

**UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC  
CENTRO DE EDUCAÇÃO SUPERIOR DO ALTO VALE DO ITAJAÍ – CEAVI  
ENGENHARIA DE SOFTWARE – ESO**

**DÊNIS DIEGO MARX, THIAGO ARTUR SCHUMANN**

**FINTRACKR**

**IBIRAMA**

**2023**

## SUMÁRIO

<b>1</b>	<b>DEFINIÇÃO DE ESCOPO DA APLICAÇÃO . . . . .</b>	<b>4</b>
1.1	OBJETIVOS DO PROJETO . . . . .	4
1.2	REQUISITOS . . . . .	5
<b>1.2.1</b>	<b>Requisitos Funcionais . . . . .</b>	<b>5</b>
<b>1.2.2</b>	<b>Requisitos Não Funcionais . . . . .</b>	<b>5</b>
1.3	REGRAS DE NEGÓCIO . . . . .	6
1.4	CASOS DE USO . . . . .	7
<b>2</b>	<b>ESPECIFICAÇÃO/ANÁLISE DA APLICAÇÃO . . . . .</b>	<b>8</b>
2.1	FORMATO DE ESPECIFICAÇÃO . . . . .	8
2.2	FERRAMENTAS DE ALM . . . . .	8
2.3	PLANEJAMENTO DO DESENVOLVIMENTO . . . . .	9
2.4	BACKLOG . . . . .	9
<b>3</b>	<b>DEFINIÇÃO DA ARQUITETURA . . . . .</b>	<b>11</b>
<b>3.0.1</b>	<b>Estilo de Arquitetura . . . . .</b>	<b>11</b>
<b>3.0.2</b>	<b>Padrões Arquiteturais . . . . .</b>	<b>11</b>
<b>3.0.3</b>	<b>SOLID . . . . .</b>	<b>12</b>
<b>3.0.4</b>	<b>Design Patterns . . . . .</b>	<b>13</b>
<b>3.0.5</b>	<b>Padrões de comunicação . . . . .</b>	<b>13</b>
<b>4</b>	<b>PIPELINE DE CI/CD . . . . .</b>	<b>15</b>
4.1	FERRAMENTA DE VERSIONAMENTO . . . . .	15
4.2	SERVIDOR DE CI/CD . . . . .	15
4.3	TECNOLOGIAS UTILIZADAS . . . . .	15
<b>4.3.1</b>	<b>Front-end . . . . .</b>	<b>16</b>
<b>4.3.2</b>	<b>Back-end Principal . . . . .</b>	<b>16</b>
<b>4.3.3</b>	<b>Microserviço de Processamento de Faturas . . . . .</b>	<b>16</b>
<b>4.3.4</b>	<b>Banco de Dados . . . . .</b>	<b>16</b>
4.4	FERRAMENTAS DE BUILD . . . . .	16
4.5	DEFINIÇÃO DE FERRAMENTAS E ESTRUTURA DE EXECUÇÃO PARA DEPLOY . . . . .	17
4.6	FERRAMENTAS PARA DEPLOY . . . . .	17
4.7	ESTRUTURA DE EXECUÇÃO . . . . .	17
<b>4.7.1</b>	<b>Contêinerização e Isolamento . . . . .</b>	<b>18</b>
<b>4.7.2</b>	<b>Desenvolvimento e Testes . . . . .</b>	<b>18</b>
<b>4.7.3</b>	<b>Integração e Entrega Contínua . . . . .</b>	<b>18</b>
<b>4.7.4</b>	<b>Execução . . . . .</b>	<b>18</b>

4.8	GITFLOW . . . . .	19
4.9	TESTES . . . . .	20
<b>4.9.1</b>	<b>Ferramentas de Testes . . . . .</b>	<b>20</b>
<b>4.9.2</b>	<b>Metrica de Cobertura de Testes . . . . .</b>	<b>20</b>
4.10	ARTIFACTS MANAGEMENT . . . . .	20
4.11	DOCUMENTAÇÃO . . . . .	21
	<b>APÊNDICE A – UC01 - AUTENTICAÇÃO NO SISTEMA . . . . .</b>	<b>22</b>
	<b>APÊNDICE B – UC02 - GERENCIAR PERFIL . . . . .</b>	<b>24</b>
	<b>APÊNDICE C – UC03 - REGISTRAR E CATEGORIZAR TRANSA- ÇÕES . . . . .</b>	<b>26</b>
	<b>APÊNDICE D – UC04 - GERENCIAR CATEGORIAS . . . . .</b>	<b>28</b>
	<b>APÊNDICE E – UC05 - GERENCIAR ORÇAMENTOS . . . . .</b>	<b>30</b>
	<b>APÊNDICE F – UC06 - VISUALIZAR DASHBOARD FINANCEIRO . . . . .</b>	<b>32</b>
	<b>APÊNDICE G – UC07 - IMPORTAR FATURAS . . . . .</b>	<b>34</b>
	<b>APÊNDICE H – UC08 - GERENCIAR CARTÕES E FATURAS . . . . .</b>	<b>36</b>
	<b>APÊNDICE I – UC09 - ACOMPANHAR EVOLUÇÃO DE SALDO . . . . .</b>	<b>38</b>
	<b>APÊNDICE J – UC10 - GERENCIAR BANCOS . . . . .</b>	<b>40</b>
	<b>APÊNDICE K – UC11 - DEFINIR MÊS DE REFERÊNCIA PARA TRANSAÇÕES . . . . .</b>	<b>42</b>
	<b>APÊNDICE L – ADR-001 - TENTATIVA DE IMPLEMENTAR TODO O PROJETO DO <i>FINTRACKR</i> USANDO MICRO- SERVIÇOS . . . . .</b>	<b>44</b>
	<b>APÊNDICE M – ADR-002 - ADOÇÃO DE ARQUITETURA MONO- LÍTICA PARA A LÓGICA DE NEGÓCIOS PRINCIPAL DO <i>FINTRACKR</i> . . . . .</b>	<b>45</b>
	<b>APÊNDICE N – ADR-003 - CRIAÇÃO DE MICROSERVIÇO ESPE- CIALIZADO PARA PROCESSAMENTO DE PLA- NILHAS NO <i>FINTRACKR</i> . . . . .</b>	<b>47</b>
	<b>APÊNDICE O – ADR-004 - ADOÇÃO DE ARQUITETURA MONO- LÍTICA NO FRONT-END DO <i>FINTRACKR</i> . . . . .</b>	<b>49</b>
	<b>APÊNDICE P – ADR-005 - FRONT-END REACT - COMPONENT- BASED ARCHITECTURE . . . . .</b>	<b>50</b>
	<b>APÊNDICE Q – ADR-006 - BACK-END FLASK - CAMADAS COM CLEAN/ONION . . . . .</b>	<b>51</b>
	<b>APÊNDICE R – ADR-007 - MICROSERVIÇO COM PANDAS - HE- XAGONAL . . . . .</b>	<b>52</b>

APÊNDICE S – ADR-008 - ADOÇÃO DA ARQUITETURA REST PARA COMUNICAÇÃO ENTRE FRONT-ENDE E BACK- END FLASK . . . . .	53
APÊNDICE T – ADR-009 - ADOÇÃO DO RABBITMQ PARA COMU- NICAÇÃO ENTRE O MICROSERVIÇO DA PLANI- LHA DE FATURA E O BACK-END FLASK . . . . .	54
APÊNDICE U – ADR-010 - ADOÇÃO DO SQLALCHEMY PARA COMUNICAÇÃO COM O BANCO DE DADOS NO FLASK . . . . .	55

# 1 DEFINIÇÃO DE ESCOPO DA APLICAÇÃO

## 1.1 OBJETIVOS DO PROJETO

O *FinTrackr* é um aplicativo web responsivo de gestão de despesas pessoais, desenvolvido para proporcionar aos usuários uma ferramenta eficiente para o controle e análise de finanças. O aplicativo visa simplificar o processo de registro de receitas e despesas, oferecendo uma categorização detalhada de cada transação. Uma das principais características do *FinTrackr* é a habilidade de importar planilhas de faturas de cartão de crédito, com o Bradesco servindo como banco piloto para esta funcionalidade.

O *FinTrackr* garante uma experiência de usuário consistente e fluida, sendo acessível por uma variedade de dispositivos graças ao seu design responsivo. O back-end do aplicativo é otimizado para assegurar um desempenho eficiente e uma comunicação ágil com o banco de dados, permitindo escalabilidade e flexibilidade na manipulação de dados.

Além do registro de transações, o aplicativo disponibiliza recursos adicionais valiosos, como a criação de orçamentos, geração de relatórios financeiros visuais e a capacidade de atribuir múltiplas categorias a uma única transação. Estas funcionalidades são fundamentais para ajudar os usuários a monitorar seus orçamentos, identificar padrões de despesa, e assim fornecer informações cruciais para uma decisão financeira informada e estratégica.

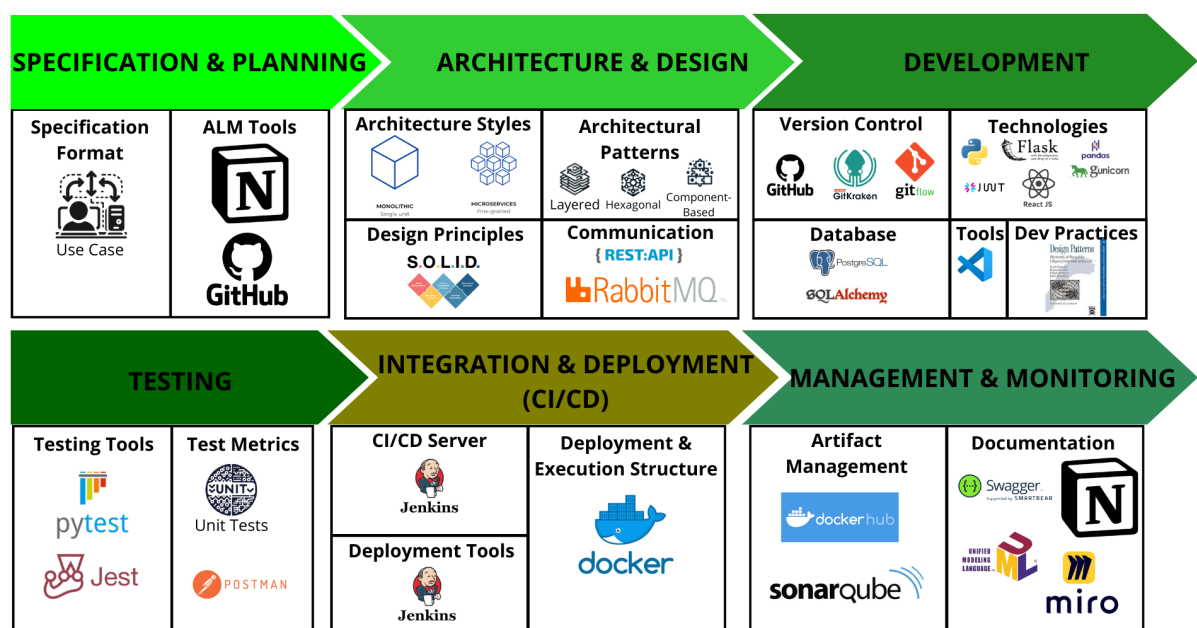


Figura 1 – tecnologias, técnicas, design, estilo arquiteturas

## 1.2 REQUISITOS

### 1.2.1 Requisitos Funcionais

1. **RF01:** Implementação de um sistema de autenticação robusto e seguro, permitindo registro, login, recuperação de senha e gerenciamento de perfil.
2. **RF02:** Facilitar o gerenciamento de informações do perfil do usuário, incluindo visualização, adição, atualização e exclusão de dados como nome, endereço e telefone.
3. **RF03:** Permitir o registro de transações financeiras, categorizando-as como receitas ou despesas e associando detalhes pertinentes.
4. **RF04:** Habilitar a criação, edição e exclusão de categorias de transações, associando detalhes como nome, cor e ícone.
5. **RF05:** Proporcionar a definição e monitoramento de orçamentos mensais por categoria, mostrando gastos realizados e disponíveis.
6. **RF06:** Prover um dashboard que apresente um resumo financeiro, incluindo saldo em contas, resumo de orçamentos, gastos por categoria e balanço geral.
7. **RF07:** Integrar com um microserviço para permitir a importação de faturas bancárias em formato .csv, iniciando pelo Bradesco.
8. **RF08:** Facilitar o gerenciamento de cartões de crédito, incluindo registro de faturas e lançamentos associados.
9. **RF09:** Oferecer uma visualização detalhada do histórico de saldo, mostrando a evolução financeira do usuário.
10. **RF10:** Permitir o registro e gerenciamento de bancos associados às contas do usuário.

