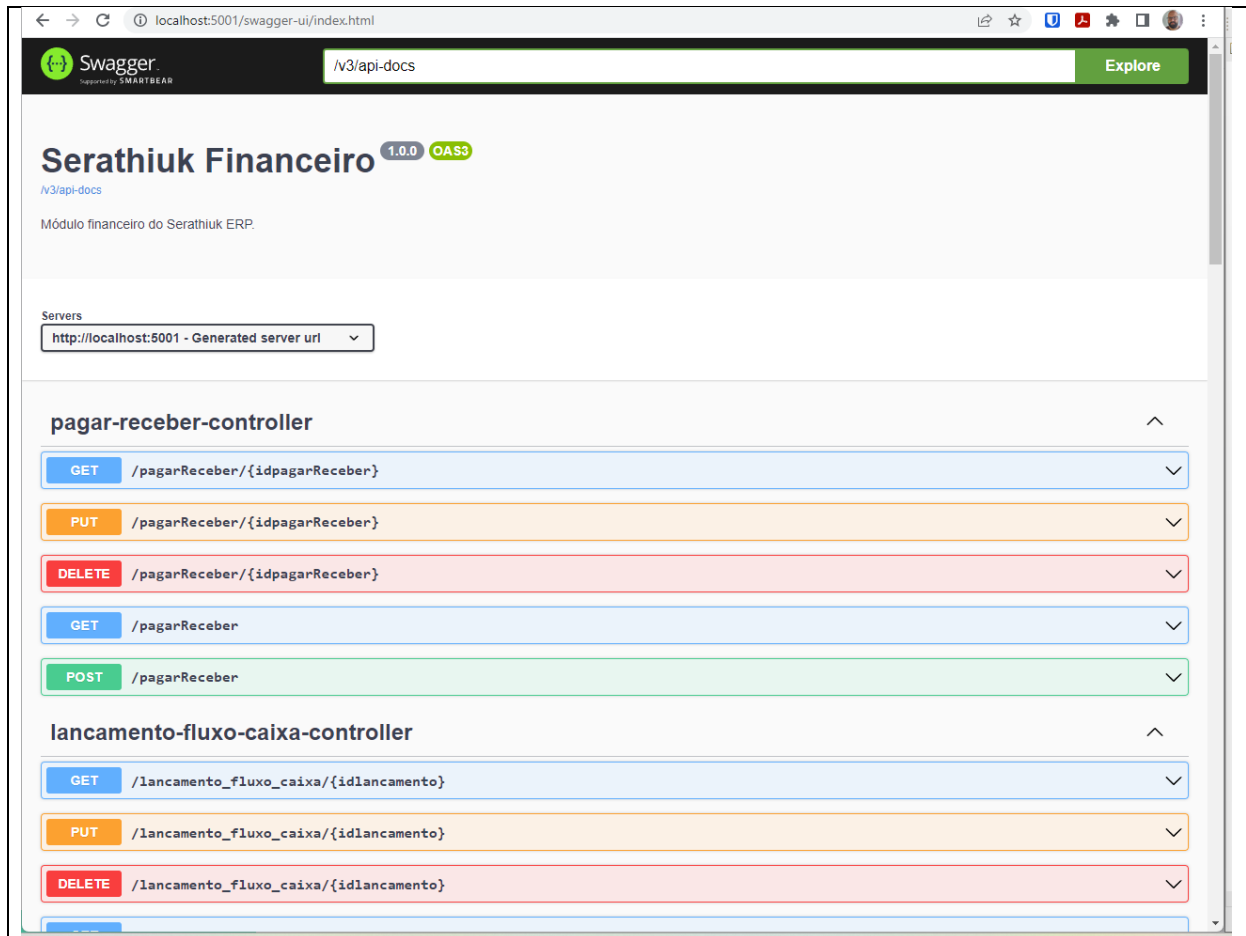
Considerações sobre a arquitetura:

Riscos:	Uma demanda muito maior que a planejada pode comprometer a utilização planejada do banco de dados e não cumprir esse requisito em alguns momentos.
Pontos de Sensibilidade:	Limitação do banco de dados.
Tradeoff:	A capacidade do banco de dados pode ser aumentada para atender esses picos, o que aumentaria o custo. Ou podem ser implementados mecanismos de cache para diminuir a demanda do banco.

