UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

LUCAS MARTINS DE BARROS

INTEGRAÇÃO DE DADOS PARA CONTROLE FINANCEIRO

BENTO GONÇALVES

2024

LUCAS MARTINS DE BARROS

INTEGRAÇÃO DE DADOS PARA CONTROLE FINANCEIRO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Profa. Dra. Helena Graziottin Ribeiro

BENTO GONÇALVES

2024

LUCAS MARTINS DE BARROS

INTEGRAÇÃO DE DADOS PARA CONTROLE FINANCEIRO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 29/11/2024

BANCA EXAMINADORA

Profa. Dra. Helena Graziottin Ribeiro
Universidade de Caxias do Sul - UCS

Prof. Dr. Daniel Luis Notari
Universidade de Caxias do Sul - UCS

Prof. Me. Leonardo Pellizzoni
Universidade de Caxias do Sul - UCS

RESUMO

A crescente diversidade de instituições financeiras, aliada à quantidade de plataformas e aplicativos específicos dessas instituições, tem tornado o controle financeiro pessoal um desafio. Essa dispersão de informações financeiras dificulta a consolidação e análise dos dados financeiros pessoais, essenciais para a tomada de decisões sobre compras, investimentos e planejamento financeiro. Este trabalho propõe e implementa uma solução para centralizar dados financeiros de uma pessoa de forma automatizada, resolvendo o problema da fragmentação dos dados. A solução desenvolvida inclui um *pipeline* de engenharia de dados para a extração e integração de dados financeiros de diferentes instituições, além da criação de um protótipo de aplicação móvel que exibe essas informações de forma consolidada. O trabalho aborda conceitos de engenharia e integração de dados, o *Open Finance* como padrão de dados financeiros, e técnicas de modelagem de sistemas que foram fundamentais para o desenvolvimento e a validação da solução.

Palavras-chave: Engenharia de dados. ETL. Aplicação móvel. Dados bancários. *Open Finance*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

elem.	elemento
RECA	Revista Eletrônica de Computação Aplicada
ETL	*Extraction, transform and load*
ELT	*Extraction, load and transform*
IIOP	*Internet InterORB Protocol*
SQL	*Structure Query Language*
API	*Application Programming Interface*
URL	*Uniform Resource Locator*
DW	*Data Warehouse*
RH	Recursos Humanos
BACEN	Banco Central do Brasil
HTTP	Protocolo de Transferência de Hipertexto
JSON	JavaScript Object Notation
DTU	unidade de transação do banco de dados
CLI	interface de linha de comando
DTU	Unidade de Transação do Banco de Dados

SUMÁRIO

1 INTRODUÇÃO

A quantidade de instituições financeiras combinada com a diversidade de plataformas e aplicativos próprios dessas instituições, tornam o controle financeiro pessoal cada vez mais desafiador. Por exemplo o banco Bradesco, que de acordo com seu próprio informativo possui oito aplicativos, sendo os principais: "Bradesco Cartões" para algumas bandeiras de cartão de crédito, "*Invest*+ Bradesco" para informações de investimentos e "Bradesco" para contas correntes (BRADESCO, s.d).

Clientes com múltiplas contas bancárias, muitas vezes em instituições financeiras diferentes, têm suas informações financeiras muito dispersas, dificultando a consolidação e análise dos próprios dados. A obtenção e consolidação desses dados financeiros são essenciais para a tomada de decisões de compras, investimentos e planejamento financeiro.

À vista disso, surge a necessidade de uma solução que unifique e padronize os dados financeiros de diferentes instituições de forma automatizada. Por outro lado, cada instituição têm sua própria estrutura de dados, além de políticas de intercâmbio específicas. Em tempos de globalização e interoperabilidade de sistemas, houve o crescimento do conceito de *Open Finance* no Brasil, através do qual foram definidos padrões de representação para esses dados:

> *Open Finance* ou Sistema Financeiro Aberto é uma iniciativa do Banco Central do Brasil que tem como principais objetivos trazer inovação ao sistema financeiro, promover a concorrência, e melhorar a oferta de produtos e serviços financeiros para o consumidor. (BACEN, 2024b)

Ao ter acesso a um panorama completo das suas finanças, os clientes podem avaliar sua situação financeira e realizar escolhas mais conscientes e estratégicas. A engenharia de dados, como apresentado por Reis e Housley (2022), desempenha um papel fundamental nesse contexto, permitindo o desenvolvimento de estratégias e programas para realizar a integração de dados em um ambiente de forma automatizada. Oferecendo uma interface única para os dados das diferentes instituições financeiras, os usuários podem contar com uma visão precisa da sua situação econômica, facilitando a análise e tomada de decisões.

Sendo assim, definiu-se como problema de pesquisa para este estudo: "é viável centralizar os dados financeiros de um indivíduo, de forma automatizada, distribuídos em mais de uma instituição?"

Há diversos trabalhos e aplicativos disponíveis para gestão financeira pessoal que apresentam diferentes abordagens para organizar as finanças de um indivíduo. Um trabalho relevante é uma publicação na Revista Eletrônica de Computação Aplicada (RECA), desenvolvido por Guilherme Assis, João Rodrigues e Geraldo Neto, intitulada "BILL: Desenvolvimento e validação de uma aplicação móvel para gerenciamento financeiro pessoal". A publicação aborda

uma aplicação para controle financeiro através de transações registradas pelos usuários, estabelecimento de metas financeiras e acompanhamento de balanços. Porém, o foco permaneceu em dados manuais, sem integração automatizada com bancos ou instituições financeiras no geral (ASSIS JOãO V. O. RODRIGUES, 2020).

Entre os aplicativos disponíveis no mercado, o *Despezzas*[1] é uma solução de gestão financeira que permite a sincronização de transações diretamente com os bancos, utilizando uma tecnologia semelhante à que será empregada neste trabalho. Porém, o aplicativo adota um modelo de assinatura, onde a versão gratuita oferece apenas funcionalidades manuais.

Os aplicativos de algumas instituições financeiras também oferecem funcionalidades de gestão financeira. Por exemplo, a aplicação do banco *Nubank* oferece a possibilidade de categorizar transações realizadas com o cartão de crédito. No entanto, permite somente a visualização de transações realizadas através do próprio banco, sem integrar dados de cartões de outras instituições financeiras, mesmo que estejam conectadas através do *Open Finance*, o que impossibilita a consolidação das informações.

1.1 OBJETIVOS

O objetivo deste trabalho é construir um *pipeline* de engenharia de dados para integração de dados financeiros reais de uma pessoa física em mais de uma instituição financeira e desenvolver um protótipo de aplicação móvel para exibição dos dados extraídos.

1.2 ESTRUTURA DO TEXTO

Primeiramente, este trabalho irá conceituar os temas necessários para o desenvolvimento de uma integração de dados financeiros. O Capítulo 2 irá explicar a área de engenharia de dados, contextualizar seu uso para integrar dados e detalhar o processo de ETL. O Capítulo 3 detalhará os padrões de dados financeiros e evidenciar a forma que será realizada a obtenção dos dados neste trabalho.

Na sequência, é apresentada a proposta de desenvolvimento da integração de dados e protótipo de aplicação móvel. O Capítulo 4 apresentará as tecnologias que serão usadas no desenvolvimento, a arquitetura da solução e os detalhes de cada um dos requisitos.

Por último, são mostrados os resultados do desenvolvimento da integração e do protótipo de exibição de dados. O Capítulo 5 mostra como foi realizada a integração dos dados, as preparações de ambiente, desenvolvimento da API e desenvolvimento do *pipeline* de integração. Em sequência, o Capítulo 6 mostra como o protótipo de exibição de dados foi desenvolvido e o resultado das telas. Ao final, no Capítulo 7, são apresentados os testes realizados para validar a proposta de solução.

[1] <https://despezzas.framer.ai>

2 ENGENHARIA DE DADOS

Engenharia de dados é a área que faz o tratamento inicial dos dados de uma empresa. Basicamente, o papel de um engenheiro de dados é extrair dados, armazená-los e prepará-los para serem usados por analistas e cientistas de dados, especialistas de negócio entre outros. Ou seja, engenharia de dados tem como produto final a padronização de informações a serem usadas em análises e tomadas de decisão (REIS; HOUSLEY, 2022).

> O dado é um tijolo, a informação é uma parede construída por vários tijolos e o conhecimento é um ou mais cômodos construídos a partir da organização e correto relacionamento de várias paredes (CÔRTES, 2017, p. 26).

O engenheiro de dados é responsável por construir o conhecimento dito pelo autor Côrtes (2017). Ele obtém os dados de uma fonte ou mais fontes e constrói a informação aplicando regras de transformação. Esses dados transformados, em geral, se tornam um novo conjunto de dados, que é carregado para um banco de dados para ser analisado e gerar conhecimento.

2.1 INTEGRAÇÃO DE DADOS

O trabalho do engenheiro de dados pode ser resumido em um termo: transformação de dados, o que inclui a integração de dados. Para entender a importância da integração de dados vamos imaginar o cenário de uma empresa chamada "*FullServe*", como exemplificado por Doan, Halevy e Ives (2012). A *FullServe* não é uma empresa real, porém, segundo os autores, a situação em que ela se encontra é plausível na situação econômica do momento dos autores (2012).

A empresa ficcional usada de exemplo por Doan, Halevy e Ives (2012), fornece serviços de Internet e vende periféricos como *modems* e roteadores. Apesar de ser uma empresa fundada nos Estados Unidos, recentemente adquiriu uma empresa europeia de cartões de crédito, a *EuroCard*, com objetivo de ampliar seus serviços.

Quando empresas passam por reestruturações, como no caso de aquisições, nem sempre os bancos de dados e sistemas das empresas são alinhados e padronizados (DOAN; HALEVY; IVES, 2012). Com essa aquisição, a complexidade dos bancos de dados da *FullServe* aumentou significativamente. Uma versão simplificada dos bancos de dados das empresas, mostrando especificamente a estrutura de dados funcionários, é representada por:

- FullServe - Banco de dados de funcionários:
  - **Contratos:** <u>funcID</u>, dataContratacao, recrutador

  - **Funcionários:** SSN, #funcId, nome, sobrenome, dataContratacao, dataRecisao, cargaHoraria

- FullServe - Banco de dados de funcionários temporários:
  - **FuncionáriosTemporários:** SSN, nomeCompleto, dataContratacao, dataRecisao, cargaHoraria

- EuroCard - Banco de dados de funcionários:
  - Contratos: ID, dataContratacao, dataRecisao, recrutador
  - **Funcionários:** ID, nome, sobrenome, SSN

É importante notar que as empresas organizam as mesmas informações de formas diferentes. Por exemplo a *FullServe* tem tabelas em bancos de dados diferentes para funcionários de tempo integral e temporários - isso pode acontecer por utilizarem uma agência externa para gerenciar esses funcionário - enquanto a *EuroCard* armazena em uma tabela única.

Agora, imagine que o departamento de Recursos Humanos (RH) precisa de uma lista de todos os funcionários, tanto da *FullServe* quanto da *EuroCard*. Essa informação existe em três fontes diferentes, duas da *FullServe* (um para funcionários de tempo integral e outro para temporários) e uma da *EuroCard*.

Esse exemplo mostra a importância da integração de dados, já que para responder perguntas simples, como a lista dos funcionários da empresa, é necessário realizar uma integração, mesmo que de forma manual. Esse é o desafio chave para o avanço de muitos campos da ciência, onde grupos de cientistas coletam dados de forma independente mas precisam colaborar entre a comunidade científica (DOAN; HALEVY; IVES, 2012).