### 1.2.2 Requisitos Não Funcionais

1. **RNF01:** A interface do sistema deve ser intuitiva e organizada.
2. **RNF02:** Adaptabilidade para diferentes dispositivos, mantendo funcionalidade e design em smartphones, tablets e desktops.
3. **RNF03:** Fornecer feedback claro e direto ao usuário em todas as interações.
4. **RNF04:** Código-fonte bem estruturado, seguindo padrões reconhecidos de programação.
5. **RNF05:** As operações do sistema devem ser otimizadas para resposta rápida, garantindo eficiência.

### 1.3 REGRAS DE NEGÓCIO

1. **RN01:** Transações devem ser associadas a uma categoria.
2. **RN02:** Cartões e contas com faturas ou transações associadas não podem ser excluídos.
3. **RN03:** Categorias com transações associadas não podem ser excluídas.
4. **RN04:** Ao dividir despesas em múltiplas categorias, o valor associado a cada categoria deve ser especificado.
5. **RN05:** O total de valores associados em uma transação dividida entre categorias deve igualar o valor total da transação.
6. **RN06:** Orçamentos não podem ter valores negativos.
7. **RN07:** Despesas em cartões de crédito devem ser associadas à fatura corrente.
8. **RN08:** Transações em contas impactam o saldo da mesma.
9. **RN09:** Transações não podem ter datas futuras.
10. **RN10:** Importações de faturas via .csv devem permitir edição pós-importação.
11. **RN11:** Faturas de cartões possuem status baseados em datas de vencimento e pagamentos.
12. **RN12:** Usuários devem ser notificados ao atingir ou exceder orçamentos.
13. **RN13:** Categorias não podem ser duplicadas em uma transação dividida.
14. **RN14:** Saldos negativos em contas devem ser prevenidos, a menos que permitidos.
15. **RN15:** Informações de cartões ou contas duplicadas não são permitidas.
16. **RN16:** Os usuários devem fornecer um email e senha válidos para se autenticar no sistema.
17. **RN17:** Senhas devem ter um mínimo de 8 caracteres e conter pelo menos uma letra maiúscula, uma letra minúscula, um número e um caractere especial.
18. **RN18:** Para recuperação de senha, o usuário deve confirmar sua identidade através de um link enviado ao seu email registrado.
19. **RN19:** O email fornecido pelo usuário durante o registro ou atualização do perfil deve ser único no sistema.
20. **RN20:** Para alterar a senha, o usuário deve fornecer a senha atual para confirmação.
21. **RN21:** Todas as transações devem ter um mês de referência para controle orçamentário.

## 1.4 CASOS DE USO

### **UC01: Autenticação no Sistema**

Apêndice A

### **UC02: Gerenciar Perfil**

Apêndice B

### **UC03: Registrar e Categorizar Transações**

Apêndice C

### **UC04: Gerenciar Categorias**

Apêndice D

### **UC05: Gerenciar Orçamentos**

Apêndice E

### **UC06: Visualizar Dashboard Financeiro**

Apêndice F

### **UC07: Importar Faturas**

Apêndice G

### **UC08: Gerenciar Cartões e Faturas**

Apêndice H

### **UC09: Acompanhar Evolução de Saldo**

Apêndice I

### **UC10: Gerenciar Bancos**

Apêndice J

### **UC11: Definir Mês de Referência para Transações**

Apêndice K



## 2 ESPECIFICAÇÃO/ANÁLISE DA APLICAÇÃO

### 2.1 FORMATO DE ESPECIFICAÇÃO

Critério	Formato de Especificação		
	Casos de Uso	Casos de Uso 2.0	Histórias de Usuário
<b>Quando Usar</b>	Projetos que necessitam de uma visão abrangente e detalhada do sistema.	Quando você precisa de um formato estruturado e iterativo, adaptável a mudanças.	Desenvolvimento Ágil, projetos menores a médios, ou quando o foco está nas necessidades do usuário final.
<b>Benefícios</b>	Oferece alto nível de detalhamento e uma visão abrangente do sistema.	Oferece uma estrutura detalhada, é iterativo, flexível e adaptável.	Simples, focado, altamente adaptável a mudanças, fácil de aprender e usar, e com baixo custo de manutenção.
<b>Desvantagens</b>	Pode ser complexo e demorado para desenvolver e manter. Ideal para sistemas maiores.	Requer planejamento cuidadoso e tem custo de manutenção moderado.	Pode ser simplista para funcionalidades complexas.

Tabela 1 – Comparação revisada entre Casos de Uso, Casos de Uso 2.0 e Histórias de Usuário

O projeto *Fintrackr*, embora seja de pequeno porte, tem uma complexidade inerente que demanda uma visão mais detalhada e estruturada do sistema. Neste cenário, os *Casos de Uso* surgem como o formato de especificação ideal. A abordagem de Casos de Uso permite uma representação mais profunda das interações do sistema, alinhando-se perfeitamente com a Clean Architecture ao focar nas interações e nos fluxos de negócios.

Os Casos de Uso oferecem uma representação clara das funções e interações do sistema, facilitando a comunicação entre desenvolvedores, stakeholders e usuários. Esta abordagem detalhada garante que todas as nuances e requisitos do *Fintrackr* sejam capturados e desenvolvidos de forma eficaz. Ao adotar Casos de Uso, garantimos uma compreensão profunda das necessidades do usuário e uma implementação sistemática dessas necessidades.

### 2.2 FERRAMENTAS DE ALM

Critério	Ferramenta				
	Jira	Trello	Tuleap	GitLab	Notion
<b>Quando Usar</b>	Projetos ágeis, de médio a grande porte.	Projetos simples, visualização Kanban.	Desenvolvimento ágil, integração contínua, projetos open source.	Desenvolvimento e integração contínua.	Geração e controle de tarefas, visões múltiplas, do mesmo datasource.
<b>Benefícios</b>	Flexível, integrações amplas, poderoso.	Simples, intuitivo, flexível.	Open source, personalizável.	Integração contínua, gestão de código, gerenciamento de projeto.	Facilidade na geração e controle de tarefas.
<b>Desvantagens</b>	Curva de aprendizado íngreme, pode ser caro.	Pode ser simplista para grandes projetos.	Pode ser complexo para novatos.	Pode ser sobrecarregado para projetos menores.	-

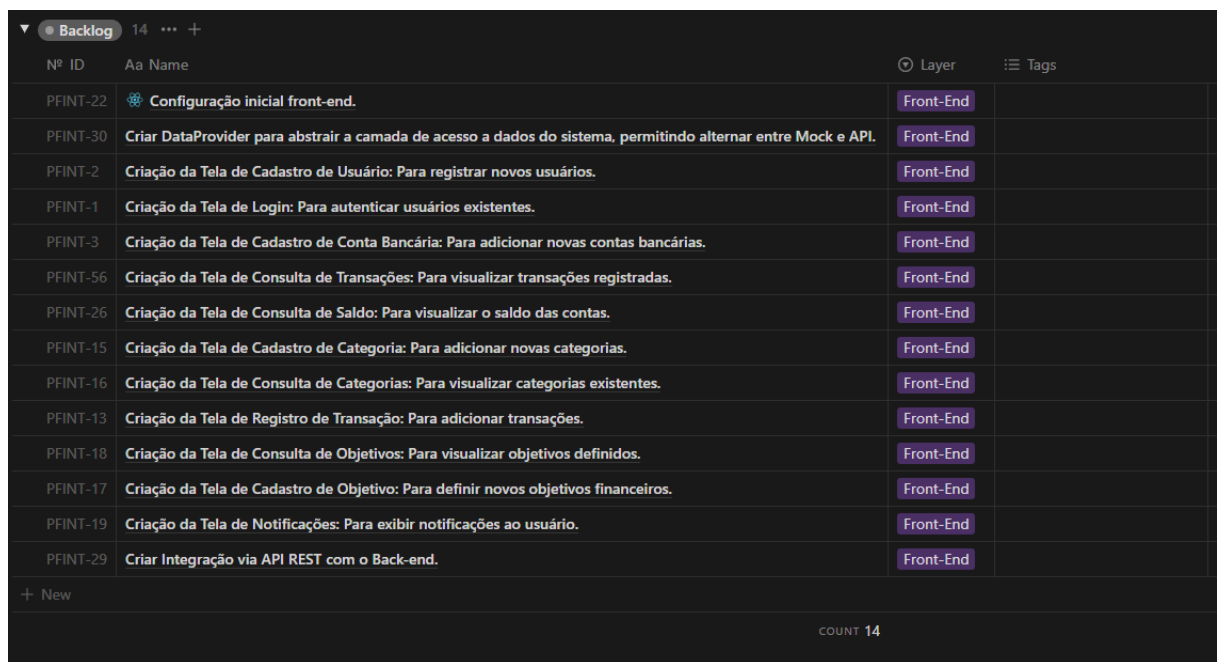
Tabela 2 – Comparação entre Jira, Trello, Tuleap, GitLab e Notion

Dada a natureza do projeto *Fintrackr*, a seleção de ferramentas para gerenciamento e desenvolvimento é crucial. Optamos pelo Notion devido à sua facilidade na geração e controle de tarefas, permitindo várias visões diferentes do mesmo datasource. Além disso, o Notion também

nos permite criar documentações detalhadas para nossos projetos. Complementando nossa escolha, adotamos um repositório no GitHub para centralizar outros aspectos da documentação e artefatos, oferecendo uma visão unificada do projeto. Essa combinação de Notion e GitHub é ideal para as necessidades do *Fintrackr*.