Atributo de Qualidade:	Usabilidade
Requisito de Qualidade:	Compatibilidade com navegadores mais recentes no mercado.
Preocupação:	
O sistema deverá ser compatível com os navegadores mais recentes do mercado, de forma que qualquer usuário com navegadores recentes não tenha problemas com a interface do sistema.	
Cenário(s):	
Cenário 3	
Ambiente:	
Ambiente frontend no AWS Amplify	
Estímulo:	
Utilização da interface do sistema.	
Mecanismo:	
VueJS + PrimeVue	
Medida de resposta:	
<p>Compatibilidade prometida pelas frameworks</p> <hr/> <p>What browsers does Vue support?</p> <p>The latest version of Vue (3.x) only supports browsers with native ES2015 support. This excludes IE11. Vue 3.x uses ES2015 features that cannot be polyfilled in legacy browsers, so if you need to support legacy browsers, you will need to use Vue 2.x instead.</p> <hr/>	
Considerações sobre a arquitetura:	
Riscos:	Mudanças maiores entre major versions do VueJS e PrimeVue podem quebrar a compatibilidade e impossibilitar a atualização fácil e compatibilidade com novos navegadores e patches de

	compatibilidade podem demorar a vir.
Pontos de Sensibilidade:	Quebras de compatibilidade navegador vs bibliotecas frontend
Tradeoff:	Não há

Atributo de Qualidade:	Conformidade
Requisito de Qualidade:	As API' deverão seguir o padrão REST e com documentação visível.
Preocupação:	
O sistema deverá ser facilmente integrável via REST com outros módulos do ERP e sistemas externos.	
Cenário(s):	
Cenário 4	
Ambiente:	
API e documentação Open API	
Estímulo:	
Acessar a URL da documentação do Open API.	
Mecanismo:	
Utilização de URL específica para acessar a documentação gerada pelo módulo Spring Doc Open API e validada via Postman.	
Medida de resposta:	
Chamadas REST válidas e retornos JSON válidos. Página com a documentação no padrão Open API.	



Considerações sobre a arquitetura:

Riscos:	Mudanças de API podem quebrar compatibilidade da integração com outros sistemas.
Pontos de Sensibilidade:	Mudanças na API.
Tradeoff:	Não há

Atributo de Qualidade:	Conformidade
Requisito de Qualidade:	O sistema deverá seguir as regras definidas pela LGPD
Preocupação:	
O sistema deverá proteger as informações sensíveis dos usuários, seguindo as regras da LGPD.	
Cenário(s):	