Integrações de sistemas podem ser realizada a partir de diferentes estratégias, como mapeamentos entre os sistemas e interfaces comuns. No caso de ter como foco a utilização de uma interface comum, é necessário o uso de padrões de representação de dados para troca de informações, uma linguagem padrão para acesso aos conjuntos de dados. De acordo com He e Xu (2014), integrações podem ser realizadas em diferentes camadas de uma aplicação:

- **Camada de comunicação:** para fazer duas aplicações distintas se comunicarem e trocarem informações e dados é comum utilizar protocolos como Protocolo de Transferência de Hipertexto (HTTP) e *Internet InterORB Protocol* (IIOP) para troca de dados.

- **Camada de dados:** se trata da movimentação ou manipulação de dados entre diferentes fontes. O maior esforço para a integração de dados é o mapeamento dos elementos entre o esquema de dados origem com o destino. Esse mapeamento pode ser facilitado quando o destino e a origem utilizam um padrão previamente estabelecido. Há padrões definidos

por instituições e organismos reconhecidos em diferentes áreas, como a HL7[1] na área da saúde, "*SWIFT*"[2] para trocas entre bancos estrangeiros , e o *Open Finance*[3] para dados financeiros no Brasil.

- **Camada de lógica de negócio:** essa integração é focada em desenvolvimento de tecnologias de *middleware*, focadas em auxiliar os desenvolvedor e facilitar integrações, conectando aplicações e fornecendo uma forma de comunicação.

- **Camada de apresentação:** a integração via camada de apresentação foca na interface de usuário. Um exemplo deste tipo de integração é o "*Portlet*", que se trata de um componente visual independente que é utilizado para ajudar a produzir aplicações com elementos externos de forma flexível.

O processo de engenharia de dados atua na camada de dados das integrações, e é definido pelo design e implementação de um *pipeline* de dados, um fluxo automatizado ou parcialmente atualizado que executa os processamentos de extração, transformação e carga - ETL. Apesar de não ser a única forma de processamento de dados, o ETL é uma das metodologias mais populares e eficientes atualmente para gerar conjuntos de dados apropriados para alimentar métodos de análise específicos (NWOKEJI; MATOVU, 2021).

Um processo semelhante é o *Extraction, load and transform* (ELT), que passou a ter destaque a partir da geração de grandes volumes de dados (*Big Data*). Nesse processo, os dados extraídos são carregados de suas fontes em um grande repositório, o "*data lake*"[4]. No momento que vão ser utilizados em algum método de análise, sofrem as transformações necessárias (NAMBIAR; MUNDRA, 2022).

2.2 ETL

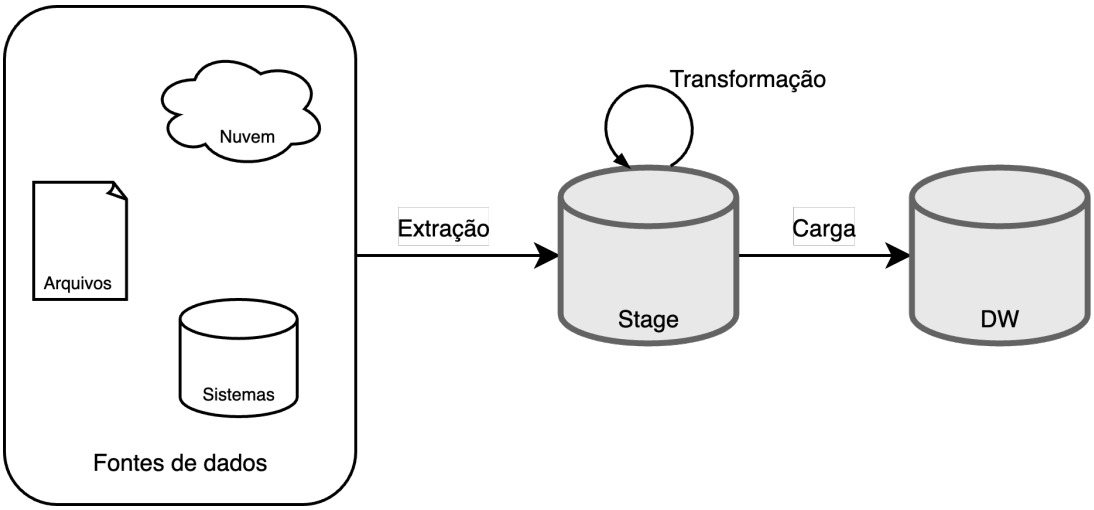
Os engenheiros de dados, de forma geral, se organizam em dois grupos, os com foco em *Structure Query Language* (SQL), e os engenheiros de *Big Data*. Enquanto os engenheiros de dados SQL fazem todo o processamento com SQL e talvez alguma ferramenta de ETL, os engenheiros de *Big Data* processam os dados utilizando linguagens de programação, como Python e Scala, além de uma linguagem de consulta, que pode ser SQL para bancos de dados relacionais ou linguagens específicas de sistemas não relacionais (REIS; HOUSLEY, 2022).

Como simbolizado pelo nome, o processo de ETL é composto por três etapas, conforme a Figura 1: extração dos dados de uma ou mais fontes, transformação (ou pré-processamento), e

[1] <www.hl7.org.br>
[2] <www.theswiftcodes.com>
[3] <www.openfinancebrasil.org.br>
[4] Um repositório centralizado que armazena dados estruturados e não estruturados, normalmente em grande escala.

se necessário, carga dos dados processados em um novo repositório, como um *Data Warehouse* (DW).

Figura 1 – Estrutura do ETL



Fonte: O Autor (2024).

2.2.1 Extração

A etapa de extração se dá pela coleta de dados brutos de fontes diversas, como por exemplo de bancos de dados operacionais, de arquivos diversos e de dados de sistemas acessados através de API específicas. A maior complexidade desta etapa é a organização dessas fontes em um repositório integrado, para reunir todos os dados necessários em um local centralizado, chamado de *stage*, que armazena os dados de forma bruta, exatamente como chegam das fontes (EL-SAPPAGH; HENDAWI; EL-BASTAWISSY, 2011).

O processo de extração é feito em duas fases distintas, uma extração inicial e as extrações incrementais. O processo de extração inicial é realizado apenas na primeira vez, para popular o DW com os dados das fontes até o momento. Já as extrações incrementais consistem em identificar quando houve uma alteração dos dados nas fontes, realizar a extração e atualização apenas dos dados modificados (EL-SAPPAGH; HENDAWI; EL-BASTAWISSY, 2011).

2.2.2 Transformação

A segunda etapa do processo de ETL é a transformação e limpeza dos dados. Essa etapa, segundo os autores Vassiliadis e Simitsis (2009), pode ser dividida em três níveis: ao nível de esquema, nível de registro e ao nível de valor.

Os problemas ao nível de esquema que necessitam de transformações são os conflitos nominais e estruturais. Conflitos nominais acontecem quando as fontes utilizam nomes diferentes para representar o mesmo objeto ou utilizam o mesmo nome para representar objetos

diferentes (objetos podem ser tabelas ou colunas). A identificação desses conflitos pode ser feita manualmente, por um ser humano, ou através de funções que identificam similaridade em graus diversos. Esse conflito normalmente é resolvido renomeando esses objetos (VASSILIADIS; SIMITSIS, 2009).

Enquanto os conflitos estruturais são diferentes representações do mesmo objeto, normalmente em fontes diferentes, como por exemplo, tipos de dados diferentes para o mesmo objeto. Esse problema é tratado convertendo os tipos de dados entre as fontes e o DW (VASSILIADIS; SIMITSIS, 2009).

A necessidade de aplicar transformações ao nível de registro acontece principalmente quando existem registros duplicados e/ou contraditórios (exigindo uma deduplicação de dados). As definições de como tratar esses registros normalmente é definida pelo engenheiro de dados em conjunto com a área de negócio (VASSILIADIS; SIMITSIS, 2009).

Outra necessidade de transformação ao nível de registros é quando existem diferenças de granularidade entre as fontes. Por exemplo, quando uma fonte registra a quantidade de vendas por dia e outra a quantidade de vendas por mês, necessitando uma padronização da granularidade das informações (VASSILIADIS; SIMITSIS, 2009).

A última categoria de transformação, segundo Vassiliadis e Simitsis (2009) é ao nível de valor, que pode ser, por exemplo, a padronização de valores com diferentes representações (e.g., para sexo: "Masculino", "M"). Outros exemplos são a substituição de constantes, definição de um valor padrão para registros nulos e a identificação e tratamento de *outliers* (valores fora de um intervalo definido para uma coluna).

Esses tipos de transformações frequentemente utilizam funções do SQL como o *UPPER*, para converter *strings* em letras maiúsculas, *SUBSTRING*, para retornar parte de uma *string*, além de diversas funções matemáticas, como para arredondamento, média e valores absolutos. Outra transformação ao nível de valores muito frequente é a criação de colunas condicionais, utilizando funções como o *CASE*, que avalia condições e retorna uma das várias expressões de resultados possíveis (MICROSOFT, 2023).

Além das transformações citadas, existem diversos outros tipos de operações de transformação. Como citado por Hameed e Naumann (2020), existem atividades de preparação de dados como: ordenação, comparação de valores, verificação de caracteres permitidos, verificação de tipos de dados compatíveis, pivotamento, união de dados, extração de partes de um valor, alteração de letras maiúsculas/minúsculas, preenchimento de células vazias, remoção de espaços em branco, entre outros.

2.2.3 Carga

A última etapa do processo de ETL é a carga, onde os dados extraídos e transformados são carregados para um ambiente para ser consumido pelos usuários finais (no DW). Assim

como a extração, a etapa de carga pode ser feita em duas etapas, uma carga inicial e depois cargas incrementais (EL-SAPPAGH; HENDAWI; EL-BASTAWISSY, 2011).

A carga inicial é realizado apenas na primeira vez, onde todos os dados são inseridos no DW, enquanto a carga incremental tem o desafio de distinguir os registros que serão inseridos dos registros que vão agir como atualização de registros previamente carregados (VASSILIADIS; SIMITSIS, 2009). A carga incremental normalmente é realizada por uma função dos bancos SQL chamada *MERGE*, que tem o objetivo de identificar quando um registro precisa ser inserido ou agir como atualização (MICROSOFT, 2023).

3 INTEGRAÇÃO DE DADOS FINANCEIROS

Incentivadas pela crise financeira de 2008, seguida por uma crescente desconfiança da população aos bancos tradicionais, as *Fintechs*, novas instituições do setor financeiro que aplicam tecnologia para serviços financeiros, cresceram e se multiplicaram. Essas empresas evoluíram a maneira como as pessoas utilizam serviços financeiros. A ascensão das *Fintechs* promoveu uma cultura de inovação e maior concorrência no setor (BUCKLEY; ARNER; BARBERIS, 2016).

Esse cenário impulsionou o conceito de um sistema financeiro aberto, com dados compartilhados entre instituições. O *Open Banking* surgiu no Brasil em 2021 com esse objetivo, o compartilhamento padronizado de dados financeiros entre instituições, porém, focando inicialmente apenas em dados das próprias instituições financeiras, e não de seus clientes. O *Open Finance* é a evolução do *Open Banking*, que incluiu novas informações sobre produtos e serviços, possibilitando o compartilhamento de dados de clientes do sistema financeiro (BACEN, 2022).

3.1 PADRÕES DE DADOS FINANCEIROS

Para que o *Open Finance* seja efetivamente implementado, é essencial que os dados das instituições financeiras estejam padronizados de maneira consistente e confiável. Para isso, o Banco Central do Brasil (BACEN) definiu normas e padrões a serem seguidas pelas instituições financeiras para receber e enviar dados.

Para participar do *Open Finance*, enviando ou recebendo dados, é necessário ser uma instituição financeira autorizada pelo BACEN, além de seguir as regras e normas impostas pelo Banco Central. Essas normas impõem que o compartilhamento de dados aconteça de forma segura, garantindo confidencialidade e proteção da informação (BACEN, 2023).