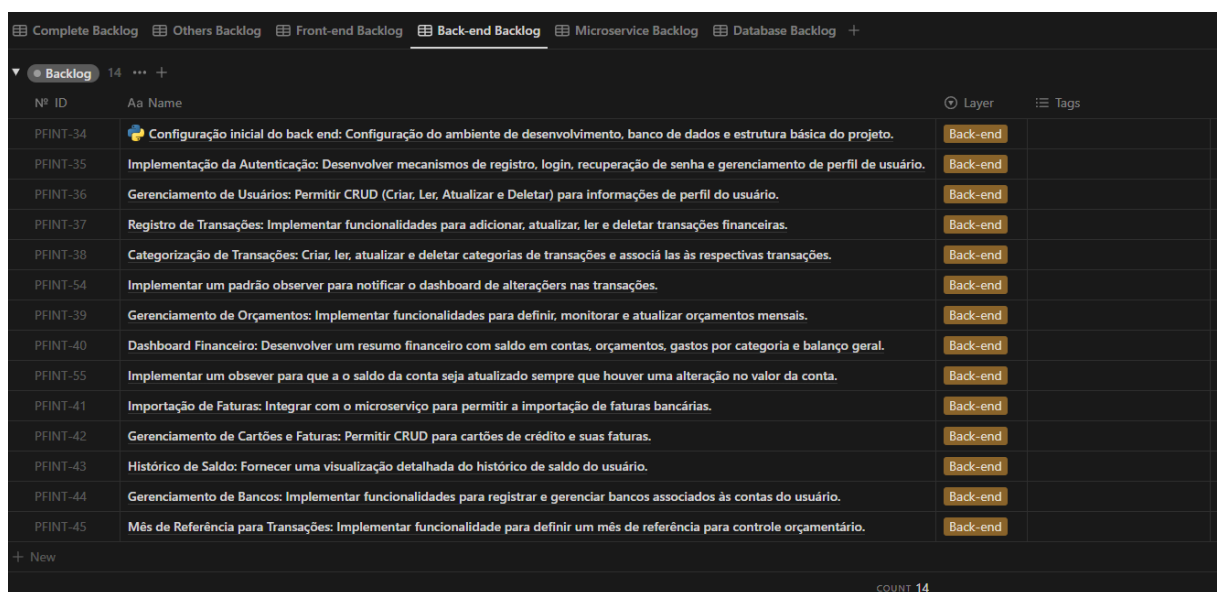
## 2.3 PLANEJAMENTO DO DESENVOLVIMENTO

## 2.4 BACKLOG



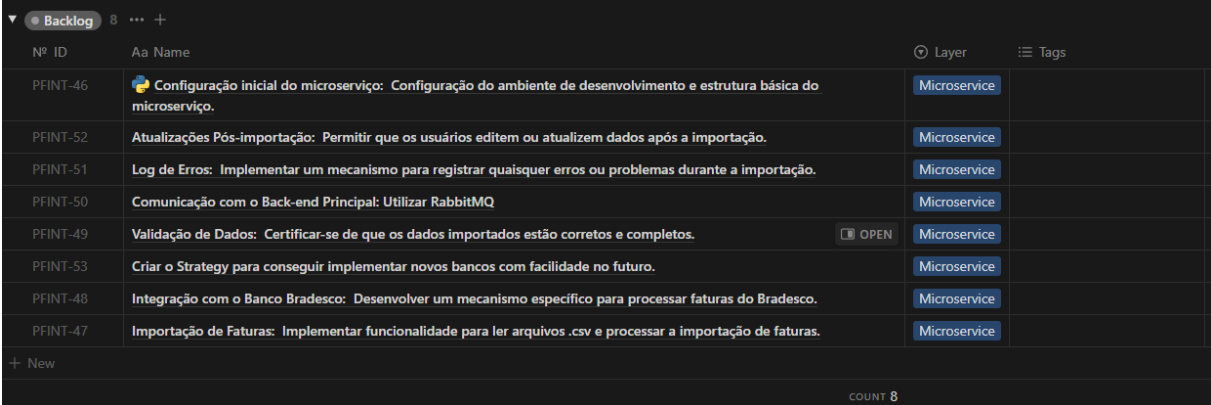
Nº	ID	Aa Name	Layer	Tags
PFINT-22		Configuração inicial front-end.	Front-End	
PFINT-30		Criar DataProvider para abstrair a camada de acesso a dados do sistema, permitindo alternar entre Mock e API.	Front-End	
PFINT-2		Criação da Tela de Cadastro de Usuário: Para registrar novos usuários.	Front-End	
PFINT-1		Criação da Tela de Login: Para autenticar usuários existentes.	Front-End	
PFINT-3		Criação da Tela de Cadastro de Conta Bancária: Para adicionar novas contas bancárias.	Front-End	
PFINT-56		Criação da Tela de Consulta de Transações: Para visualizar transações registradas.	Front-End	
PFINT-26		Criação da Tela de Consulta de Saldo: Para visualizar o saldo das contas.	Front-End	
PFINT-15		Criação da Tela de Cadastro de Categoria: Para adicionar novas categorias.	Front-End	
PFINT-16		Criação da Tela de Consulta de Categorias: Para visualizar categorias existentes.	Front-End	
PFINT-13		Criação da Tela de Registro de Transação: Para adicionar transações.	Front-End	
PFINT-18		Criação da Tela de Consulta de Objetivos: Para visualizar objetivos definidos.	Front-End	
PFINT-17		Criação da Tela de Cadastro de Objetivo: Para definir novos objetivos financeiros.	Front-End	
PFINT-19		Criação da Tela de Notificações: Para exibir notificações ao usuário.	Front-End	
PFINT-29		Criar Integração via API REST com o Back-end.	Front-End	
+ New				
			COUNT 14	

Figura 2 – Backlog referente ao Front-end.



Nº	ID	Aa Name	Layer	Tags
PFINT-34		Configuração inicial do back end: Configuração do ambiente de desenvolvimento, banco de dados e estrutura básica do projeto.	Back-end	
PFINT-35		Implementação da Autenticação: Desenvolver mecanismos de registro, login, recuperação de senha e gerenciamento de perfil de usuário.	Back-end	
PFINT-36		Gerenciamento de Usuários: Permitir CRUD (Criar, Ler, Atualizar e Deletar) para informações de perfil do usuário.	Back-end	
PFINT-37		Registro de Transações: Implementar funcionalidades para adicionar, atualizar, ler e deletar transações financeiras.	Back-end	
PFINT-38		Categorização de Transações: Criar, ler, atualizar e deletar categorias de transações e associá-las às respectivas transações.	Back-end	
PFINT-54		Implementar um padrão observer para notificar o dashboard de alterações nas transações.	Back-end	
PFINT-39		Gerenciamento de Orçamentos: Implementar funcionalidades para definir, monitorar e atualizar orçamentos mensais.	Back-end	
PFINT-40		Dashboard Financeiro: Desenvolver um resumo financeiro com saldo em contas, orçamentos, gastos por categoria e balanço geral.	Back-end	
PFINT-55		Implementar um observer para que o saldo da conta seja atualizado sempre que houver uma alteração no valor da conta.	Back-end	
PFINT-41		Importação de Faturas: Integrar com o microserviço para permitir a importação de faturas bancárias.	Back-end	
PFINT-42		Gerenciamento de Cartões e Faturas: Permitir CRUD para cartões de crédito e suas faturas.	Back-end	
PFINT-43		Histórico de Saldo: Fornecer uma visualização detalhada do histórico de saldo do usuário.	Back-end	
PFINT-44		Gerenciamento de Bancos: Implementar funcionalidades para registrar e gerenciar bancos associados às contas do usuário.	Back-end	
PFINT-45		Mês de Referência para Transações: Implementar funcionalidade para definir um mês de referência para controle orçamentário.	Back-end	
+ New				
			COUNT 14	

Figura 3 – Backlog referente ao Back-end Principal.



Backlog 8 ... +			
Nº ID	Aa Name	Layer	Tags
PFINT-46	🔧 Configuração Inicial do microserviço: Configuração do ambiente de desenvolvimento e estrutura básica do microserviço.	Microservice	
PFINT-52	Atualizações Pós-importação: Permitir que os usuários editem ou atualizem dados após a importação.	Microservice	
PFINT-51	Log de Erros: Implementar um mecanismo para registrar quaisquer erros ou problemas durante a importação.	Microservice	
PFINT-50	Comunicação com o Back-end Principal: Utilizar RabbitMQ	Microservice	
PFINT-49	Validação de Dados: Certificar-se de que os dados importados estão corretos e completos. <span>OPEN</span>	Microservice	
PFINT-53	Criar o Strategy para conseguir implementar novos bancos com facilidade no futuro.	Microservice	
PFINT-48	Integração com o Banco Bradesco: Desenvolver um mecanismo específico para processar faturas do Bradesco.	Microservice	
PFINT-47	Importação de Faturas: Implementar funcionalidade para ler arquivos .csv e processar a importação de faturas.	Microservice	
+ New			
COUNT 8			

Figura 4 – Backlog referente ao Microserviço.

### 3 DEFINIÇÃO DA ARQUITETURA

#### Architecture Decision Record (ADR)

As ADRs (Documentos de Decisão de Arquitetura) implementadas neste projeto foram desenvolvidas seguindo o modelo proposto por Michael Nygard, disponível no GitHub. Este modelo fornece uma estrutura clara e concisa para documentar decisões arquiteturais cruciais, garantindo consistência e compreensibilidade em todo o projeto.

#### 3.0.1 Estilo de Arquitetura

*ADR-001 - Tentativa de Implementar Todo o Projeto do Fintrackr Usando Microserviços*

Apêndice L

*ADR-002 - Adoção de Arquitetura Monolítica para a Lógica de Negócios Principal do Fintrackr*

Apêndice M

*ADR-003 - Criação de Microserviço Especializado para Processamento de Planilhas no Fintrackr*

Apêndice N

*ADR-004 - Adoção de Arquitetura Monolítica no Front-end do Fintrackr*

Apêndice O

#### 3.0.2 Padrões Arquiteturais

*ADR-005 - Front-end React - Component-Based Architecture*

Apêndice P

*ADR-006 - Back-end Flask - Camadas com Clean/Onion*

Apêndice Q

*ADR-007 - Microserviço com Pandas - Hexagonal*

Apêndice R

### 3.0.3 SOLID

#### *S - Single Responsibility Principle (Princípio da Responsabilidade Única)*

Sugere que uma classe ou módulo deve ter apenas uma razão para mudar, ou seja, deve ter apenas uma responsabilidade. Isso facilita a manutenção, testabilidade e compreensão do código. Dentro da arquitetura em camadas e do modelo Clean/Onion, é vital que cada camada ou serviço do FinTrackr tenha uma responsabilidade clara. Por exemplo, no contexto de um monólito, a camada de acesso a dados deve focar estritamente na persistência e recuperação de dados, enquanto a lógica de negócios reside em uma camada separada, desacoplada de detalhes específicos de banco de dados ou da apresentação.

#### *O - Open/Closed Principle (Princípio do Aberto/Fechado)*

Propõe que entidades de software (como classes ou módulos) devem estar abertas para extensão, mas fechadas para modificação. Isso permite que novas funcionalidades sejam adicionadas sem alterar o código existente. Na arquitetura Clean/Onion, a lógica central do FinTrackr deve ser projetada de forma que novas funcionalidades ou regras de negócios possam ser adicionadas sem alterar o núcleo existente. Isso é especialmente útil em um ambiente de microserviços, onde novos serviços podem ser introduzidos sem perturbar os já existentes.