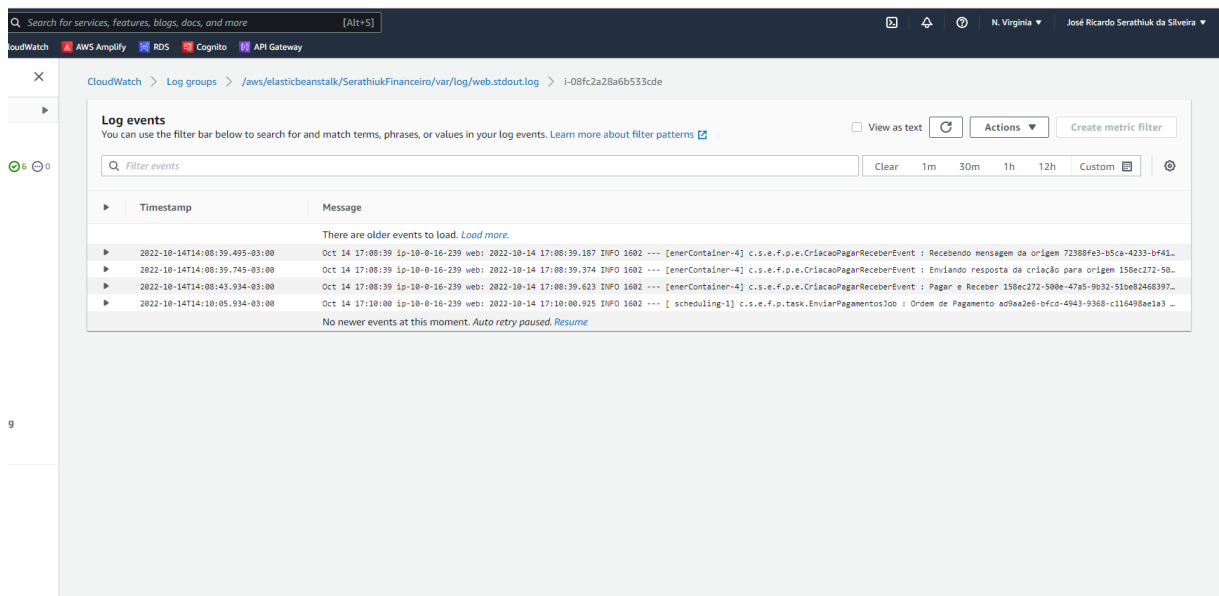
Cenário 5	
Ambiente:	
Utilização normal do sistema	
Estímulo:	
Checagem das informações retornas por API's com informações sensíveis	
Mecanismo:	
Checagem de segurança e conformidade periódicas. Restringir acesso a backups e chaves de acesso da cloud AWS.	
Medida de resposta:	
Não há.	
Considerações sobre a arquitetura:	
Riscos:	Vazamento de dados
Pontos de Sensibilidade:	Backups e acessos não autorizados
Tradeoff:	Não há

Atributo de Qualidade:	Rastreabilidade
Requisito de Qualidade:	O sistema deverá armazenar logs de operações relevantes do sistema.
Preocupação:	
O sistema deverá armazenar os erros, alertas e operações relevantes em logs, com data e descrição do evento que ocorreu.	
Cenário(s):	
Cenário 6	
Ambiente:	
Chamadas da API	
Estímulo:	
Chamadas da API simulando erros ou operações não permitidas, como chamadas REST com valores obrigatórios em branco ou chamadas com valores inválidos (valor numérico como string por exemplo)	
Mecanismo:	
Envio de logs para AWS Cloudwatch Logs via Agent AWS na instância ECs, que faz	

a leitura dos logs do Spring / Log4J em arquivos.

Medida de resposta:

Visualização do log no AWS Cloudwatch Logs



Considerações sobre a arquitetura:

Riscos:

O envio do log é feito de maneira paralela ao sistema, por mecanismos do AWS Beanstalk. Se a instância parar de responder por algum, pode ser que logs de momentos anteriores ao erro que o ocasionou a parada possam não ser enviados para o AWS Cloudwatch Logs.

Pontos de Sensibilidade:

Não há

Tradeoff:

Não há

Atributo de Qualidade:

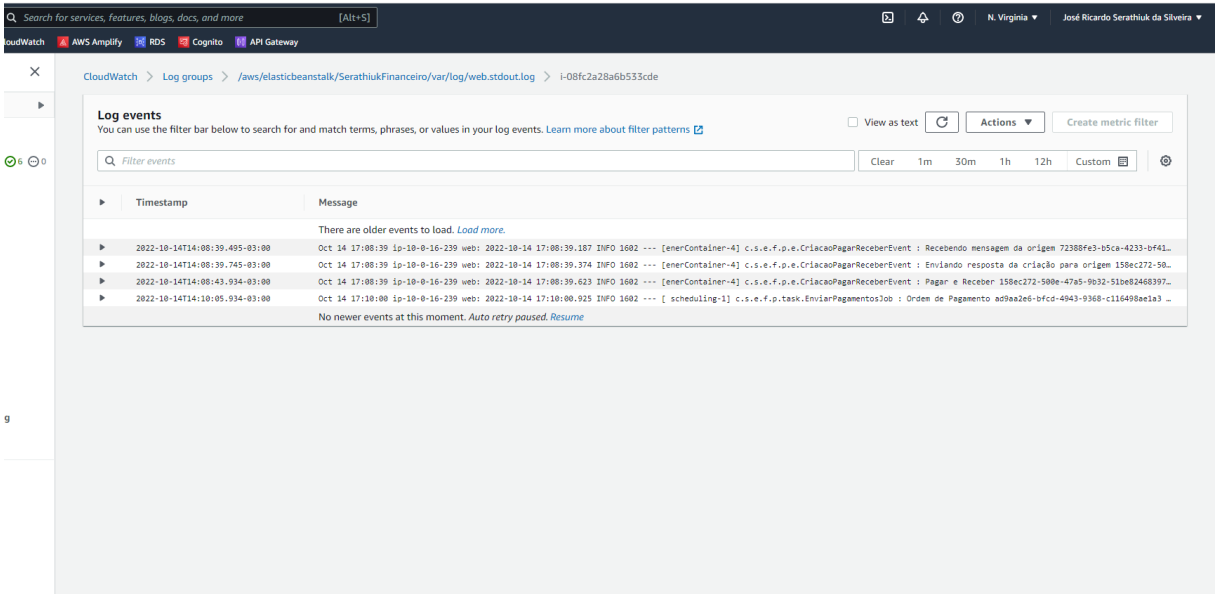
Rastreabilidade

Requisito de Qualidade:

O sistema deverá armazenar as mensagens/eventos entre sistema em log

Preocupação:

O sistema deverá manter as mensagens enviadas entre os serviços, para fins de auditoria ou até mesmo de rastreabilidade de possíveis problemas futuros.

Cenário(s):	
Cenário 7	
Ambiente:	
Mensagens entre serviços	
Estímulo:	
O sistema irá enviar uma mensagem/evento via um tópico AWS SNS.	
Mecanismo:	
Ao envia uma mensagem para um tópico AWS SNS, a mensagem irá para a(s) filas AWS SQS relativa(s) ao tópico, e também irão ser encaminhadas para um função AWS Lambda que irá registrar a mensagem em log.	
Medida de resposta:	
Visualização do log no AWS Cloudwatch Logs	
 <p>The screenshot shows the AWS CloudWatch Logs console. The breadcrumb navigation indicates the log group is <code>/aws/elasticbeanstalk/SerathiukFinanceiro/var/log/web.stdout.log</code> with log stream <code>i-08fc2a28a6b533cde</code>. The 'Log events' section displays a list of log entries. The first entry is a message: 'There are older events to load. Load more.' The subsequent entries are from an EC2 instance (ip-10-0-16-239 web) and show container events and payment processing logs. The messages include: 'Recebendo mensagem da origem 72388fe3-b5ca-4233-bf61...', 'Enviando resposta da criação para origem 158ec172-50...', 'Pagar e Receber 158ec172-508e-47a5-9a32-51b683468397...', and 'Ordem de Pagamento ad9a2e6-bfcd-4943-9368-c116498ae1a3...'.</p>	
Considerações sobre a arquitetura:	
Riscos:	Não há
Pontos de Sensibilidade:	Não há
Tradeoff:	A mensagem não pode ser reenviada via AWS SQS. Apenas fica registrada para fins de log.

6.4. Resultados Obtidos

Apresente os resultados da arquitetura produzida, indicando seus pontos fortes e suas limitações. A título de sugestão construa uma tabela apresentando esses resultados, como no exemplo que segue:

Requisitos Não Funcionais	Teste	Homologação
RNF01: O sistema deve ser apresentar disponibilidade de funcionamento mensal de 99%.	OK	OK
RNF02: O tempo máximo aceitável de duração de qualquer requisição síncrona a API será de 5 segundos.	OK	N.A.
RNF03: A interface frontend deverá ser compatível com os navegadores Google Chrome, Mozilla Firefox, Apple Safari, em versões com até 2 anos.	OK	OK
RNF04: O sistema deverá atender as normas legais descritas na LGPD (Lei Geral de Proteção de Dados)	OK	N.A.
RNF05: A API REST deverá ser pública e documentada para integrações externas.	OK	OK
RNF06: Todos os logs internos deverão ser acompanhados com a data e hora do evento e identificador de origem do evento (mensagem, usuário, ip, etc)	OK	OK
RNF07: O sistema deverá manter um registro dos eventos.	OK	OK

Obs: N.A.: não se aplica.