A documentação técnica, localizada na "Área do Desenvolvedor" no site do *Open Finance*, especifica todos padrões a serem seguidos pelas instituições financeiras participantes. Alguns exemplos de padrões impostos são os campos obrigatórios e opcionais de cada *endpoint*[1] das API's , seus tipos e como formatá-los, convenções de nomenclatura, envio de cabeçalhos e códigos de resposta HTTP (BACEN, 2024a).

A Tabela 1 mostra os campos, obrigatórios e opcionais, que devem ser fornecidos no *endpoint* que lista as transações de uma conta de cliente, de acordo com a documentação técnica do BACEN (2024a). Esse retorno é realizado por um objeto JavaScript Object Notation (JSON), por isso, pode conter listas, objetos e campos opcionais.

[1] *Endpoint* é uma *Uniform Resource Locator* (URL) que serve como ponto de comunicação, recebendo e respondendo consultas.

Nome	Tipo	Mandatoriedade	Descrição
/data	Lista	Obrigatório	Lista de lançamentos
/data/transactionId	Texto	Obrigatório	Identificador único prestado pela instituição que matém a conta.
/data/completedAuthorisedPaymentType	Texto	Obrigatório	Indicador de transação efetivada, futura ou em processamento.
/data/creditDebitType	Texto	Obrigatório	Identificador de débito ou crédito.
/data/transactionName	Texto	Obrigatório	Nome descritivo da transação, como apresentado no extrato.
/data/type	Texto	Obrigatório	Identificador de tipo de transação, como TED, DOC, PIX...
/data/transactionAmount	Objeto	Obrigatório	Contém informações sobre o valor da transação.
/data/transactionAmount/amount	Texto	Obrigatório	Valor da transação.
/data/transactionAmount/currency	Texto	Obrigatório	Moeda do valor.
/data/transactionDateTime	Texto	Obrigatório	Data e hora da transação.
/data/partieCnpjCpf	Texto	Condicional	CPF ou CNPJ do pagador ou recebedor envolvido na transação.
/data/partiePersonType	Texto	Opcional	Identificação de Pessoa Físicula ou Jurídica envolvida na transação.
/data/partieCompeCode	Texto	Opcional	Código identificador atribuído pelo BACEN.
/data/partieBranchCode	Texto	Opcional	Código da Agência detentora da conta da pessoa envolvida na transação.
/data/partieNumber	Texto	Opcional	Número da conta da pessoa envolvida na transação.
/data/partieCheckDigit	Texto	Opcional	Dígito da conta da pessoa envolvida na transação.

Tabela 1 – Campos do *endpoint* de transações

Alguns campos, como "*/data/completedAuthorisedPaymentType*", "*/data/creditDebitType*" e "*/data/type*" são campos que devem conter valores pré-determinados pelo BACEN. Por exemplo, o campo "*/data/creditDebitType*" pode conter apenas os valores "CREDITO" ou "DEBITO", que identificam se o valor da transação é positivo ou negativo.

Um padrão que se diferencia é o formato do valor, que também é definido previamente pelo BACEN. Um exemplo é o campo "*/data/transactionAmount/currency*" que deve mostrar a moeda seguindo o modelo da ISO-4217 (e.g. "BRA" para Real Brasileiro e "USD" para Dólar Americano). Outro campo que segue um padrão de formato é o "*/data/transactionDateTime*", que define que as datas devem seguir o formato "yyyy-MM-ddTHH:mm:ss.SSSZ" (e.g. "2016-01-29T12:29:03.374Z").

Os padrões de *Open Finance* não cobrem todas as possibilidades de objetos ou API's que os participantes desejam mostrar. Portanto, o BACEN prevê o princípio de extensibilidade, que permite que as instituições forneçam informações que não estão previstas nos padrões do *Open Finance*. Essas extensões também servem como base para futuras alterações na própria definição dos padrões. Os participantes tem a possibilidade de estender os aspectos de:

- **API:** uma API completa que não está coberta nos padrões definidos.
- ***Endpoints*:** novos *endpoints* em uma API que já está definida pelo *Open Finance*.
- **Campos de entrada e retorno opcionais:** novos campos para um *endpoint* já definido.

Essas extensões não devem impedir que um consumidor que foi projetado para consumir apenas os campos padrões funcione como esperado. Além disso, as extensões ainda devem seguir alguns padrões gerais definidos pelo BACEN.

3.2 OBTENÇÃO DE DADOS FINANCEIROS

A obtenção dos dados por meio do *Open Finance* é limitada a instituições financeiras, portanto pessoas físicas ou instituições de outras áreas não têm acesso a esses dados (BACEN, 2023). Essas informações, apesar de não estarem disponíveis para esse público, podem ser interessantes principalmente para o controle financeiro pessoal, sendo ele realizado por meio de uma instituição que provê uma uma aplicação de controle financeiro ou até mesmo um cliente lendo seus próprios dados diretamente via *Open Finance*.

Para resolver esse problema, surgiram *fintechs* especializadas em fornecer esses dados do *Open Finance* para o público que não possui acesso. Essas *fintechs* são instituições financeiras autorizadas pelo BACEN que têm o objetivo de prover os dados em um local centralizado, tanto para pessoas físicas que querem acessar seus próprios dados, quanto para empresas não diretamente do setor financeiro que buscam construir aplicações usando dados do *Open Finance*. Além de disponibilizar os dados para esse público que não teria acesso, também eliminam a necessidade de extrair os dados de cada instituição individualmente, já que elas agrupam os dados das diferentes instituições, fazendo o trabalho mais intenso de integração.

Uma das plataformas, que tem como um dos objetivos fornecer os dados para esse público sem acesso é a Pluggy[2], uma *fintech* paulistana especializada em dados financeiros do *Open Finance*. Segundo a própria Pluggy (2024a), eles possuem diversos produtos, alguns exemplos são as funções de enriquecimento de dados, que pode ser feita, por exemplo, através da categorização das transações, a iniciação de pagamentos, que utiliza as contas do cliente cadastradas na Pluggy para realizar pagamentos usando o *Open Finance* e a portabilidade de crédito, que possibilita o cliente analisar e solicitar a portabilidade de crédito para outras instituições financeiras.

A Pluggy também fornece um produto chamado "*MeuPluggy*", que é focado nos próprios clientes do setor financeiro que querem agrupar suas informações de diferentes contas em um ambiente único. O *MeuPluggy* pode ser usado tanto para consulta desses dados, através da API da instituição, quanto para visualização desses dados, através do próprio produto. Segundo a documentação técnica da Pluggy (2024b), alguns tipos de dados são:

- **Identidade:** nome, endereço, telefone e documento.
- **Cartões de crédito:** limite, valor da fatura, valor de pagamento mínimo e dívidas.
- **Transações:** data, descrição, valor, moeda;
- **Investimentos:** nome, balanço, tipo, taxas, status, quantidade de ativos.
- **Empréstimos:** data, valor, taxa, periodicidade de pagamento.

[2] <www.pluggy.ai>

É importante ressaltar que esses dados, apesar de que não vão ser acessados diretamente pelas API's do *Open Finance* das instituições, ainda são originários das mesmas, portanto, os campos são muito semelhantes aos definidos pelo BACEN. As diferenças em relação aos campos da Pluggy com os do *Open Finance* se dá pelas transformações feitas pela Pluggy, que podem ser necessárias para por exemplo integrar campos "extras" das instituições, como previsto pelo princípio da extensibilidade pelo BACEN (PLUGGY, 2024b; BACEN, 2024a).

Outra plataforma com objetivo semelhante à Pluggy é a Belvo[3], outra *fintech* brasileira que é especializada em *Open Finance*, com produtos que fornecem dados de forma centralizada até funcionalidades de iniciação de pagamentos, assim como a Pluggy. A Belvo, de acordo sua própria documentação (BELVO, 2024), entrega, de forma geral, dados de uma forma muito similar à Pluggy, já que as duas têm como fontes API's padronizadas pelo *Open Finance*.

Por outro lado, a Belvo mantém o foco em empresas que buscam acesso ao *Open Finance*, e não diretamente a clientes e pessoas físicas que buscam acesso aos seus próprios dados, como no "*MeuPluggy*" da *fintech* anterior. Por esse motivo, para o desenvolvimento da solução apresentada, a *Pluggy* será usada para fornecer os dados do *Open Finance*.

[3] <www.belvo.com>

4 PROPOSTA DE DESENVOLVIMENTO

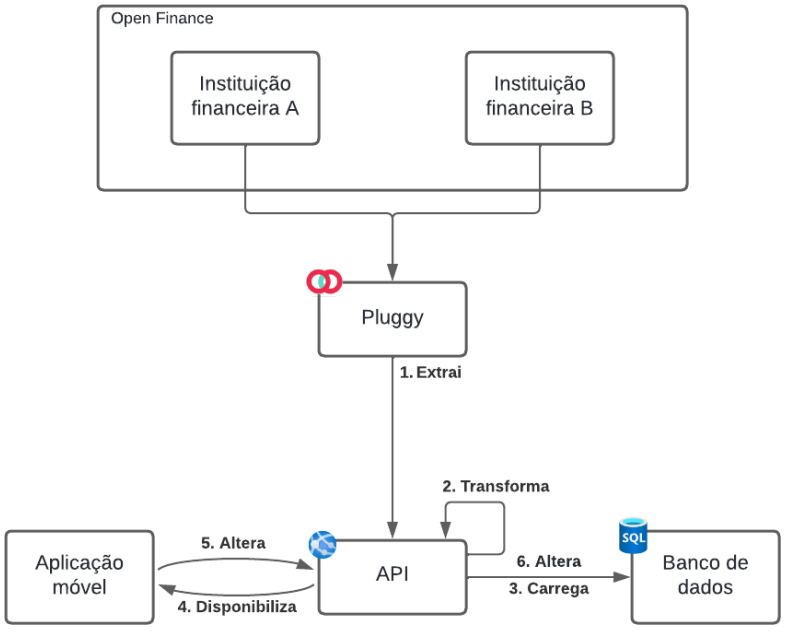
Este capítulo apresenta a proposta de desenvolvimento do *pipeline* de integração de dados do *Open Finance* e de um protótipo de aplicação móvel para visualização desses dados e categorização das transações[1].

4.1 ARQUITETURA

Para atender o objetivo proposto, foi definida a arquitetura da solução representada na Figura 2. As instituições financeiras, dentro do ambiente do *Open Finance*, obedecendo aos seus padrões e regras são acessadas via *Pluggy* (plataforma apresentada na Seção 3.2).

Uma API, que será desenvolvida neste trabalho, irá fazer o processo ETL, extraindo os dados de transações fornecidos pela *Pluggy*, realizando as transformações necessárias e carregando os dados transformados para um banco de dados. A mesma API será responsável por disponibilizar os dados a um protótipo de aplicação móvel, responsável por apresentar os dados integrados. Além de apresentar os dados, o protótipo também irá possibilitar o usuário de realizar modificações nos dados extraídos, como a categorização das transações, representada nas etapas cinco e seis da arquitetura.

Figura 2 – Arquitetura



Fonte: O Autor (2024).

[1] Transações são qualquer movimentação financeira apresentada em uma fatura, positiva ou negativa.

A arquitetura da solução é baseada em um modelo de três camadas. A camada de apresentação é representada pela aplicação móvel, que interage com o usuário e envia requisições à API. A camada de lógica de negócio é implementada pela API, responsável por processar as requisições e aplicar regras de negócio. Por fim, a camada de dados é composta pelo banco de dados, que armazena e gerencia todas as informações necessárias. A organização em três camadas melhora a modularidade do sistema e permite que cada componente seja atualizado ou dimensionado independentemente.

4.2 TECNOLOGIAS USADAS NO DESENVOLVIMENTO

Esta seção lista os recursos tecnológicos e ferramentas que serão utilizadas no desenvolvimento do pipeline de integração de dados e protótipo de aplicação móvel propostos neste trabalho.