#### *L - Liskov Substitution Principle (Princípio da Substituição de Liskov)*

Estabelece que subtipos devem ser substituíveis por seus tipos-base sem alterar a correção do programa. Independentemente da arquitetura adotada, seja monolítica ou baseada em microserviços, é crucial garantir que qualquer extensão ou modificação de entidades respeite este princípio. Por exemplo, se o FinTrackr introduzir uma nova variação de uma conta, qualquer instância dessa variação deve ser tratável como uma "conta" no sistema.

#### *I - Interface Segregation Principle (Princípio da Segregação de Interfaces)*

Sugere que nenhuma classe deve ser forçada a implementar interfaces das quais não precisa. Na arquitetura Clean/Onion, isso se traduz em criar contratos claros e específicos entre as camadas, garantindo que a lógica de negócios se comunique apenas com as operações de que realmente precisa. Em uma abordagem de microserviços, cada serviço deve expor apenas as operações relevantes para seu domínio.

#### *D - Dependency Inversion Principle (Princípio da Inversão de Dependência)*

Propõe que módulos de alto nível não devem depender de módulos de baixo nível, mas ambos devem depender de abstrações. Na arquitetura Clean/Onion, a lógica de negócios no núcleo não deve depender diretamente de detalhes externos, como bancos de dados ou APIs. Isso se alinha bem com o uso de ORMs, que fornecem uma abstração sobre o banco de dados. Em

uma abordagem de microserviços, os serviços devem comunicar-se por meio de contratos bem definidos.

### 3.0.4 Design Patterns

#### Padrão Observer para a camada Transação

##### *Funcionamento*

O padrão Observer é um padrão comportamental que define uma dependência entre objetos de modo que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente. O padrão é composto por um *Subject* que mantém uma lista de seus dependentes, chamados *Observers*, e os notifica sobre quaisquer mudanças de estado.

##### *Aplicação ao FinTrackr*

No contexto do FinTrackr, o módulo "Transação" atua como o *Subject*. Sua responsabilidade é notificar os usuários sobre transações que vencem no dia atual. Os módulos "Notificações por E-mail", "Notificações Push" e "Alertas no Dashboard" atuam como *Observers*. Diariamente, o sistema verifica todas as transações com vencimento no dia atual e, para cada transação identificada, o módulo "Transação" notifica todos os observadores registrados.

#### Padrão Strategy para Importação de Faturas

##### *Funcionamento*

O padrão Strategy define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis. Esse padrão permite que o algoritmo varie independentemente dos clientes que o utilizam, promovendo uma separação entre a definição de estratégias e sua execução no contexto.

##### *Aplicação ao FinTrackr*

No microserviço de "Importação de Faturas" do FinTrackr, o padrão Strategy é implementado por meio de uma interface chamada *ImportStrategy*. Essa interface define um contrato comum para todos os algoritmos de importação. Implementações concretas, como *BradescoImportStrategy*, definem algoritmos específicos para diferentes bancos. Quando uma requisição de importação é recebida, o microserviço determina qual estratégia usar e processa a importação conforme definido por essa estratégia.

### 3.0.5 Padrões de comunicação

*ADR-008 - Adoção da Arquitetura REST para Comunicação entre Front-end e Back-end Flask*

*ADR-009 - Adoção da Arquitetura REST para Comunicação entre Microserviços*

Apêndice T

*ADR-010 - Adoção do SQLAlchemy para Comunicação com o Banco de Dados no Flask*

Apêndice U

## 4 PIPELINE DE CI/CD

### 4.1 FERRAMENTA DE VERSIONAMENTO

Plataforma	Comparação		
	GitHub	GitLab	Bitbucket
<b>Benefícios</b>	Comunidade ampla, Marketplace extenso.	CI/CD integrado, Privacidade gratuita.	Integração Atlassian, Repositórios privados gratuitos.
<b>Desvantagens</b>	Preço por repositório privado, Limitações de armazenamento.	Interface menos intuitiva, Recursos de hardware.	Limitações em repositórios gratuitos, Menos recursos de CI/CD.
<b>Uso Comum</b>	Projetos open-source, Colaboração em larga escala.	Solução completa de DevOps, Controle total sobre CI/CD.	Equipes usando Atlassian. Integração de gerenciamento de projetos.

Tabela 3 – Comparação entre GitHub, GitLab e Bitbucket

Será utilizado o GitHub para controle de versões e colaboração eficiente no desenvolvimento de software. Ele fornece um ambiente centralizado para armazenar, acompanhar e gerenciar código-fonte, facilita a colaboração de equipes distribuídas e ajuda a controlar mudanças e atualizações no código, tornando o processo de desenvolvimento mais organizado, transparente e eficaz.

### 4.2 SERVIDOR DE CI/CD

Critério	Servidor de CI/CD			
	Jenkins	TravisCI	CircleCI	AWS CodePipeline
<b>Quando Usar</b>	Projetos grandes com necessidade de customização.	Projetos open-source com diferentes ambientes.	Projetos pequenos com integração rápida.	Projetos na AWS com serviços AWS integrados.
<b>Benefícios</b>	Alta customização, pipelines codificáveis.	Integração rápida, fácil configuração.	Integração rápida, fácil configuração.	Integração com serviços AWS.
<b>Desvantagens</b>	Complexo e demorado para configurar.	Ideal para projetos menores.	Pode ser limitado para projetos grandes.	Menos flexível com integrações de terceiros.

Tabela 4 – Comparação entre Servidores de CI/CD

A escolha do Jenkins como nossa ferramenta de CI/CD é estratégica devido à sua natureza de código aberto e à familiaridade preexistente da nossa equipe com ela. O caráter open source do Jenkins alinha-se com nosso valor de promover a transparência e flexibilidade, enquanto a familiaridade da equipe acelera a integração e minimiza a curva de aprendizado. Com uma comunidade ativa e um ecossistema robusto, o Jenkins oferece um ambiente confiável para automatizar nossos processos de desenvolvimento, tornando-se uma escolha sólida para nosso projeto.

### 4.3 TECNOLOGIAS UTILIZADAS

Neste projeto, empregamos uma combinação de linguagens de programação, frameworks, bibliotecas e sistemas de gerenciamento de banco de dados para construir e gerenciar o sistema em diferentes camadas. A seguir, descrevemos as tecnologias utilizadas em cada camada:



#### 4.3.1 Front-end

- **Linguagem:** JavaScript - Uma linguagem de programação amplamente adotada para desenvolvimento web.
- **Framework:** React.js - Um popular framework JavaScript para construir interfaces de usuário interativas.
- **Estilo:** Bootstrap - Um framework CSS para desenvolver sites responsivos e projetos web móveis de primeira classe.

#### 4.3.2 Back-end Principal

- **Linguagem:** Python - Conhecida por sua legibilidade e eficiência.
- **Framework:** Flask - Um microframework web para Python.
- **Autenticação:** Flask-JWT-Extended - Uma extensão do Flask para lidar com a autenticação JWT (JSON Web Token).

#### 4.3.3 Microserviço de Processamento de Faturas

- **Linguagem:** Python.
- **Biblioteca de Análise de Dados:** Pandas - Usada para manipulação e análise eficiente dos dados das faturas.
- **Mensageria:** RabbitMQ - Facilita a comunicação assíncrona entre o microserviço e o back-end monolítico.

#### 4.3.4 Banco de Dados

- **SGBD:** PostgreSQL - Um sistema de gerenciamento de banco de dados relacional poderoso e de código aberto.

### 4.4 FERRAMENTAS DE BUILD

A seleção de uma ferramenta adequada para gerenciamento de dependências e construção do projeto é crucial para a eficiência do processo de desenvolvimento. Maven, Gradle e Docker são três tecnologias populares, cada uma com suas peculiaridades. Maven é conhecido por sua simplicidade e reprodutibilidade, Gradle por sua flexibilidade e desempenho, enquanto Docker destaca-se na criação e distribuição de contêineres. A tabela abaixo apresenta uma comparação entre essas tecnologias, visando auxiliar na escolha consciente da ferramenta mais alinhada às necessidades do projeto.

Critério	Tecnologia		
	Maven	Gradle	Docker
<b>Quando Usar</b>	Projetos com estruturação clara.	Projetos que necessitam de flexibilidade.	Quando a criação e distribuição de contêineres é necessária.
<b>Benefícios</b>	Simplicidade, reprodutibilidade.	Flexibilidade, desempenho.	Portabilidade, isolamento.
<b>Desvantagens</b>	Menos flexível, pode ser lento.	Curva de aprendizado inicial.	Requer conhecimento em contêineres.

Tabela 5 – Comparação entre Maven, Gradle e Docker

Contudo, optamos em utilizar o Docker como ferramenta de build, por oferecer vantagens como a criação de ambientes de construção consistentes, isolados e portáteis. Isso garante a reprodutibilidade, a segurança e a facilidade de integração com CI/CD. A decisão de usar o Docker depende do contexto e dos requisitos do projeto.

#### 4.5 DEFINIÇÃO DE FERRAMENTAS E ESTRUTURA DE EXECUÇÃO PARA DEPLOY

A implantação eficaz do *Fintrackr* é crucial para garantir uma experiência de usuário contínua e robusta. Para alcançar essa eficiência, adotamos um conjunto de ferramentas líderes de mercado e definimos uma estrutura de execução que prioriza a escalabilidade, consistência e facilidade de manutenção.

#### 4.6 FERRAMENTAS PARA DEPLOY

A correta implantação do *FinTrackr* é vital para garantir uma experiência de usuário suave e robusta. Para este fim, optamos por um conjunto de ferramentas líderes de mercado e estabelecemos uma estrutura de execução que prioriza a escalabilidade, consistência e facilidade de manutenção.

- **GitHub:** Nossa escolha para controle de versões, o GitHub, permite uma colaboração eficaz e mantém um registro detalhado de todas as alterações no código, tornando o rastreamento e a reversão, quando necessário, um processo simplificado.
- **Jenkins:** Para a integração contínua e entrega contínua (CI/CD), optamos pelo Jenkins devido à sua flexibilidade, ampla comunidade de suporte e capacidade de integração com uma variedade de outras ferramentas, incluindo o Ansible e o Kubernetes.

#### 4.7 ESTRUTURA DE EXECUÇÃO

A natureza modular e escalável do *FinTrackr* se alinha perfeitamente com as capacidades fornecidas pelo Docker, uma plataforma de contêineres que permite empacotar, distribuir e executar aplicações de forma consistente e isolada. A utilização do Docker não apenas facilita o processo de desenvolvimento, mas também otimiza a operação e a distribuição da aplicação.

### 4.7.1 Contêinerização e Isolamento

O Docker proporciona um ambiente controlado e consistente para executar o *FinTrackr*, isolando suas dependências e serviços associados em contêineres. Essa contêinerização assegura que o aplicativo se comportará da mesma maneira, independentemente de onde seja implantado, mitigando assim os problemas comuns de "funciona na minha máquina". Além disso, a isolamento fornecido pelos contêineres do Docker protege o *FinTrackr* de possíveis interferências de outras aplicações ou serviços no mesmo ambiente.

### 4.7.2 Desenvolvimento e Testes

Durante as fases de desenvolvimento e testes, os desenvolvedores podem tirar proveito da facilidade de inicialização e descarte de contêineres Docker. Isso permite que eles configurem, testem e reconfigurem o ambiente do *FinTrackr* de maneira rápida e sem esforço, acelerando o ciclo de desenvolvimento e garantindo que os testes sejam executados em um ambiente controlado e previsível.