7. Avaliação Crítica dos Resultados

A arquitetura proposta atende os requisitos iniciais desse projeto, que é definir uma arquitetura padrão para serviços de uma futura refatoração de um projeto de software ERP. Os pontos previstos nesse projeto são tecnicamente viáveis. Porém, durante sua implementação, foram observados alguns pontos que podem ser melhorados.

Sobre a estrutura de mensageria/eventos. O AWS SQS se mostrou viável para a integração entre serviços. Porém para atender os requisitos não funcionais utilizando-o, foi utilizada uma solução de um agente externo, o AWS Lambda, apenas para salvar logs. Essa solução soa como workaround, pois não existe uma solução nativa do SQS/SNS para atender esses requisitos. Talvez como uma futura versão do protótipo, avaliar outras ferramentas de mensageria, que possam atender esse requisito de log.

O Cloudwatch é uma solução bem completa que atendeu boa parte dos requisitos não funcionais. Toda a parte de observabilidade da aplicação pode ser atingida utilizando essa ferramenta.

Por envolver apenas 1 módulo, não houve a oportunidade de testar transações distribuídas mais complexas, envolvendo mais de 2 módulos, talvez utilizando o padrão SAGA. Esse é um problema que deve ser avaliado, e que transações entre 2 serviços já trouxe um grau maior de complexidade, se comparada com uma solução de software monolito e pode ser um grande problema no futuro na medida que o número de serviços aumente.

Outras soluções da AWS, como o API Gateway e Cognito se mostraram de fácil utilização com o Spring Boot / Spring Framework. A solução AWS Beanstalk trouxe facilidade no momento de criar e gerenciar a infraestrutura.

No frontend utilização do Vue + PrimeVue se mostrou boa e traz um bom grau de produtividade para criação de telas.

8. Conclusão

Nesse trabalho foi apresentada a migração de 1 módulo de um software gestão, que servirá como protótipo para possíveis migrações futuras.

Mesmo tendo uma arquitetura simples, se for comparar uma arquitetura de mercado, deu para enxergar vários possíveis pontos de falhas e cuidados que devemos ter ao fazer uma migração nessa escala. Primeiramente, ficou claro, que não é uma tarefa simples. Transformar o monolito em serviços independentes pode aumentar significativamente a complexidade da manutenção e a gestão do desenvolvimento do software. Deu para entender que talvez seja melhor seja primeiro adotar uma arquitetura de monolito modular, para depois dar um salto software distribuído, já separando cada módulo com seus contextos e domínios. Um monolito modular iria atender parcialmente ou quase totalmente os objetivos descritos na introdução desse trabalho.

O trabalho foi importante, para aprender mais sobre software distribuído, sobre o Spring Framework e sobre AWS. E mais importante de tudo, deu muito conhecimento para ter mais pé no chão na hora de fazer uma migração dessa natureza e dessa complexidade. Deu para aprender o que fazer, e o principalmente o que não fazer.

A arquitetura se mostrou viável, porém para o número de clientes que o ERP tem hoje, migrar do monolito para um modelo distribuído sem um bom planejamento, pode trazer grandes problemas para os usuários do ERP e para a equipe que o desenvolve e mantém.

Referências

FOWLER, M. StranglerFigApplication. **Martin Fowler**, 2004. Disponível em: <<https://martinfowler.com/bliki/StranglerFigApplication.html>>. Acesso em: 10 fev. 2022.

FOWLER, M. **Refactoring**: Improving the Design of Existing Code. Second Edition. ed. [S.l.]: Addison-Wesley Professional, 2018.

NEWMAN, S. **Building Microservices**. 2nd Edition. ed. [S.l.]: O'Reilly Media, Inc., 2021.

WIKIPEDIA. Conway's law. **Wikipedia**, 2022. Disponível em:

<https://en.wikipedia.org/wiki/Conway%27s_law>. Acesso em: 10 fev. 2022.

YANAGA, E. **Migrating to Microservice Databases**. [S.l.]: O'Reilly Media, Inc., 2017.