4.2.1 Flask

A API da solução será desenvolvida em *Flask*[2], um *micro-framework* para desenvolvimento web do *Python*, sendo voltado especialmente para aplicações simples, porém extensíveis. O *micro-framework* fornece ferramentas de gerenciamento de rotas e requisições HTTP.

O *Flask* tem como principal concorrente o *Django*[3], um *framework* de desenvolvimento web também escrito em *Python*. Apesar de possuírem características similares, o contexto de utilização é diferente. O *Django* é mais adequado para aplicações com maior complexidade, enquanto o *Flask* é voltado para pequenas aplicações que necessitam de mais agilidade, tanto de desenvolvimento quanto de execução (IDRIS; FOOZY; SHAMALA, 2020).

A API *Flask* será hospedada em uma aplicação web na nuvem *Azure*[4], da *Microsoft*. O serviço da *Azure* possui uma opção de plano gratuito, com fácil escalonamento caso necessário. Além disso, a ferramenta disponibiliza um modelo de código *Flask* com autenticação via "*Microsoft Entra ID*"[5] já implementada.

4.2.2 Pandas

As transformações de dados realizadas pelo pipeline de ETL da solução serão realizadas através da biblioteca *Pandas*[6], uma biblioteca *Python* de código aberto voltada para processamento e transformação de dados. Auxilia na limpeza, análise e manipulação de conjuntos de dados estruturados, se tornando uma importante ferramenta para a segunda etapa do processo de ETL, descrito na Seção 2.2.2.

[2] <www.flask.palletsprojects.com>
[3] <www.djangoproject.com>
[4] <www.azure.microsoft.com/pt-br/products/app-service/web>
[5] <www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id>
[6] <www.pandas.pydata.org>

A biblioteca tem como base duas classes, que representam duas estruturas de dados, as séries e os *DataFrames*. Séries são estruturas de uma dimensão, que armazenam algum tipo de dados (como uma coluna), enquanto *DataFrames* são estruturas bidimensionais (tabulares).

4.2.3 Azure SQL Server

Azure SQL Server[7] é um banco de dados relacional em nuvem oferecido pela plataforma *cloud* da *Microsoft*, a *Azure*. Trata-se de uma plataforma de banco de dados altamente escalável e gerenciada que possibilita fácil integração com outros serviços da *Azure*. O banco de dados será responsável por armazenar os dados extraídos pelo *pipeline* de integração.

4.2.4 Swift

Swift[8] é uma linguagem de programação de código aberto criada pela Apple para desenvolvimento de aplicações para *iOS*, *macOS*, *tvOS* e *watchOS*. As aplicações desenvolvidas em *Swift* são denominadas "nativas", ou seja, são criadas especificamente para uma plataforma, neste caso, para os sistemas operacionais da *Apple*.

De acordo com Choudhary, Mudi e Sharma (2020), o desenvolvimento nativo no lugar do híbrido traz vantagens e desvantagens. Em questão de performance e experiência de usuário, o desenvolvimento nativo se sobressai. Entretanto, por ser necessário o desenvolvimento de aplicações distintas caso haja necessidade de uma aplicação em plataformas diferentes, como no *iOS* e *Android* por exemplo, o custo e tempo de desenvolvimento é muito maior, tornando o desenvolvimento híbrido mais interessante.

Para o desenvolvimento do protótipo de aplicação tema deste trabalho não será necessário desenvolver a aplicação em plataformas diferentes das suportadas pelo *Swift*. Portanto, o desenvolvimento híbrido é dispensável para a proposta.

4.3 DIAGRAMA DE CASOS DE USO

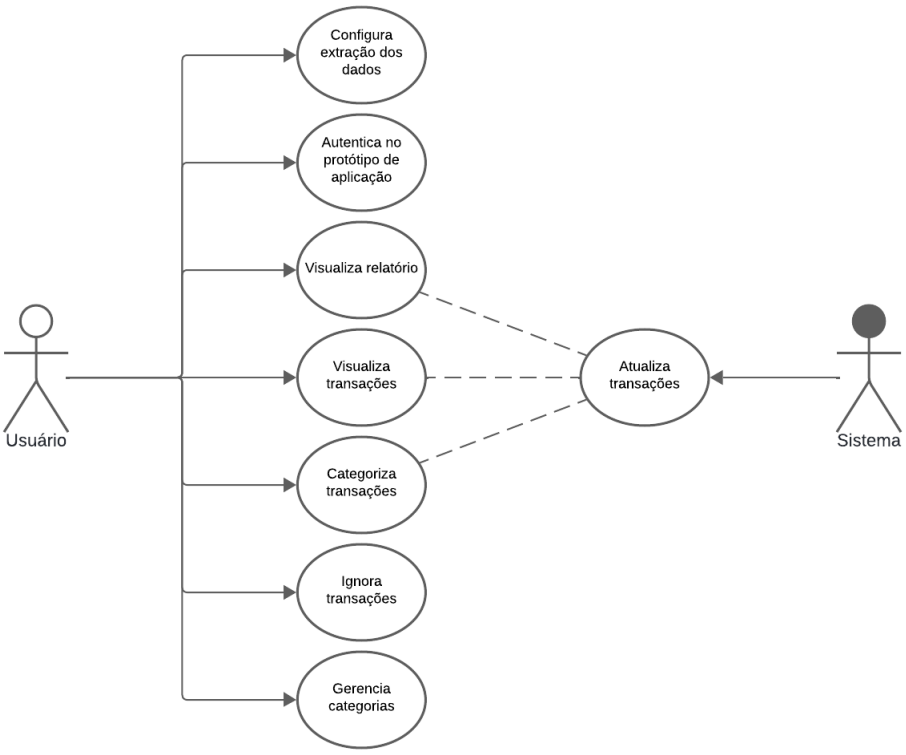
Os diagramas de caso de uso são responsáveis pelo rastreamento de funcionalidades e as interações entre usuário e sistema (PRESSMAN; MAXIM, 2021). Na Figura 3 é possível observar o diagrama de casos de uso do sistema. Existem dois atores, o usuário e o próprio sistema, esses atores interagem com oito requisitos funcionais, que são:

- **Configura extração dos dados:** o usuário deve realizar etapas para preparar a *Pluggy* para extrair dados das suas contas bancárias.
- **Autentica no protótipo de aplicação:** o usuário deve realizar o processo de autenticação via *Entra ID* para acessar o protótipo de aplicação.

[7] <www.azure.microsoft.com/pt-br/products/azure-sql/database>
[8] <www.apple.com/br/swift>

- **Visualização de transações:** o usuário pode visualizar uma lista de suas transações consolidadas em contas de diferentes instituições.
- **Categorização de transações:** o usuário pode categorizar uma transação.
- **Ignorar transação:** o usuário pode marcar uma transação como ignorada, desconsiderando o valor dela nos relatórios.
- **Visualização de relatório:** o usuário pode visualizar gráficos com indicadores de gastos e receitas mensais.
- **Gerencia categorias:** o usuário pode criar e editar categorias para agrupar transações.
- **Atualiza transações:** o sistema irá atualizar a lista de transações das diferentes instituições financeiras de acordo com um gatilho do usuário.

Figura 3 – Diagrama de casos de uso



Fonte: O Autor (2024).

4.4 REQUISITOS FUNCIONAIS

Esta seção irá apresentar detalhadamente cada um dos requisitos do sistema proposto neste trabalho. Requisitos funcionais representam como o sistema deve se comportar, as funcionalidades que deve possuir para atender as necessidades do usuário (PRESSMAN; MAXIM, 2021).

Cada requisito, com exceção dos requisitos de configuração (Seção 4.4.1 e Seção 4.4.2), irá possuir uma descrição e o protótipo das telas necessárias. Toda a arquitetura da solução, assim como a estruturação dos dados é baseada na condição de que o protótipo e a extração dos dados será para para um único usuário.

4.4.1 Configuração da extração de dados

Para realizar a extração de dados, o usuário deverá realizar um cadastro na plataforma "*MeuPluggy*" e conectar as contas bancárias via *Open Finance*. Ao criar a conta no "*MeuPluggy*" também é liberado o acesso à API da *Pluggy*.

Nas configurações da API, deverá ser criado um time, esse time poderá conter um conector e uma aplicação. Conectores são os componentes que farão a conexão com as contas bancárias, que nesse caso será o próprio "*MeuPluggy*". Enquanto as aplicações são registros de aplicativos que poderão conectar nesse time e fazer o uso dos seus conectores. Cada aplicação gera credenciais que serão responsáveis pela autenticação da conexão.

Com as contas bancárias cadastradas, o time criado usando o conector do "*MeuPluggy*" e as credenciais de aplicação geradas, a extração dos dados bancários estará disponível para a solução. A API da solução (Seção 4.2.1) deverá ser configurada passando as credenciais da aplicação criada na *Pluggy*. Com isso, a API da solução irá conseguir extrair os dados do *Open Finance* via "*MeuPluggy*".

4.4.2 Autenticação do protótipo de aplicação

Ao acessar a API da solução, será necessário realizar uma autenticação, portanto, o protótipo de aplicação móvel deverá ter um fluxo de *login* da *Microsoft*, para conseguir acessar a API. Esse *login* será via "Entra ID"[9] um serviço de gerenciamento de acesso da *Microsoft*.

O "Entra ID" integra facilmente com os outros serviços da nuvem *Azure*. Tanto a aplicação web *Flask* quanto a aplicação móvel em *Swift* possuem modelos prontos fornecidos pela *Microsoft* que integram o método de autenticação. Para acessar qualquer uma das telas que serão citadas nas próximas seções será pré-requisito realizar o fluxo de autenticação da API.

A autenticação para acessar a aplicação web na nuvem *Azure* (API da solução) irá validar se o usuário que fez o login via "Entra ID" está na mesma organização da aplicação *web* hospedada. Caso o usuário estiver na organização, poderá realizar consultas na API e será direcionado para a página inicial da aplicação móvel (que será descrita pelo requisito da Seção 4.4.3).

[9] <www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id>

4.4.3 Visualização de relatório

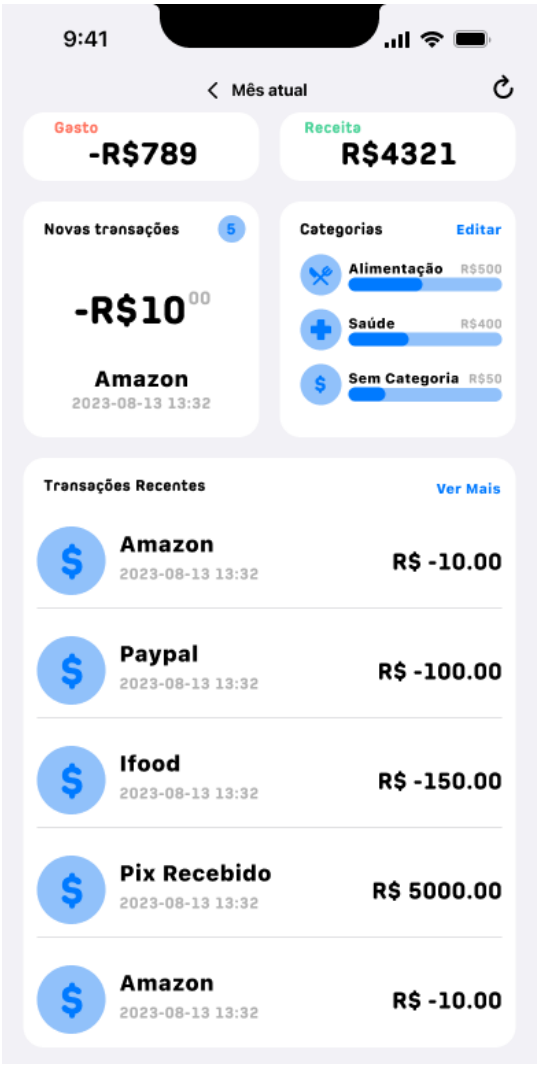
Ao acessar o protótipo, caso o usuário esteja autenticado, irá visualizar um relatório de suas receitas e gastos mensais. Essa tela deverá apresentar a soma de todos os gastos e separadamente a soma de todas receitas, assim como a quantidade de transações que ainda não foram categorizadas, denominadas como “novas”. Além disso, a tela também deverá mostrar as categorias com maiores gastos do mês atual e uma lista com as transações mais recentes, com uma opção de expandir para visualizar todas as transações efetuadas.