### 4.7.3 Integração e Entrega Contínua

A integração do Docker com ferramentas de Integração Contínua/Entrega Contínua (CI/CD) adotado, o Jenkins, facilita a automação do pipeline de desenvolvimento do *FinTrackr*. Os contêineres Docker podem ser construídos, testados e implantados automaticamente através de pipelines CI/CD, garantindo que cada alteração no código seja verificada e entregue de maneira eficaz e eficiente.

### 4.7.4 Execução

Para executar o *FinTrackr* utilizando Docker, seguiremos uma estrutura padronizada:

1. **Instalação do Docker:** O Docker será instalado no ambiente de execução desejado, seja um servidor local, um ambiente de nuvem ou uma máquina de desenvolvimento.
2. **Criação de Imagens:** Imagens Docker serão criadas para cada componente do *FinTrackr*, incluindo o back-end, front-end e o microserviço de processamento de faturas. Cada imagem será construída usando um Dockerfile que define as dependências, variáveis de ambiente e comandos necessários para executar o componente.
3. **Construção de Imagens:** Usaremos o comando *docker build* para construir as imagens a partir dos Dockerfiles.
4. **Execução de Contêineres:** Os contêineres serão iniciados usando o comando *docker run*, configurando as redes, volumes e outras opções necessárias para conectar os contêineres entre si e com os recursos externos necessários.

## 4.8 GITFLOW

O *GitFlow* é uma estratégia de fluxo de trabalho do Git que define uma estrutura robusta para projetos, facilitando o desenvolvimento paralelo, a colaboração em equipe e o lançamento de versões estáveis. A estratégia é composta por várias *branches* que têm propósitos específicos. A seguir, detalhamos essas *branches* e suas interações:

- **main:** Contém o código considerado como produção. Cada *commit* nesta branch é uma nova versão de produção.
- **develop:** Atua como uma branch de integração para novos recursos e é o principal local de desenvolvimento contínuo.
- **feature branches:** Criadas a partir da branch *develop*. São utilizadas para desenvolver novas funcionalidades sem afetar o código principal. Quando uma funcionalidade é concluída, um *Pull Request* (PR) é criado para mesclar a *feature branch* de volta à branch *develop*.
- **hotfix branches:** Surgem a partir da branch *main*. Destinadas para correções rápidas diretamente no código de produção. Quando uma correção é finalizada, PRs são abertos para integrar a *hotfix branch* tanto na branch *main* quanto na *develop*.

Os *Pull Requests* (PRs) desempenham um papel fundamental no GitFlow. Eles oferecem um ponto de revisão e teste, garantindo que o código seja de alta qualidade antes da integração. Esse processo de revisão é vital para a colaboração em equipe, permitindo que os membros da equipe revisem, comentem e testem as alterações propostas antes de serem incorporadas ao código base.

O GitFlow, com sua estrutura clara e regras definidas, ajuda as equipes a gerenciar e versionar seu código de forma eficaz, garantindo lançamentos estáveis e facilitando o desenvolvimento contínuo e a colaboração.

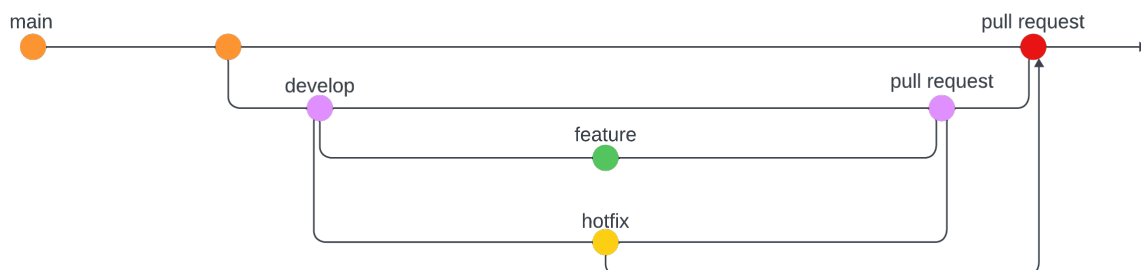


Figura 5 – *GitFlow*

## 4.9 TESTES

### 4.9.1 Ferramentas de Testes

Para a linguagem Python será utilizado o Pytest, conhecido por sua sintaxe simples, identificação automática de testes, uso de fixtures, suporte a parametrização, extensibilidade por meio de plugins, e facilidade de integração com outras ferramentas de desenvolvimento. É amplamente adotado em projetos de código aberto e empresas, tornando o processo de teste eficiente e agradável para os desenvolvedores.

Para o React, será utilizado o Jest, ele é um framework de teste desenvolvido pelo Facebook que é amplamente usado na comunidade React. Ele é projetado para testar JavaScript, incluindo código React. O Jest é conhecido por sua simplicidade e integração perfeita com o React. Ele suporta testes de componentes, testes de unidades e testes de integração.

### 4.9.2 Metrica de Cobertura de Testes

A meta do nosso projeto é atingir uma cobertura de testes de 60%. A cobertura de testes é fundamental para garantir a qualidade e a confiabilidade do nosso software. Com uma cobertura adequada, podemos identificar e corrigir erros de maneira eficaz, melhorar a manutenção do código e aumentar a confiança na estabilidade do sistema.

Para alcançar essa meta, planejamos implementar uma estratégia abrangente de testes que serão executados regularmente em um ambiente de integração contínua para garantir que nossa cobertura de testes seja mantida ao longo do tempo.

Atingir 60% de cobertura de testes é um processo contínuo, e planejamos monitorar nosso progresso ao longo do ciclo de vida do projeto. Estabelecemos marcos específicos para avaliar nosso avanço e faremos ajustes conforme necessário para atender a essa meta.

Acreditamos que essa abordagem nos permitirá entregar um software de alta qualidade, com menos bugs e maior confiabilidade, beneficiando nossos usuários finais e nossa equipe de desenvolvimento.

## 4.10 ARTIFACTS MANAGEMENT

Tecnologia	Aspectos			
	Benefícios	Desvantagens	Uso Comum	Indicado Para
Nexus	- Gerenciamento de dependências.	- Requer configuração.	- Armazenamento de artefatos.	- Projetos grandes.
DockerHub	- Hospedagem de imagens Docker.	- Limitações em repositórios públicos.	- Distribuição de contêineres.	- Projetos com Docker.
Sonar	- Análise contínua da qualidade.	- Configuração inicial.	- Melhoria da qualidade do código.	- Projetos com foco em qualidade.

Tabela 6 – Comparação entre Nexus, DockerHub, Sonar e Infraestrutura

Em nosso projeto, usaremos o SonarQube com o Docker, pois como vimos em sala é muito prático levá-lo junto ao Docker. Ainda é muito vantajoso porque permite analisar e melhorar a qualidade do código fonte, identificar vulnerabilidades de segurança, detectar bugs e defeitos, manter padrões de codificação consistentes, integrar análises de código ao pipeline

de desenvolvimento, fornecer feedback imediato aos desenvolvedores, rastrear o histórico das análises, personalizar a configuração, gerar relatórios detalhados e, em última análise, aprimorar a qualidade do software. Isso contribui para a segurança, confiabilidade e desempenho de aplicativos em containers Docker.

#### 4.11 DOCUMENTAÇÃO

A documentação é um aspecto crucial de qualquer projeto, e no *Fintrackr* não é diferente. Para além das especificações e detalhes técnicos da API, que utilizaremos o Swagger, o Notion desempenhará um papel fundamental como uma plataforma de documentação complementar. No Notion, criaremos uma página dedicada para manter informações pertinentes do projeto, incluindo eventuais documentações que auxiliem no desenvolvimento. Esta abordagem centralizada no Notion garantirá que toda a equipe tenha acesso fácil e rápido a informações críticas, melhorando a eficiência e a colaboração.

O Swagger é uma poderosa ferramenta para a construção de APIs RESTful, proporcionando uma suite completa para desenho, construção, documentação e uso de APIs. Ele fornece um conjunto de recursos que auxiliam os desenvolvedores a criar APIs consistentes, testáveis e padronizadas. Um dos componentes mais valiosos do Swagger é o Swagger UI, que fornece uma interface visual para interagir com a API, permitindo que os desenvolvedores e os usuários finais visualizem e testem as funcionalidades da API de forma intuitiva. No contexto do *FinTrackr*, o Swagger será inestimável para documentar e testar as várias endpoints da API. Isso não só facilitará a integração entre o front-end e o back-end, mas também fornecerá uma referência clara para os desenvolvedores e partes interessadas sobre como a API funciona, quais dados ela aceita e quais respostas podem ser esperadas. Ao adotar o Swagger no *FinTrackr*, garantimos uma abordagem padronizada e eficiente para a construção e documentação da API.

## APÊNDICE A – UC01 - AUTENTICAÇÃO NO SISTEMA

### Identificador

*UC01*

### Título

Autenticação no Sistema

### Atores

- Usuário
- Sistema de Autenticação

### Pré-condições

- O usuário possui acesso à interface de login.

### Fluxo Principal

1. O usuário acessa a página de login.
2. O usuário insere seu email e senha.
3. O sistema valida as credenciais.
4. O usuário é autenticado e redirecionado para o dashboard.

### Fluxos Alternativos

#### • Registro de Novo Usuário

1. O usuário seleciona a opção para registrar-se.
2. O sistema direciona o usuário para o formulário de registro.
3. O usuário preenche os detalhes necessários e submete o formulário.
4. O sistema valida as informações e cria uma nova conta.
5. O usuário é redirecionado para a página de login.

#### • Recuperação de Senha

1. O usuário seleciona a opção "Esqueceu a senha?".
2. O sistema solicita o email do usuário.
3. O usuário insere seu email e submete a solicitação.
4. O sistema envia um link de recuperação de senha para o email do usuário.

## **Fluxos de Exceção**

- **Credenciais Inválidas**

1. O sistema detecta que o email ou senha inseridos estão incorretos.
2. O sistema informa ao usuário sobre as credenciais inválidas.

## **Pós-condições**

- O usuário obtém acesso às funcionalidades autenticadas do sistema.

## **Requisitos**

- **RF01:** Implementação de um sistema de autenticação robusto e seguro, permitindo registro, login, recuperação de senha e gerenciamento de perfil.

## **Regras de Negócio**

- **RN16:** Os usuários devem fornecer um email e senha válidos para se autenticar no sistema.
- **RN17:** Senhas devem ter um mínimo de 8 caracteres e conter pelo menos uma letra maiúscula, uma letra minúscula, um número e um caractere especial.

## **Notas e Comentários**

- A interface de autenticação deve ser simples e intuitiva, minimizando erros de usuário.



## APÊNDICE B – UC02 - GERENCIAR PERFIL

### Identificador

*UC02*

### Título

Gerenciar Perfil

### Atores

- Usuário
- Sistema de Gerenciamento de Perfil

### Pré-condições

- O usuário está autenticado no sistema.

### Fluxo Principal

1. O usuário acessa a seção de perfil.
2. O sistema apresenta as informações atuais do perfil do usuário.
3. O usuário seleciona a opção para editar o perfil.
4. O usuário atualiza as informações desejadas e confirma as mudanças.
5. O sistema valida e salva as alterações.
6. O usuário é informado sobre o sucesso da atualização.

### Fluxos Alternativos

- **Alterar Senha:**
  1. O usuário seleciona a opção para alterar a senha.
  2. O sistema solicita a senha atual e a nova senha.
  3. O usuário insere as informações e confirma.
  4. O sistema valida a senha atual, aplica as regras de complexidade à nova senha e, se válido, atualiza a senha do usuário.

## Fluxos de Exceção

- **Dados Inválidos:**

1. Durante a atualização do perfil, o sistema detecta dados inválidos ou incompletos.
2. O sistema informa ao usuário sobre os campos inválidos ou o que precisa ser corrigido.

## Pós-condições

- As informações do perfil do usuário são atualizadas no sistema.

## Requisitos

- **RF01:** Implementação de um sistema de autenticação robusto e seguro, permitindo registro, login, recuperação de senha e gerenciamento de perfil.
- **RF02:** Facilitar o gerenciamento de informações do perfil do usuário, incluindo visualização, adição, atualização e exclusão de dados como nome, endereço e telefone.

## Regras de Negócio

- **RN17:** Senhas devem ter um mínimo de 8 caracteres e conter pelo menos uma letra maiúscula, uma letra minúscula, um número e um caractere especial.
- **RN20:** Para alterar a senha, o usuário deve fornecer a senha atual para confirmação.

## Notas e Comentários

- As informações do perfil devem ser tratadas com privacidade, e quaisquer alterações devem ser realizadas de forma segura.

## APÊNDICE C – UC03 - REGISTRAR E CATEGORIZAR TRANSAÇÕES

### Identificador

*UC03*

### Título

Registrar e Categorizar Transações

### Atores

- Usuário
- Sistema de Gestão Financeira

### Pré-condições

- O usuário está autenticado no sistema.

### Fluxo Principal

1. O usuário acessa a seção de transações.
2. O usuário seleciona a opção para adicionar uma nova transação.
3. O sistema apresenta o formulário de registro de transações.
4. O usuário escolhe o tipo de transação: RegularTransaction ou CreditCardTransaction.
5. O usuário insere os detalhes da transação, incluindo valor, tipo (receita ou despesa), data, descrição e, se for o caso, a fatura associada (para CreditCardTransaction).
6. O usuário seleciona ou cria categorias para a transação.
7. Se necessário, o usuário define o mês de referência para a transação.
8. O sistema valida e registra a transação.
9. O usuário é informado do sucesso do registro e a transação é exibida na lista de transações.

## Fluxos Alternativos

- **Dividir Transação em Múltiplas Categorias:**

1. Durante o registro da transação, o usuário opta por dividir a despesa em múltiplas categorias.
2. O usuário especifica o valor associado a cada categoria.
3. O sistema verifica se o total dos valores distribuídos corresponde ao valor total da transação.
4. A transação é registrada com múltiplas categorias associadas.

## Fluxos de Exceção

- **Valor da Transação Não Corresponde à Distribuição entre Categorias:**

1. O sistema detecta que o valor total distribuído entre as categorias não corresponde ao valor total da transação.
2. O usuário é informado do erro e solicitado a ajustar os valores.

## Pós-condições

- A transação é registrada e categorizada, e reflete no saldo e relatórios do usuário.

## Requisitos

- **RF03:** Permitir o registro de transações financeiras, categorizando-as como receitas ou despesas e associando detalhes pertinentes.

## Regras de Negócio

- **RN01:** Transações devem ser associadas a uma categoria.
- **RN05:** O total de valores associados em uma transação dividida entre categorias deve igualar o valor total da transação.
- **RN09:** Transações não podem ter datas futuras.
- **RN13:** Categorias não podem ser duplicadas em uma transação dividida.
- **RN23:** Todas as transações devem ter um mês de referência para controle orçamentário.

## Notas e Comentários

- A interface de registro de transações deve ser simples e permitir a rápida inserção de múltiplas entradas.

## APÊNDICE D – UC04 - GERENCIAR CATEGORIAS

### Identificador

*UC04*

### Título

Gerenciar Categorias

### Atores

- Usuário
- Sistema de Gerenciamento de Categorias

### Pré-condições

- O usuário está autenticado no sistema.

### Fluxo Principal

1. O usuário acessa a seção de gerenciamento de categorias.
2. O sistema apresenta a lista de categorias existentes.
3. O usuário seleciona uma ação: criar nova categoria, editar categoria existente ou excluir categoria.
4. O sistema executa a ação desejada e fornece feedback ao usuário.

### Fluxos Alternativos

#### • Criação de Nova Categoria:

1. O usuário seleciona a opção para criar uma nova categoria.
2. O sistema apresenta um formulário para inserção dos detalhes da categoria (nome, cor, ícone).
3. O usuário preenche os detalhes e submete o formulário.
4. O sistema valida as informações e cria a nova categoria.

#### • Edição de Categoria Existente:

1. O usuário seleciona uma categoria da lista.
2. O sistema apresenta o formulário preenchido com os detalhes da categoria selecionada.

3. O usuário altera as informações desejadas e submete o formulário.
4. O sistema valida as alterações e atualiza a categoria.

### **Fluxos de Exceção**

- **Exclusão de Categoria Associada a Transações:**

1. O usuário tenta excluir uma categoria que possui transações associadas.
2. O sistema detecta a associação e informa ao usuário que a categoria não pode ser excluída.

### **Pós-condições**

- As categorias são atualizadas conforme as ações realizadas pelo usuário.

### **Requisitos**

- **RF04:** Habilitar a criação, edição e exclusão de categorias de transações, associando detalhes como nome, cor e ícone.

### **Regras de Negócio**

- **RN01:** Transações devem ser associadas a uma categoria.
- **RN03:** Categorias com transações associadas não podem ser excluídas.
- **RN04:** Ao dividir despesas em múltiplas categorias, o valor associado a cada categoria deve ser especificado.
- **RN13:** Categorias não podem ser duplicadas em uma transação dividida.

### **Notas e Comentários**

- A interface de gerenciamento de categorias deve ser clara e intuitiva, permitindo a criação, edição e exclusão com facilidade.

## APÊNDICE E – UC05 - GERENCIAR ORÇAMENTOS

### Identificador

*UC05*

### Título

Gerenciar Orçamentos

### Atores

- Usuário
- Sistema de Gestão Financeira

### Pré-condições

- O usuário está autenticado e possui acesso à seção de orçamentos.

### Fluxo Principal

1. O usuário acessa a seção de orçamentos.
2. O sistema exibe os orçamentos existentes categorizados por mês e categoria.
3. O usuário seleciona um mês específico para visualizar ou editar.
4. O sistema exibe detalhes do orçamento para o mês selecionado.
5. O usuário pode adicionar, editar ou remover valores de orçamento para categorias específicas.
6. O sistema valida e salva as alterações feitas pelo usuário.

### Fluxos Alternativos

- **Criar Novo Orçamento:**

1. O usuário seleciona a opção para criar um novo orçamento.
2. O sistema apresenta um formulário para inserir valores de orçamento por categoria.
3. O usuário preenche os valores e submete.
4. O sistema valida os dados e cria o novo orçamento.

## **Fluxos de Exceção**

- **Valores Negativos ou Inválidos:**

1. O sistema detecta valores negativos ou inválidos durante a edição ou criação de um orçamento.
2. O sistema informa ao usuário sobre o erro e solicita correção.

## **Pós-condições**

- As alterações feitas pelo usuário nos orçamentos são salvas e refletidas no sistema.

## **Requisitos**

- **RF05:** Proporcionar a definição e monitoramento de orçamentos mensais por categoria, mostrando gastos realizados e disponíveis.

## **Regras de Negócio**

- **RN05:** Orçamentos não podem ter valores negativos.
- **RN12:** Usuários devem ser notificados ao atingir ou exceder orçamentos.

## **Notas e Comentários**

- A seção de orçamentos deve ser de fácil acesso e visualização, permitindo uma gestão rápida e eficiente dos valores.



## **APÊNDICE F – UC06 - VISUALIZAR DASHBOARD FINANCEIRO**

### **Identificador**

*UC06*

### **Título**

Visualizar Dashboard Financeiro

### **Atores**

- Usuário
- Sistema de Finanças

### **Pré-condições**

- O usuário está autenticado no sistema.

### **Fluxo Principal**

1. O usuário acessa a seção de dashboard.
2. O sistema exibe o saldo total em contas.
3. O sistema lista todas as contas com seus respectivos saldos e transações recentes.
4. O sistema apresenta um resumo dos orçamentos definidos para o mês.
5. O sistema mostra o total gasto por categoria.
6. O sistema exibe um balanço geral de receitas versus despesas para o período selecionado.

### **Fluxos Alternativos**

- **Exclusão de Conta do Saldo Total:**
  1. O usuário seleciona uma ou mais contas para serem excluídas do saldo total.
  2. O sistema atualiza o saldo total exibido, excluindo as contas selecionadas.

## **Fluxos de Exceção**

- **Dados Indisponíveis:**

1. O sistema detecta um erro ao recuperar os dados financeiros do usuário.
2. O sistema informa ao usuário sobre o erro e sugere que ele tente novamente mais tarde.

## **Pós-condições**

- O usuário tem uma visão clara e atualizada de sua situação financeira.

## **Requisitos**

- **RF06:** Prover um dashboard que apresente um resumo financeiro, incluindo saldo em contas, resumo de orçamentos, gastos por categoria e balanço geral.

## **Regras de Negócio**

- **RN05:** O total de valores associados em uma transação dividida entre categorias deve igualar o valor total da transação.
- **RN06:** Orçamentos não podem ter valores negativos.
- **RN07:** Despesas em cartões de crédito devem ser associadas à fatura corrente.
- **RN08:** Transações em contas impactam o saldo da mesma.
- **RN09:** Transações não podem ter datas futuras.

## **Notas e Comentários**

- A interface do dashboard deve ser projetada para facilitar a compreensão rápida dos principais indicadores financeiros do usuário.

## APÊNDICE G – UC07 - IMPORTAR FATURAS

### Identificador

*UC07*

### Título

Importar Faturas

### Atores

- Usuário
- Microserviço de Importação

### Pré-condições

- O usuário possui uma fatura no formato .csv para importar.
- O usuário está autenticado e tem permissão para importar faturas.

### Fluxo Principal

1. O usuário navega até a seção de importação de faturas.
2. O usuário seleciona e carrega o arquivo .csv da fatura desejada.
3. O sistema valida o formato e conteúdo do arquivo.
4. O microserviço de importação processa o arquivo e extrai as transações.
5. As transações são registradas no sistema associadas à conta ou cartão do usuário.
6. O usuário é informado de que a importação foi bem-sucedida.

### Fluxos Alternativos

- **Edição Pós-Importação:**

1. Após a importação bem-sucedida, o usuário opta por revisar as transações importadas.
2. O sistema apresenta uma lista das transações importadas.
3. O usuário edita ou exclui transações conforme necessário.
4. O sistema salva as alterações feitas pelo usuário.

## **Fluxos de Exceção**

- **Arquivo Inválido:**

1. O sistema detecta que o arquivo .csv carregado é inválido ou corrompido.
2. O sistema informa ao usuário sobre o problema com o arquivo e solicita um novo arquivo.

## **Pós-condições**

- As transações da fatura são registradas no sistema e podem ser visualizadas e gerenciadas pelo usuário.

## **Requisitos**

- **RF07:** Integrar com um microserviço para permitir a importação de faturas bancárias em formato .csv, iniciando pelo Bradesco.