4.4.3.1 Protótipo de interface

Na Figura 4 é mostrado um protótipo da interface da página inicial do sistema. O protótipo de interface mostra uma seleção de mês, que por padrão será “Mês atual”, uma opção para atualizar os dados (que aciona o requisito de atualização de transações da Seção 4.4.8), e cinco cartões com dados financeiros do mês selecionado, sendo eles:

1. Soma de todos gastos (não ignorados) do mês.
2. Soma de todas receitas (não ignoradas) do mês.
3. Informações da primeira transação não categorizada e quantidade de novas transações. Esse cartão também é um botão que redirecionará o usuário à tela de categorização de novas transações (Seção 4.4.5).
4. Gráfico de barras horizontais com as três categorias com maior soma de gastos. O “total” de cada barra é a soma total de gastos do mês (do cartão 1) e a quantidade preenchida mostra o quanto aquela categoria representa do total. O gráfico também acompanha o valor gasto (não ignorado) e os ícones de cada categoria.
5. As cinco transações mais recentes, com uma opção de “Ver mais”, que direcionará para a tela de visualização de transações da Seção 4.4.4.

Figura 4 – Protótipo: relatório



Fonte: O Autor (2024).

4.4.4 Visualização de transações

O usuário deve conseguir acessar o protótipo, visualizar a lista das últimas transações e clicar em uma opção de "Ver mais" para listar todas as transações das suas contas no mês que estava selecionado. Essas transações devem conter as informações de descrição (campo "*/data/transactionName*" do *endpoint* de transações do *Open Finance* apresentado na Tabela 1), valor (campo "*/data/transactionAmount/amount*"), data (campo "*/data/transactionDateTime*") e uma representação da categoria, descrita pelo seu ícone.

Além de visualizar, o usuário poderá selecionar uma transação para alterar os atributos de categoria e "ignorar". Ao usuário manter pressionada uma transação da listagem deverão aparecer duas opções "Ignorar" e "Editar categoria", sendo que a opção de editar categoria deverá levar para a tela de categorização, porém somente para a transação que estava selecionada.

4.4.4.1 Protótipo de interface

Na Figura 5 é apresentado um protótipo da interface que irá mostrar a lista de transações. A imagem mostra uma lista de transações, cada uma com um ícone da categoria em que a transação está incluída (representado pelo cifrão no protótipo), descrição, data e valor.

Figura 5 – Protótipo: listagem de transações

Fonte: O Autor (2024).

4.4.5 Categorização de transações

O usuário, ao acessar o protótipo de aplicação deve ter a possibilidade de categorizar as transações. Na página inicial (Seção 4.4.3), o usuário terá a opção clicar no cartão de "Novas transações" que irá levar para uma tela com todas as transações "novas", ou seja, as que ainda não foram categorizadas. O usuário irá ver uma por vez, selecionando a categoria em que ela se encaixa.

4.4.5.1 Protótipo de interface

Na Figura 6 é apresentado um protótipo de interface da tela de categorização. A tela é composta por uma pilha de cartões, que representa as transações não categorizadas, sendo que o primeiro é a transação em questão.

O cartão do topo da pilha mostra o valor, descrição, data e tipo de transação. Abaixo da pilha de transações há uma lista com as categorias e seus ícones. Além disso, é possível, a partir do botão de "Editar categorias" do canto superior direito, acessar a tela de gerenciamento de categorias, que possibilita editar, excluir ou criar uma categoria.

Figura 6 – Protótipo: categorização

Fonte: O Autor (2024).

4.4.6 Marcar transações como ignoradas

O usuário, ao acessar a página de categorização de transações, poderá também marcar uma transação como "ignorada". Uma transação ignorada não é considerada no relatório, no somatório de gastos, receitas, ou gastos por categoria.

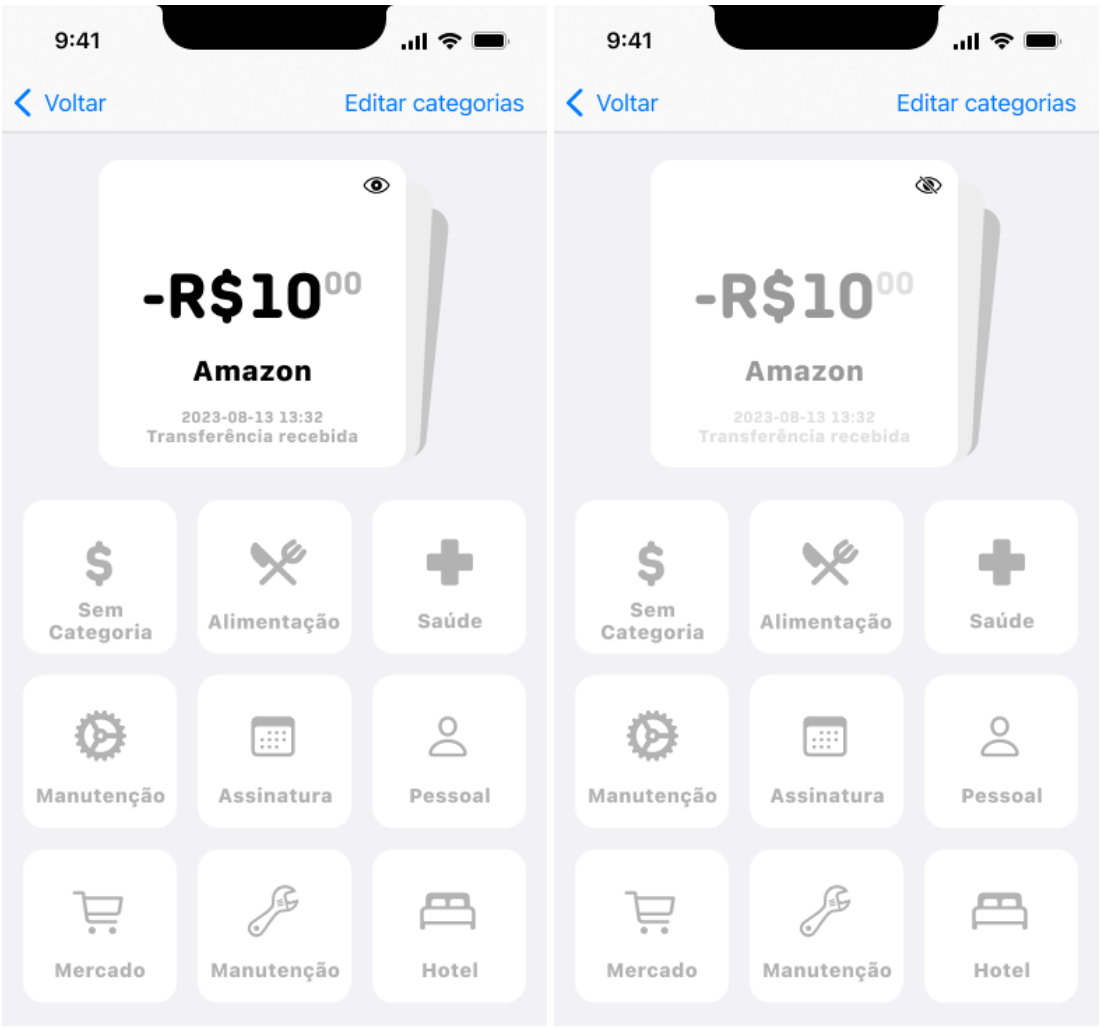
A opção de ignorar transações é importante para garantir que os valores do relatório não fiquem fora da realidade. Um exemplo de uso da função de ignorar seria em uma transferência entre duas contas integradas: o valor transferido iria ser somado tanto como uma despesa (em uma das contas) quanto como receita (na outra conta), fazendo com que os valores de gastos e receitas do mês fiquem fora da realidade, mesmo que com o mesmo saldo.

4.4.6.1 Protótipo de interface

A Figura 7 mostra a tela de categorização (mesma tela da Seção 4.4.5) com a funcionalidade de ignorar a transação. A seleção para ignorar ou não uma transação é através do ícone de olho no canto superior direito do cartão de transação. A Figura 7b mostra uma transação ignorada, identificada por estar com os textos mais acinzentados do que a transação não ignorada,

da Figura 7a. A representação de textos com uma cor mais clara que o normal para transações marcadas como ignoradas deve ser usada também na listagem de transações do requisito da Seção 4.4.4.

Figura 7 – Protótipo: ignorar transações



(a) Protótipo: ignorar desativado (b) Protótipo: ignorar ativado

Fonte: O Autor (2024).

4.4.7 Gerenciamento de categorias

O usuário deve ter a possibilidade de gerenciar as categorias. Na tela de categorização (Seção 4.4.5), ao clicar em "Editar categorias", deve mostrar uma lista com todas as categorias criadas[10], assim como uma opção de criar uma nova categoria.

Cada categoria deve conter um nome e um ícone, que poderão ser alterados posteriormente. Uma categoria pode ser deletada, porém, ao deletar, todas as transações atribuídas à essa

[10] A categoria padrão, chamada "Sem categoria" não deve aparecer já que não foi criada pelo usuário e não é editável.

categoria irão ser movidas para a categoria padrão, chamada “Sem Categoria”.

4.4.7.1 Protótipo de interface

A Figura 8 mostra três protótipos de interface de edição e criação de categorias. A Figura 8a é uma lista de todas as categorias criadas, acompanhadas de seus respectivos nomes e ícones. A mesma lista será responsável por excluir as categorias, com o gesto padrão do *iOS*, arrastando o item da lista para o lado, como representado pela Figura 8b.

A criação de categorias, apresentada na Figura 8c será acessível através de botão de “Criar”, no cabeçalho da tela de listagem. A tela de criação irá sobrepor parte da tela de listagem, com um quadro para seleção de ícone (ao clicar no ícone irá abrir uma listagem com todos ícones disponíveis) e um campo para o nome da categoria a ser criada. A edição de categorias existentes será através da mesma tela de criação, porém ao invés de aparecer com nome em branco e ícone padrão, irá aparecer com os valores da categoria a ser alterada.

Figura 8 – Protótipo: gerenciamento de categorias

(a) Listagem de categorias (b) Exclusão de categoria

(c) Criação de categoria

Fonte: O Autor (2024).

4.4.8 Atualização de transações

O usuário deve ter a possibilidade de forçar uma atualização na base de dados. A tela do relatório (Seção 4.4.3) deve possuir um botão que irá acionar o *pipeline* de ETL, buscando novas transações na Pluggy (Seção 3.2), e, de forma assíncrona, atualizar o banco de dados da aplicação.

4.5 DIAGRAMA LÓGICO DO BANCO DE DADOS

O diagrama de banco de dados é um modelo lógico que mostra os atributos e as relações das tabelas de um banco de dados. A solução irá utilizar dois bancos de dados diferentes, um em nuvem e outro local, no protótipo de aplicação. A estrutura de tabelas e colunas dos dois bancos será similar, as únicas diferenças serão que no banco local irão existir colunas para controlar as modificações de dados realizadas pelo usuário no protótipo de aplicação e o banco em nuvem terá uma tabela de contas bancárias, que será usada para facilitar a extração de dados, porém não será necessária para o protótipo de aplicação.

A Figura 9 representa o diagrama do banco de dados que será usado para armazenar os dados transformados da extração da *Pluggy* e as informações usadas pelo protótipo de aplicação. O banco de dados consiste em três tabelas: “f_transacao” para a listagem de transações, “d_conta” para contas bancárias e “d_categoria” para informações das categorias criadas pelo usuário.

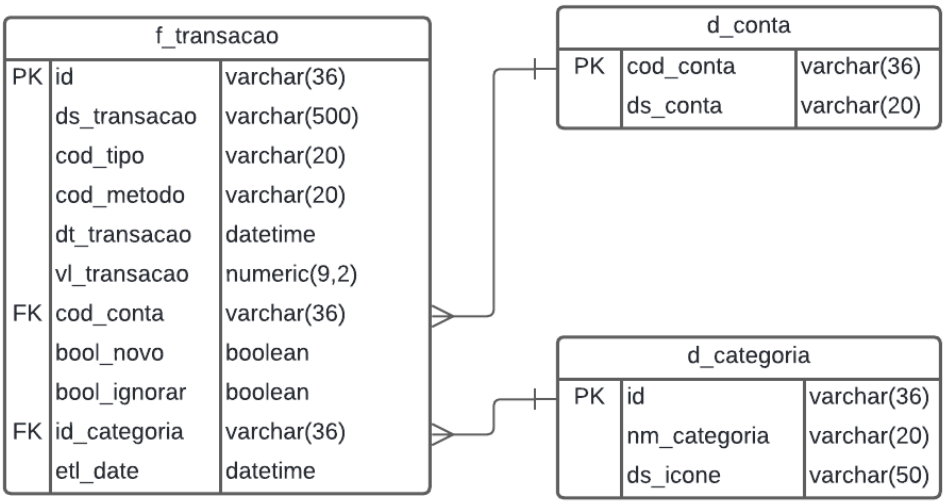
A tabela de transações tem tanto colunas com informações provenientes da *Pluggy*, quanto colunas com informações gerenciadas somente pelo protótipo de aplicação. As colunas gerenciados pelo protótipo são: “bool_novo”, “bool_ignorar” e “id_categoria”. Esses três campos irão receber um valor padrão durante a etapa de carga: verdadeiro para o “bool_novo”, falso para o “bool_ignorar” e o *Id* de uma categoria chamada “Sem categoria”, em que todas as transações serão direcionadas inicialmente. Os valores dessas três colunas podem ser alterados pelo usuário ao realizar a classificação das transações, como representado no requisito da Seção 4.4.5.

A tabela “d_conta” será populada manualmente ao configurar a solução, diretamente pelo banco de dados, para passar a identificação (“cod_conta”) das contas *Pluggy* e definir um apelido para cada conta bancária cadastrada na plataforma da *Pluggy*.

A tabela “d_categoria” será composta por um código (“cod_categoria”), que será definido aleatoriamente pelo protótipo de aplicação, um nome, definido pelo usuário e um código de ícone “*SF Symbols*”[11]. *SF Symbols* são ícones nativos do *Swift*, que o usuário poderá definir na criação ou edição da categoria, como mostrado nos protótipos de gerenciamento de categoria da Figura 8.

[11] <www.developer.apple.com/sf-symbols>

Figura 9 – Diagrama lógico de banco de dados - nuvem

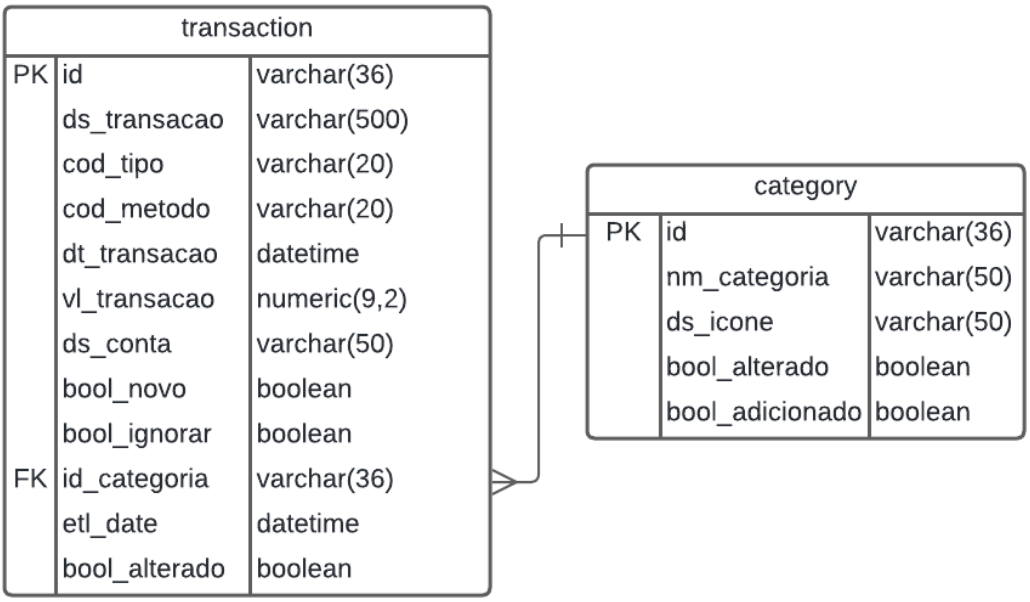


Fonte: O Autor (2024).

Por outro lado, a Figura 10 representa o diagrama do banco de dados do protótipo de aplicação, que terá uma tabela chamada "*transaction*", equivalente à "f_transacao" e "category", equivalente à "d_categoria". As diferenças para as tabelas da nuvem são:

- Tabela "*transaction*":
    1. "ds_conta": ao invés de ter um código de conta, o banco de dados do protótipo armazena diretamente a descrição da conta, como da tabela "d_conta" da nuvem.
    2. "bool_alterado": coluna que auxilia no controle de quando uma informação de transação foi alterada no protótipo de aplicação, para enviar essa atualização de dados para a nuvem.
- Tabela "*category*":
    1. "bool_alterado": assim como o da "*transaction*", controla quando uma informação de categoria foi alterada no protótipo de aplicação.
    2. "bool_adicionado": auxilia no controle de categorias adicionadas no protótipo de aplicação que ainda não foram enviadas para nuvem. É utilizada uma coluna diferente do "bool_alterado" já que a requisição para comunicar para a nuvem categorias novas não é a mesma usada para edição de categorias.

Figura 10 – Diagrama lógico de banco de dados - local



Fonte: O Autor (2024).

5 DESENVOLVIMENTO DA INTEGRAÇÃO

O desenvolvimento da integração foi separado em três etapas sequenciais: a preparação de ambientes (Seção 5.1 e Seção 5.2), desenvolvimento da API (Seção 5.3) e desenvolvimento do *pipeline* de ETL (Seção 5.4).

5.1 PREPARAÇÃO *PLUGGY*

Conforme representado pelo diagrama da Figura 11, o processo para preparar o ambiente responsável por disponibilizar os dados foi:

1-2. **Adicionar conta:** que consiste em conectar uma conta bancária que irá compartilhar os dados através do *Open Finance*. O fluxo para conectar a conta pode variar dependendo da instituição. No exemplo da Figura 12 foram adicionadas três contas bancárias: "Nubank", "Bradesco" e "XP Banking". O fluxo de autenticação dessas três contas foram iguais, primeiramente foi solicitado o CPF do titular e, após a inserção do CPF, é mostrado um código QR que direciona para o aplicativo da instituição, onde é realizada a autorização do compartilhamento dos dados com a *Pluggy*.

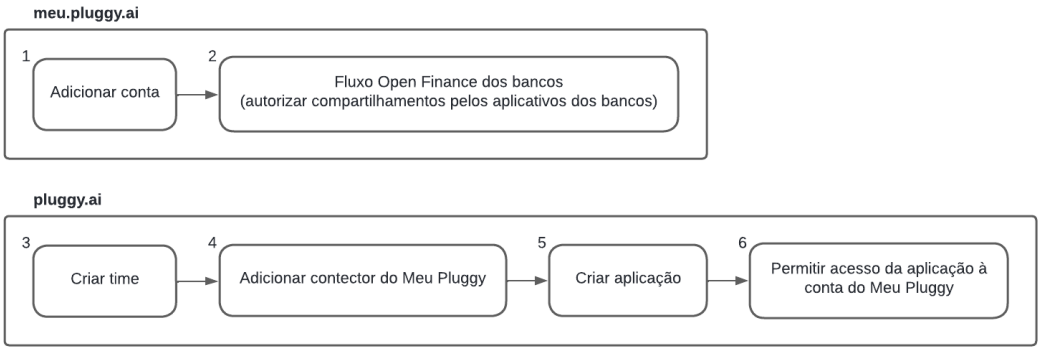
3. **Criar time:** o time da *Pluggy* se trata de um grupo administrado por um ou mais membros que contém as aplicações e conectores. Os planos (pagos ou não) são atribuídos aos times.

4. **Adicionar conector:** conectores são responsáveis por fazer a conexão entre os dados das instituições com o time. Os conectores, normalmente são das instituições financeiras, a única exceção são os conectores da própria *Pluggy*, como o "Meu *Pluggy*" que foi o utilizado neste desenvolvimento. A seleção de conector ainda não garante o acesso dos dados para o time, serve somente para especificar quais conectores estarão disponíveis para as aplicações do time.

5. **Criar aplicação:** a aplicação é responsável pelo acesso aos dados, a ela são atribuídas as contas dos conectores permitidos no time da aplicação. Além de ser responsável pela permissão de acesso às contas, também disponibiliza uma credencial (*Client ID* e *Client Secret*), que é utilizada para a extração dos dados através da API da *Pluggy*.

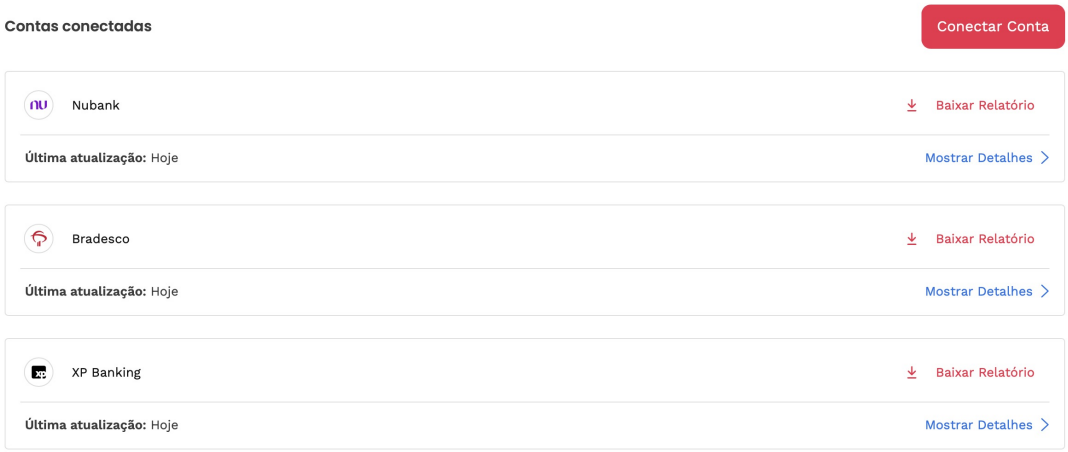
6. **Acesso da aplicação ao "Meu *Pluggy*":** através da *preview* da aplicação disponibilizada pela *Pluggy*, é possível adicionar as contas daqueles conectores permitidos no time da aplicação. Neste caso, foi adicionada a conta do "Meu *Pluggy*", herdando todas as contas bancárias que foram adicionadas nas etapas 1-2.

Figura 11 – Fluxograma *Pluggy*



Fonte: O Autor (2024).

Figura 12 – Contas conectadas Meu *Pluggy*



Fonte: O Autor (2024).

Com todas etapas de preparação da *Pluggy*, o resultado é uma credencial de aplicação (*Client ID* e *Client Secret*) que tem acesso à três contas bancárias diferentes. Com essa credencial é possível utilizar a API da *Pluggy* para acessar os dados, padronizados pelo *Open Finance*, dessas contas.