## **Regras de Negócio**

- **RN07:** Despesas em cartões de crédito devem ser associadas à fatura corrente.
- **RN10:** Importações de faturas via .csv devem permitir edição pós-importação.

## **Notas e Comentários**

- A funcionalidade de importação deve suportar faturas de diferentes bancos, iniciando pelo Bradesco.

## APÊNDICE H – UC08 - GERENCIAR CARTÕES E FATURAS

### Identificador

*UC08*

### Título

Gerenciar Cartões e Faturas

### Atores

- Usuário
- Sistema de Gerenciamento Financeiro

### Pré-condições

- O usuário está autenticado e possui acesso à seção de gerenciamento de cartões.

### Fluxo Principal

1. O usuário acessa a seção de gerenciamento de cartões.
2. O usuário visualiza seus cartões cadastrados.
3. O usuário seleciona um cartão específico para visualizar detalhes e faturas associadas.
4. O usuário pode adicionar, editar ou excluir informações do cartão.
5. O usuário pode visualizar, adicionar, editar ou excluir faturas associadas ao cartão.

### Fluxos Alternativos

- **Adicionar Novo Cartão:**

1. O usuário seleciona a opção para adicionar um novo cartão.
2. O sistema apresenta um formulário para inserção de detalhes do cartão.
3. O usuário preenche os detalhes e submete o formulário.
4. O sistema valida as informações e adiciona o novo cartão à lista do usuário.

- **Excluir Cartão:**

1. O usuário seleciona um cartão e escolhe a opção para excluir.
2. O sistema solicita confirmação.

3. O usuário confirma a exclusão.
4. O sistema remove o cartão e todas as faturas associadas.

### **Fluxos de Exceção**

- **Erro ao Adicionar Cartão:**

1. O sistema detecta um erro durante a adição do cartão (por exemplo, informações inválidas).
2. O sistema informa ao usuário sobre o erro.

- **Erro ao Excluir Cartão:**

1. O sistema detecta um erro durante a exclusão do cartão (por exemplo, faturas pendentes).
2. O sistema informa ao usuário sobre o erro.

### **Pós-condições**

- As informações do cartão e faturas associadas são atualizadas conforme as ações do usuário.

### **Requisitos**

- **RF08:** Facilitar o gerenciamento de cartões de crédito, incluindo registro de faturas e lançamentos associados.

### **Regras de Negócio**

- **RN02:** Cartões e contas com faturas ou transações associadas não podem ser excluídos.
- **RN07:** Despesas em cartões de crédito devem ser associadas à fatura corrente.
- **RN15:** Informações de cartões ou contas duplicadas não são permitidas.

### **Notas e Comentários**

- A interface de gerenciamento de cartões deve oferecer uma visão clara de todas as informações e operações disponíveis.

## APÊNDICE I – UC09 - ACOMPANHAR EVOLUÇÃO DE SALDO

### Identificador

*UC09*

### Título

Acompanhar Evolução de Saldo

### Atores

- Usuário
- Sistema Financeiro

### Pré-condições

- O usuário está autenticado e possui acesso ao dashboard.

### Fluxo Principal

1. O usuário seleciona a opção para visualizar o histórico de saldo.
2. O sistema apresenta uma representação gráfica (por exemplo, um gráfico de linhas) da evolução do saldo ao longo do tempo.
3. O usuário pode selecionar diferentes intervalos de tempo (por exemplo, mês atual, últimos 6 meses, ano atual, etc.).
4. O sistema atualiza a visualização com base no intervalo de tempo selecionado.
5. O usuário pode interagir com o gráfico para obter detalhes específicos de datas e valores.

### Fluxos Alternativos

- **Exportar Dados:**
  1. O usuário seleciona a opção para exportar os dados do histórico de saldo.
  2. O sistema oferece formatos de exportação (por exemplo, CSV, PDF).
  3. O usuário seleciona o formato desejado e confirma.
  4. O sistema gera e disponibiliza o arquivo para download.

## **Fluxos de Exceção**

- **Sem Dados Suficientes:**

1. O sistema detecta que não há dados suficientes para exibir uma evolução significativa do saldo.
2. O sistema informa ao usuário sobre a falta de dados e sugere o registro de mais transações.

## **Pós-condições**

- O usuário tem uma visão clara da evolução do seu saldo financeiro ao longo do tempo.

## **Requisitos**

- **RF09:** Oferecer uma visualização detalhada do histórico de saldo, mostrando a evolução financeira do usuário.

## **Regras de Negócio**

- **RN08:** Transações em contas impactam o saldo da mesma.
- **RN09:** Transações não podem ter datas futuras.

## **Notas e Comentários**

- A interface de visualização do histórico de saldo deve ser clara e proporcionar fácil interpretação dos dados.



## APÊNDICE J – UC10 - GERENCIAR BANCOS

### Identificador

*UC10*

### Título

Gerenciar Bancos

### Atores

- Usuário
- Sistema de Gerenciamento de Bancos

### Pré-condições

- O usuário está autenticado no sistema.
- O usuário possui acesso à seção de gerenciamento de bancos.

### Fluxo Principal

1. O usuário acessa a seção de gerenciamento de bancos.
2. O sistema exibe a lista de bancos cadastrados pelo usuário.
3. O usuário pode escolher adicionar, editar ou excluir informações de um banco.
4. O sistema realiza a ação solicitada e atualiza a lista de bancos.

### Fluxos Alternativos

#### • Adicionar Novo Banco:

1. O usuário seleciona a opção para adicionar um novo banco.
2. O sistema apresenta um formulário para inserção de detalhes do banco.
3. O usuário preenche os detalhes e submete o formulário.
4. O sistema valida as informações e adiciona o novo banco à lista.

#### • Editar Informações de um Banco Existente:

1. O usuário seleciona um banco da lista e escolhe a opção para editar.
2. O sistema apresenta um formulário preenchido com as informações atuais do banco.

3. O usuário realiza as alterações desejadas e submete o formulário.
4. O sistema valida e atualiza as informações do banco.

### **Fluxos de Exceção**

- **Informações do Banco Duplicadas:**

1. O sistema detecta que as informações inseridas para um novo banco ou após a edição já existem para outro banco registrado.
2. O sistema informa ao usuário sobre a duplicidade e solicita correção.

### **Pós-condições**

- A lista de bancos do usuário é atualizada com as informações corretas.

### **Requisitos**

- **RF10:** Permitir o registro e gerenciamento de bancos associados às contas do usuário.

### **Regras de Negócio**

- **RN10:** O email fornecido pelo usuário durante o registro ou atualização do perfil deve ser único no sistema.
- **RN15:** Informações de cartões ou contas duplicadas não são permitidas.

### **Notas e Comentários**

- O gerenciamento eficaz de informações bancárias é crucial para a integridade dos dados e a experiência do usuário.

## **APÊNDICE K – UC11 - DEFINIR MÊS DE REFERÊNCIA PARA TRANSAÇÕES**

### **Identificador**

*UC11*

### **Título**

Definir Mês de Referência para Transações

### **Atores**

- Usuário
- Sistema de Gestão Financeira

### **Pré-condições**

- O usuário está autenticado no sistema.
- O usuário está registrando ou editando uma transação.

### **Fluxo Principal**

1. Durante o registro ou edição de uma transação, o usuário identifica a necessidade de definir um mês de referência.
2. O sistema apresenta a opção para selecionar o mês de referência.
3. O usuário seleciona o mês e ano desejados.
4. O sistema valida a escolha.
5. O mês de referência é associado à transação.

### **Fluxos Alternativos**

*N/A*

### **Fluxos de Exceção**

*N/A*

### **Pós-condições**

- O mês de referência é definido para a transação, permitindo um controle orçamentário mais preciso.

**Requisitos**

- **RF03:** Permitir o registro de transações financeiras, categorizando-as como receitas ou despesas e associando detalhes pertinentes.

**Regras de Negócio**

- **RN23:** Todas as transações devem ter um mês de referência para controle orçamentário.

**Notas e Comentários**

- A definição do mês de referência é especialmente crucial para transações de cartão de crédito, onde o impacto orçamentário pode não coincidir com a data da transação.

## APÊNDICE L – ADR-001 - TENTATIVA DE IMPLEMENTAR TODO O PROJETO DO *FINTRACKR* USANDO MICROSERVIÇOS

### *Title*

Tentativa de Implementar Todo o Projeto do *Fintrackr* Usando Microserviços.

### *Status*

Rejected - Devido ao tempo e à complexidade aumentada, decidiu-se não seguir essa abordagem.

### *Context*

O projeto *Fintrackr* visa oferecer uma ferramenta robusta para o gerenciamento de finanças. Com o aumento da complexidade e a variedade de funcionalidades, considerou-se a possibilidade de utilizar uma abordagem totalmente baseada em microserviços.

### *Decision*

Tentar implementar todo o projeto do *Fintrackr* usando uma arquitetura baseada em microserviços.

### *Consequences*

- **Especialização:** Cada microserviço pode ser otimizado para sua função específica.
- **Flexibilidade:** Microserviços podem ser desenvolvidos, escalados ou modificados independentemente.
- **Resiliência:** Falhas em um microserviço não afetariam outros microserviços.
- **Complexidade Aumentada:** A gestão de múltiplos microserviços pode complicar o desenvolvimento e a manutenção.

## APÊNDICE M – ADR-002 - ADOÇÃO DE ARQUITETURA MONOLÍTICA PARA A LÓGICA DE NEGÓCIOS PRINCIPAL DO *FINTRACKR*

### *Title*

Adoção de Arquitetura Monolítica para a Lógica de Negócios Principal do *Fintrackr*.

### *Status*

Accepted

### *Context*

O projeto *Fintrackr* visa fornecer uma solução robusta para o gerenciamento de finanças. Uma parte significativa do sistema é composta por operações de cadastro e lógica de negócios. Para acomodar essas necessidades de maneira coesa, é essencial considerar uma abordagem arquitetural que possa integrar eficientemente esses componentes.

### *Decision*

Adotar uma arquitetura monolítica para a lógica de negócios principal e cadastros do *Fintrackr*.

### *Consequences*

- **Consistência:** Um monólito oferece uma base de código unificada, o que pode simplificar o desenvolvimento e a manutenção.
- **Performance:** Com todos os componentes principais no mesmo processo, as latências de comunicação são minimizadas.
- **Simplicidade:** Menos preocupações com a comunicação entre serviços.
- **Escalabilidade Vertical:** O monólito pode ser escalado verticalmente, mas pode haver limitações na escalabilidade horizontal.
- **Potencial de Acoplamento:** Pode haver riscos de acoplamento apertado entre os módulos ao longo do tempo.
- **Curva de aprendizado:** Desenvolvedores familiarizados com microserviços podem precisar de tempo para se adaptar à abordagem monolítica.
- **Flexibilidade:** Modificações no sistema podem requerer reimplantações completas em vez de serviços individuais.

- **Manutenção:** À medida que o projeto cresce, a manutenção pode se tornar mais desafiadora devido ao tamanho do código base.

## APÊNDICE N – ADR-003 - CRIAÇÃO DE MICROSERVIÇO ESPECIALIZADO PARA PROCESSAMENTO DE PLANILHAS NO *FINTRACKR*

### *Title*

Criação de Microserviço Especializado para Processamento de Planilhas no *Fintrackr*.