Outro passo realizado, após a configuração do ambiente da *Pluggy* foi obter os Id's das contas conectadas. Cada conexão de instituição do Meu *Pluggy* é representada por um *ItemId*. Um *Item* pode possuir mais de um produto financeiro (da mesma instituição) vinculado, chamado de "*Account*". Todas as instituições usadas neste desenvolvimento possuem como produto uma conta corrente (chamada de "BANK") e uma conta para o cartão de crédito (chamada de "CREDIT").

Através da *preview* da aplicação (mesmo ambiente em que foram adicionadas as contas do Meu *Pluggy* à aplicação), é possível obter o *Id* de cada um dos *Item's* conectados. Com os Id's de itens, é possível fazer requisições no *endpoint* "/accounts", que lista todos os produtos

vinculados àquele *Item*. Um exemplo do retorno do *endpoint* pode ser encontrado no Apêndice A.

5.2 PREPARAÇÃO NUVEM

Todos os recursos de computação em nuvem utilizados são da *Azure*, *cloud* da *Microsoft*. Os recursos utilizados foram:

- Banco de dados *Azure* SQL;
- Serviço de aplicativo web;
- Cofre de chaves.

O cofre de chaves[1] foi o responsável por armazenar os segredos[2] usados pela API da solução. Foram utilizados dois cofres diferentes, um para as credenciais da API e outro para os ID's das contas bancárias da *Pluggy*. Os segredos das credenciais da API foram: *Client ID*, *Client Secret*, tanto do Entra ID da aplicação quanto da *Pluggy* e a *string* de conexão do banco de dados. Enquanto o cofre de chaves das contas bancárias contém um segredo para cada instituição financeira (“Bradesco”, “Nubank” e “XP”). Os segredos das contas contém, em formato de JSON, o ID da conta corrente, ID do cartão de crédito e o itemId do conector da conta da instituição, que une os dois produtos. Um exemplo de segredo da conta XP pode ser encontrado no Apêndice B.

A única configuração necessária para o banco de dados foi de segurança, criando regras de *firewall* no servidor do banco de dados, habilitando o IP da rede local para acessar o banco de dados e a criação de um usuário com permissões de leitura e escrita para ser usado pela API da solução.

O terceiro recurso *cloud* utilizado foi o serviço de aplicativo web para hospedar a API da solução. Conforme apresentado na Seção 4.2.1, foi utilizado um modelo de código *Flask* disponibilizado pela própria *Microsoft*, que já conta com a autenticação via *Entra ID* implementada. Para realizar a implantação do código no serviço foi configurado um repositório no *GitHub*[3] que realiza o *deploy* automático no serviço de aplicativo web. Além disso, também foi configurada uma identidade gerenciada, que é responsável por acessar os cofres de chaves quando a aplicação está rodando na nuvem. A *Azure* também disponibiliza um subdomínio padrão, iniciando com o nome da aplicação web criada e terminando com “.azurewebsites.net”,

[1] Cofre de chaves é um serviço responsável por armazenar dados criptografados. Esses dados podem ser em formatos de chaves, certificados ou segredos, que foi o único formato utilizado nesse desenvolvimento.

[2] Segredos são usados para salvar parâmetros em formato de cadeia de caracteres que não podem ser escritos diretamente no código fonte por motivos de segurança. Por exemplo senhas e *strings* de conexão com banco de dados.

[3] <www.github.com>

dispensando a necessidade de contratar um domínio para hospedar a API. Mais detalhes sobre os planos e custo com a infraestrutura em nuvem podem ser vistos no Apêndice C.

5.3 DESENVOLVIMENTO DA API

A Figura 13 mostra o funcionamento geral da API desenvolvida. Após a inicialização da API, são criados objetos da classe "*Secrets*", que será abordada na Seção 5.3.1 e da classe "Database" (Seção 5.3.2), que recebe como parâmetro de inicialização a *string* de conexão do banco de dados obtida através dos segredos. Os dois objetos serão utilizados durante toda a execução da API, independente de estar recebendo consultas ou não. Após a instanciação dos dois objetos, a API fica disponível para processar as requisições. Como exemplo, o fluxograma mostra uma requisição *POST* para o *endpoint* de transações, que será abordado a fundo no Seção 5.3.3. De forma geral, essa requisição é um gatilho para o *pipeline* de integração, que realiza todo processo de ETL e retorna uma lista de todas transações disponíveis para atualizar o protótipo de aplicação.

O momento zero do fluxo é a realização da requisição, que não necessita de nenhum parâmetro adicional (além do parâmetro padrão de autenticação, abordado na Seção 5.3.3). Essa requisição aciona o *pipeline* de ETL, que é visto de forma simplificada no fluxograma, mas que na realidade é composto por três classes que serão abordadas na Seção 5.4. O momento de número um é a obtenção dos segredos necessários para extração dos dados, vindos do objeto "*Secrets*". Utilizando os segredos como parâmetros, o *pipeline* realiza a extração da API da *Pluggy*, que retorna uma lista com todas as transações disponíveis, caracterizando o terceiro momento. No quarto momento do fluxograma são aplicadas transformações nas transações extraídas. As transações transformadas são repassadas para o objeto "*Database*" para realizar o *merge* na etapa cinco. O último momento da requisição é o retorno da lista de transações, vindo do objeto *Database*.

Uma versão mais detalhada do fluxo de requisições da API pode ser visto no Apêndice D, que mostra um diagrama de sequência do mesmo *endpoint* e também do *endpoint* responsável por retornar os dados necessários para o protótipo de aplicação.

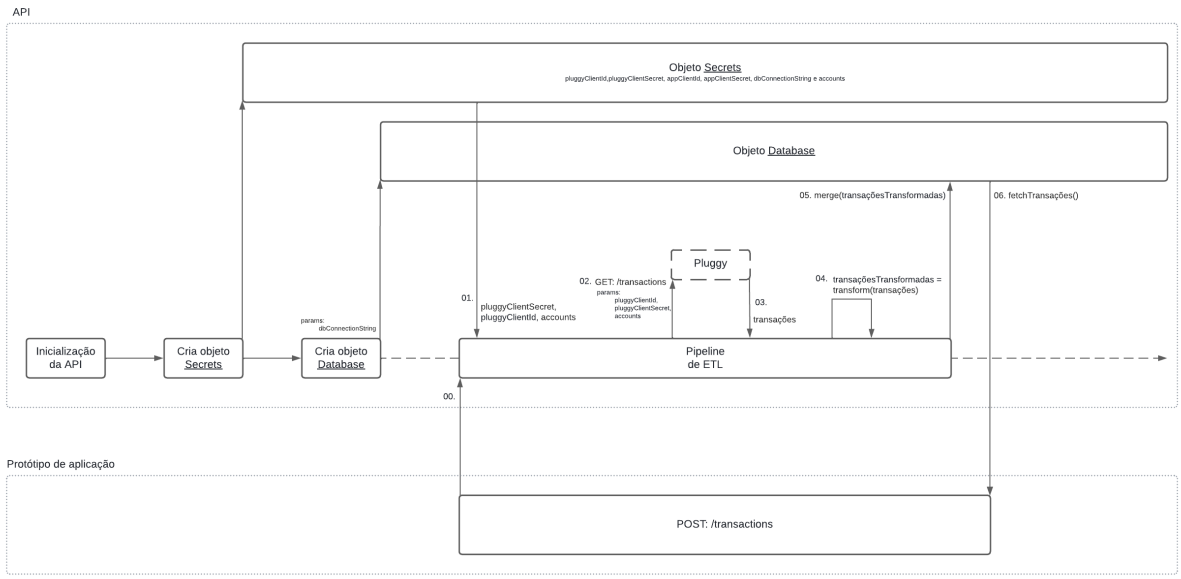
5.3.1 Segredos

A classe "*Secrets*" foi criada para gerenciar todos os segredos do cofre de chaves *Azure*. Ela utiliza as bibliotecas "*azure.keyvault.secrets*"[4] para acessar o conteúdos dos cofres de chaves e "*azure.identity*"[5] para realizar a autenticação na *Azure*.

A autenticação é configurada de acordo com o ambiente de execução. No ambiente de produção (nuvem), a biblioteca "*azure.identity*" utiliza a identidade gerenciada (configurada na

[4] <https://learn.microsoft.com/en-us/python/api/overview/azure/keyvault-secrets-readme?view=azure-python>
[5] <https://learn.microsoft.com/en-us/python/api/overview/azure/identity-readme?view=azure-python>

Figura 13 – Fluxograma API



Fonte: O Autor (2024).

Seção 5.2). Enquanto no ambiente de desenvolvimento (máquina local), ela é a identidade do usuário configurada no "interface de linha de comando (CLI) do *Azure*"[6].

Para acessar os segredos, a classe cria dois objetos da classe "*SecretClient*" da biblioteca *azure.keyvault.secrets* - um para cada cofre de chaves. Cada instância é inicializada com o URL do cofre e a credencial. Com esses objetos, a classe pode obter o valor de cada segredo utilizando o método "get_secret", que recebe o nome do segredo como parâmetro. Assim, são criados cinco atributos para armazenar os segredos da API: "*pluggyClientId*", "*pluggyClientSecret*", "*appClientId*", "*appClientSecret*" e "*dbConnectionString*".

O sexto atributo, "accounts", armazena os segredos relacionados às contas bancárias. O nome dos segredos deste cofre podem variar de acordo com as contas, portanto, é necessário obter uma lista com os segredos disponíveis no cofre. A classe armazena esses segredos em uma lista de dicionários, em que cada entrada usa o nome do segredo como chave e o valor do segredo como valor correspondente do dicionário (um exemplo deste valor pode ser visto no Apêndice B).

A classe "*Secrets*" é instanciada na primeira execução da aplicação e armazenada como um contexto de aplicação no *Flask*, por meio do módulo "g"[7]. Esses segredos são utilizados posteriormente para conectar ao banco de dados da solução e extrair os dados da *Pluggy*.

6 <https://learn.microsoft.com/pt-br/cli/azure/authenticate-azure-cli-interactively>
7 <https://flask.palletsprojects.com/en/stable/appcontext/>

5.3.2 Banco de dados

Para realizar o gerenciamento e a obtenção de dados no banco de dados, foi implementada uma classe denominada "*Database*". Esta classe recebe, como parâmetro de inicialização, uma *string* de conexão do banco de dados, a qual é armazenada na classe *secrets*, conforme descrito na Seção 5.3.1. Para a conexão com o banco de dados, a classe utiliza a biblioteca "pyodbc"[8]. A classe, assim como a *Secrets*, é instanciada no momento da inicialização da API e é armazenada como um contexto de aplicação do *Flask*.

Os métodos da classe permitem realizar consultas nas tabelas de transações e categorias, retornando os resultados em listas de dicionários, prontos para serem retornados pelos *endpoints* utilizados pelo protótipo de aplicação. Além de métodos para obter dados das tabelas de transações e categorias, a classe oferece métodos para: inserir transações, que será usado pelo *pipeline* de ETL, conforme explicado na Seção 5.4.3; atualizar os atributos gerenciados pelo protótipo de uma transação; e criar, atualizar e deletar categorias.

5.3.3 *Endpoints* da API da solução

Os *endpoints* implementados na aplicação permitem o gerenciamento das entidades de categorias e transações. Todas as requisições para a API precisam estar acompanhadas de um *token* de acesso da plataforma de identidade da *Microsoft*. Como explicado anteriormente, na Seção 4.2.1, foi utilizado um modelo de código *Flask* da própria *Microsoft* com a autenticação já implementada. Em toda requisição, a API verifica se o *token* é válido e se tem acesso ao recurso da aplicação web, se não, é retornado um erro de "Token inválido". Abaixo estão descritos os *endpoints* e os métodos disponíveis para cada operação. Uma versão detalhada desta lista, contendo o corpo da requisição e resposta pode ser vista no Apêndice E.