### *Status*

Accepted

### *Context*

O *Fintrackr* tem uma necessidade específica de processar informações de planilhas, transformando-as em um formato desejado para integração com o sistema. Esta operação é distinta das funções de lógica de negócios principais e, portanto, requer uma abordagem arquitetural especializada.

### *Decision*

Criar um microserviço dedicado responsável por receber, processar e retornar informações de planilhas em um formato desejado.

### *Consequences*

- **Especialização:** O microserviço pode ser otimizado para processar planilhas de forma eficiente.
- **Flexibilidade:** O microserviço pode ser desenvolvido, escalado ou modificado independentemente do monólito.
- **Isolamento:** Falhas ou problemas no microserviço não afetarão o funcionamento do monólito.
- **Complexidade Adicional:** Introduz a necessidade de gerenciar a comunicação entre o monólito e o microserviço.
- **Overhead de Rede:** A comunicação entre o monólito e o microserviço pode introduzir latências.
- **Curva de aprendizado:** Os desenvolvedores podem precisar de formação ou familiarização com a gestão de microserviços.
- **Gerenciamento:** Monitoramento, logging e rastreamento adicionais serão necessários para gerenciar o microserviço.



- **Integração:** Garantir uma integração robusta e eficiente com o monólito será crucial.

## APÊNDICE O – ADR-004 - ADOÇÃO DE ARQUITETURA MONOLÍTICA NO FRONT-END DO *FINTRACKR*

### *Title*

Adoção de Arquitetura Monolítica no Front-end do *Fintrackr*.

### *Status*

Accepted

### *Context*

O projeto *Fintrackr* busca fornecer uma solução eficaz para o gerenciamento de finanças. Dada a pouca experiência da equipe com o desenvolvimento de front-end e a necessidade de garantir uma rápida aprendizagem e eficiência no desenvolvimento, é crucial escolher uma abordagem arquitetural que seja familiar e minimamente complexa.

### *Decision*

Adotar uma arquitetura monolítica para o front-end do *Fintrackr*.

### *Consequences*

- **Simplicidade:** Um front-end monolítico é mais simples de entender, desenvolver e manter, especialmente para equipes menos experientes.
- **Integração Estreita:** Todos os componentes do front-end estarão estreitamente integrados, permitindo uma melhor coesão.
- **Menor Curva de Aprendizado:** A equipe pode se concentrar em desenvolver funcionalidades em vez de gerenciar a complexidade de múltiplos serviços ou componentes independentes.
- **Desafios de Escalabilidade:** À medida que o projeto cresce, pode haver desafios associados à escalabilidade e manutenção do front-end monolítico.

## APÊNDICE P – ADR-005 - FRONT-END REACT - COMPONENT-BASED ARCHITECTURE

### *Title*

Front-end React - Component-Based Architecture.

### *Status*

Accepted

### *Context*

O *Fintrackr* visa fornecer uma interface de usuário intuitiva e eficiente para gerenciamento de finanças. A escolha da arquitetura do front-end é crucial para garantir uma experiência de usuário consistente e de alta qualidade.

### *Decision*

Adotar a arquitetura baseada em componentes usando o framework React para o front-end do *Fintrackr*.

### *Consequences*

- **Modularidade:** A arquitetura baseada em componentes permite modularidade, facilitando o desenvolvimento, teste e manutenção de partes individuais da interface do usuário.
- **Reusabilidade:** Componentes podem ser reutilizados em diferentes partes da aplicação ou até mesmo em outros projetos.
- **Eficiência:** Com o Virtual DOM do React, as atualizações da interface do usuário são otimizadas para serem rápidas e eficientes.
- **Comunidade:** React tem uma grande comunidade e ecossistema, oferecendo muitas bibliotecas e ferramentas auxiliares.
- **Curva de Aprendizado:** Pode haver uma curva de aprendizado para desenvolvedores não familiarizados com React ou arquitetura baseada em componentes.
- **Dependências:** Dependência de bibliotecas e ferramentas específicas do ecossistema React.

## APÊNDICE Q – ADR-006 - BACK-END FLASK - CAMADAS COM CLEAN/ONION

### *Title*

Back-end Flask - Camadas com Clean/Onion.

### *Status*

Accepted

### *Context*

O *Fintrackr* tem a responsabilidade de gerenciar a lógica de negócios relacionada ao gerenciamento de finanças. A escolha da arquitetura do back-end é essencial para garantir que a aplicação seja flexível, testável e manutenível.

### *Decision*

Adotar uma combinação de arquitetura em camadas com Clean/Onion para o back-end do *Fintrackr* construído com Flask.

### *Consequences*

- **Separação de Responsabilidades:** Esta abordagem garante que as responsabilidades sejam claramente separadas em diferentes camadas.
- **Flexibilidade:** Facilita mudanças e evolução do código ao manter a lógica de negócios separada da infraestrutura.
- **Testabilidade:** Torna mais fácil escrever testes unitários para a lógica de negócios.
- **Manutenibilidade:** O código torna-se mais fácil de manter e entender.
- **Curva de Aprendizado:** Pode haver uma curva de aprendizado para desenvolvedores não familiarizados com a arquitetura Clean/Onion.
- **Complexidade:** A introdução de várias camadas e a separação rigorosa podem aumentar a complexidade inicial.

## APÊNDICE R – ADR-007 - MICROSERVIÇO COM PANDAS - HEXAGONAL

### *Title*

Microserviço com Pandas - Hexagonal.

### *Status*

Accepted

### *Context*

O *Fintrackr* tem uma necessidade específica de processar dados de planilhas para integração com o sistema. Esta funcionalidade é distinta das funções principais da aplicação e pode necessitar de escalabilidade e adaptabilidade independentes.

### *Decision*

Adotar a arquitetura hexagonal para o microserviço especializado em processamento de dados com Pandas.

### *Consequences*

- **Flexibilidade:** A arquitetura hexagonal permite que o microserviço seja facilmente adaptado para diferentes interfaces ou integrações no futuro.
- **Testabilidade:** Facilita a escrita de testes, pois a lógica de negócios é desacoplada dos adaptadores.
- **Isolamento:** Falhas ou problemas no microserviço não afetarão outras partes da aplicação.
- **Especialização:** O microserviço pode ser otimizado para processar planilhas de forma eficiente.
- **Curva de Aprendizado:** Pode haver uma curva de aprendizado para desenvolvedores não familiarizados com a arquitetura hexagonal.
- **Gerenciamento Adicional:** Requer monitoramento, logging e rastreamento adicionais para gerenciar o microserviço.

## APÊNDICE S – ADR-008 - ADOÇÃO DA ARQUITETURA REST PARA COMUNICAÇÃO ENTRE FRONT-END E BACK-END FLASK

### *Title*

Adoção da Arquitetura REST para Comunicação entre Front-end e Back-end Flask.

### *Status*

Accepted

### *Context*

O *Fintrackr* possui componentes de front-end e um back-end construído com Flask. Para garantir uma comunicação eficiente e padronizada entre esses dois componentes, é necessário escolher uma arquitetura de comunicação apropriada.

### *Decision*

Adotar a arquitetura REST (Representational State Transfer) como a principal forma de comunicação entre o front-end e o back-end Flask do *Fintrackr*.

### *Consequences*

- **Padronização:** A arquitetura REST fornece um conjunto padronizado de convenções para a criação de APIs.
- **Escalabilidade:** REST é stateless, o que facilita a escalabilidade horizontal do sistema.
- **Flexibilidade:** Permite que o front-end e o back-end Flask se comuniquem de maneira eficiente.
- **Interoperabilidade:** Facilita a integração com sistemas externos ou de terceiros.
- **Simplicidade e Eficiência:** REST utiliza os métodos HTTP padrão e é baseado em recursos, tornando-o intuitivo e eficiente.
- **Desacoplamento:** Permite que o front-end e o back-end Flask evoluam separadamente, desde que a API permaneça consistente.
- **Conformidade com Padrões da Web:** REST é amplamente adotado na web, e muitas ferramentas e bibliotecas suportam essa arquitetura.
- **Considerações de Segurança:** Como qualquer API exposta, medidas de segurança adequadas, como autenticação e autorização, devem ser implementadas ao comunicar entre o front-end e o back-end Flask.

## APÊNDICE T – ADR-009 - ADOÇÃO DO RABBITMQ PARA COMUNICAÇÃO ENTRE O MICROSERVIÇO DA PLANILHA DE FATURA E O BACK-END FLASK

### *Title*

Adoção do RabbitMQ para Comunicação entre o Microserviço da Planilha de Fatura e o Back-end Flask.

### *Status*

Accepted

### *Context*

Para integrar o microserviço especializado da planilha de fatura com o back-end Flask do *Fintrackr*, é crucial ter uma solução de messageiria confiável que facilite a comunicação assíncrona, garanta a entrega de mensagens e suporte a escalabilidade.

### *Decision*

Adotar o RabbitMQ como a principal solução de messageiria para garantir a comunicação eficiente entre o microserviço da planilha de fatura e o back-end Flask.

### *Consequences*

- **Comunicação Assíncrona:** RabbitMQ permite que o microserviço e o back-end Flask se comuniquem de forma assíncrona, o que pode melhorar a eficiência e a resposta do sistema.
- **Garantia de Entrega:** As mensagens podem ser armazenadas e reenviadas em caso de falhas, garantindo a entrega.
- **Escalabilidade:** RabbitMQ suporta balanceamento de carga entre múltiplas instâncias, o que pode ajudar na escalabilidade da solução.
- **Flexibilidade:** Permite a integração com outros sistemas e serviços no futuro, graças ao seu modelo de messageiria baseado em tópicos e filas.
- **Complexidade Adicional:** A introdução de uma solução de messageiria pode aumentar a complexidade do sistema e requerer monitoramento e gerenciamento adicionais.
- **Dependência Externa:** A adoção do RabbitMQ introduz uma dependência externa que precisa ser gerenciada e mantida.

## APÊNDICE U – ADR-010 - ADOÇÃO DO SQLALCHEMY PARA COMUNICAÇÃO COM O BANCO DE DADOS NO FLASK

### *Title*

Adoção do SQLAlchemy como ORM para Comunicação com o Banco de Dados no Flask.

### *Status*

Aceito

### *Context*

O *Fintrackr*, construído usando Flask, precisa interagir de maneira eficiente e segura com o banco de dados. A escolha de uma estratégia ou ferramenta adequada para esta comunicação é crucial para a robustez e manutenibilidade do sistema.

### *Decision*

Adotar o SQLAlchemy, um ORM popular e amplamente utilizado na comunidade Flask, como a principal ferramenta para estabelecer a comunicação entre a aplicação Flask e o banco de dados.

### *Consequences*

- **Abstração:** O SQLAlchemy permite que a equipe de desenvolvimento trabalhe com o banco de dados usando a linguagem Python, abstraindo muitos detalhes do SQL.
- **Segurança:** O SQLAlchemy tem proteções embutidas contra injeções SQL, reduzindo o risco desses tipos de ataques.
- **Produtividade:** Facilita operações comuns do banco de dados, como inserções, atualizações, leituras e exclusões, através de uma API intuitiva.
- **Portabilidade:** O SQLAlchemy suporta vários sistemas de gerenciamento de banco de dados, tornando mais fácil mudar para um banco de dados diferente no futuro, se necessário.
- **Complexidade Adicional:** Apesar de ser poderoso, pode haver uma curva de aprendizado inicial para desenvolvedores não familiarizados com o SQLAlchemy ou seus padrões.