- ***/category***: gerencia todas as operações relacionadas a categorias, oferecendo os métodos:
  - ***GET***: recupera todas as categorias cadastradas.
  - ***POST***: cria novas categorias.
  - ***PUT***: atualiza uma ou mais categorias existentes.
  - ***DELETE***: exclui uma ou mais categorias.
- ***/transaction***: gerencia todas as operações relacionadas a categorias, oferecendo os métodos:
  - ***GET***: recupera todas as transações do banco de dados.
  - ***POST***: aciona o gatilho para realizar o processo de ETL, inserindo as novas transações.

[8] <https://pypi.org/project/pyodbc/>

  - ***PUT***: altera os atributos gerenciados pelo protótipo de aplicação de uma ou mais transações existentes.

- ***/data***: retorna todos os dados necessários para o protótipo de aplicação atualizar o banco de dados local. Este *endpoint* aceita somente requisições de *GET*:

  - ***GET***: recupera a lista de todas as transações e categorias, ou seja, todas as informações necessárias para atualizar o banco de dados local do protótipo de aplicação.

5.4 DESENVOLVIMENTO DO *PIPELINE* DE DADOS

O *pipeline* de dados, como dito na Seção 5.3.3, é acionado através do *endpoint* "*/transaction*" utilizando o método "*POST*". Toda vez que é realizada uma requisição para o *endpoint*, são instanciados três objetos: um para extração, que será mostrado na Seção 5.4.1; para transformação, da Seção 5.4.2; e para carga, Seção 5.4.3.

5.4.1 Extração

Toda a extração de dados da solução é realizada por uma classe. Essa classe possui como atributos o URL base da *Pluggy*, clientId e clientSecret da *Pluggy* e uma lista de Id's de contas (que são obtidos através do cofre de chaves). Quando um objeto da classe de extração é instanciado, um *token* de acesso é gerado com o *clientId* e *clientSecret*, que tem a validade de noventa minutos. A classe implementa métodos para requisição HTTP, assim como métodos específicos para extrair as transações e gerar *tokens* de acesso.

Todos dados extraídos da solução são provenientes da *Pluggy*, através do *endpoint* "*/transactions*". O Apêndice F mostra um fluxograma do método responsável por extrair as transações da *Pluggy*, enquanto o Apêndice G mostra como exemplo duas transações retornadas pelo mesmo *endpoint*. A saída da etapa de extração é uma lista com as transações, em formato de dicionários, de todas as contas configuradas no cofre de chaves de contas.

A ideia inicial da extração seria realizar uma primeira carga completa e as seguintes serem apenas incrementais, interrompendo a extração na primeira transação com data mais antiga que a última carga realizada e fazendo o *merge* pelo "Id" da transação. Porém, durante o desenvolvimento, foi observado que algumas transações estavam sendo duplicadas. Apesar de conterem Id's e descrições diferentes, se tratavam de uma única operação, que aparecia somente uma vez no extrato. De acordo com a própria *Pluggy*, em sua documentação, as transações podem ser deletadas e substituídas com um novo Id caso algum campo, como a descrição, seja modificado pela instituição financeira.

Por esse motivo, foi decidido não realizar extrações incrementais, e sim sempre realizar a extração de todas as transações disponíveis na *Pluggy*, assim, dispensando a necessidade de controlar as transações que foram deletadas e re-inseridas pela *Pluggy*. Outra possibilidade seria

receber um gatilho vindo da *Pluggy* para excluir as transações que foram excluídas da base de dados (que trocaram de Id), porém, foi definido sempre realizar a extração completa, devido ao baixo volume de dados, de menos de duas mil transações combinando as três instituições.

5.4.2 Transformação

A transformação dos dados extraídos da *Pluggy* foi realizada por outra classe específica para transformação. A classe recebe os dados de transações vindo da *Pluggy* de forma bruta, em formato de lista de dicionários (lista única para todas as contas) e a transforma em um *DataFrame* da biblioteca *Pandas*. No *DataFrame*, a primeira etapa de transformação é selecionar somente as colunas que serão usadas: "id", "*description*", "*type*", "*amount*", "*date*", "*paymentData*", "creditCardMetadata" e "accountId".

Com as colunas selecionadas, é realizada uma renomeação, que substitui os nomes vindos da API conforme apresentado na Tabela 2.

Tabela 2 – Colunas renomeadas após transformação

Nome *Pluggy*	**Novo nome**
id	id
description	ds_transacao
type	cod_tipo
amount	vl_transacao
date	dt_transacao
paymentData	tempDadosTransacao
creditCardMetadata	tempDadosCredito
accountId	cod_conta

Com os campos renomeados são realizados os seguintes tratamentos nos valores:

- **cod_metodo**: é criada uma coluna chamada de cod_metodo, que apresenta os valores: "Recebimento", caso o cod_tipo seja "CREDIT" (valor positivo); "Pix", quando a descrição da transação contém a palavra "Pix"; "Crédito", caso o campo tempDadosCredito não esteja vazio; "Débito", caso não tenha se encaixado em nenhuma regra anterior.

- **vl_transacao**: o valor da transação é multiplicado por -1 caso o cod_tipo da transação seja "DEBIT" (valor negativo).

- **ds_transacao**: para as transações *Pix* originadas do *Nubank*, a descrição é alterada para concatenar o "documentNumber" do pagador, caso seja um valor positivo, ou do recebedor, caso seja um valor negativo. Este processamento é feito para auxiliar na identificação da transação, já que as descrições de transações *Pix* do *Nubank* contêm somente "Transferência Enviada" ou "Transferência Recebida".

Após as transformações de valores, os campos “tempDadosTransacao” e “tempDadosCredito” são eliminados do *DataFrame* e as linhas são ordenadas de forma decrescente pela data de transação. Com todas transformações prontas, a saída desta etapa é um *DataFrame* com os dados sanitizados e prontos para serem carregados no banco de dados da solução.

5.4.3 Carga

Como mencionado na Seção 5.4.1, todas as transações são extraídas em todas execuções do *pipeline* da solução. Desta forma, para realizar a carga dos novos dados não é possível realizar somente uma inserção. Também não é possível deletar todos os dados já carregados no banco e substituir pelos novos, já que os atributos gerenciados pelo protótipo, como as categorias, seriam perdidos.

Para resolver o problema, é realizado um processo de *merge* personalizado, representado pela Figura 14. Esse processo pode ser dividido em três etapas principais:

1. Verifica se existe algum registro no banco de dados que não existe nos novos dados extraídos da *Pluggy*, comparando pelo “Id”.

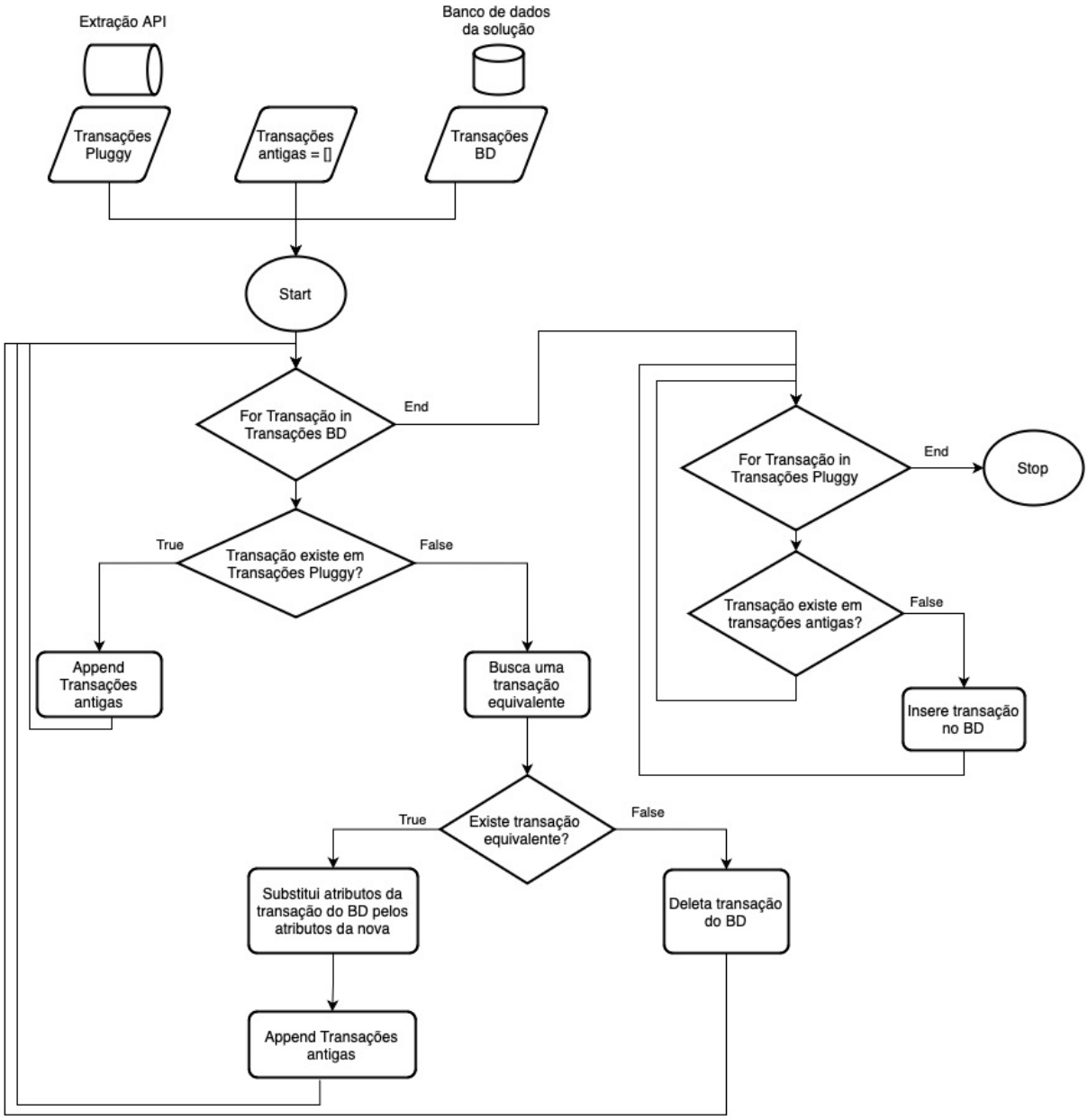
2. Caso exista algum registro que se encaixe nesse caso, busca entre as novas transações da extração (as transações que ainda não existem no banco de dados) uma transação equivalente à esse registro, com o mesmo valor, data, hora e conta.

    a) Se existir uma transação equivalente, realiza um *merge*, substituindo os atributos (incluindo o “Id”) da transação do banco de dados pelos atributos da nova transação. Os únicos atributos que não são substituídos pelos novos são os atributos gerenciados pelo protótipo (que não são provenientes da extração dos dados).

    b) Se não existir nenhuma transação equivalente entre as extraídas pela *Pluggy*, a transação do banco de dados é deletada.

3. Os registros novos, que não se encaixam em uma atualização de atributos, são adicionados na tabela com a categoria padrão (“Sem categoria”) e identificador de “Ignorar” padrão (falso). Enquanto os registros vindos da *Pluggy*, que já existem no banco de dados (com o mesmo Id) não são inseridos.

Figura 14 – Fluxograma do *merge*



Fonte: O Autor (2024).

6 DESENVOLVIMENTO DO PROTÓTIPO

Este capítulo irá abordar os principais tópicos do desenvolvimento do protótipo de aplicação, que é responsável pela apresentação dos dados extraídos e gerenciamento dos atributos "categoria" e "ignorar" das transações. Será abordado como o protótipo realiza a autenticação para acessar os dados da API na Seção 6.1; como os dados locais do protótipo estão sendo gerenciados na Seção 6.2 e Seção 6.3; e como as telas de apresentação dos dados foram desenvolvidas na Seção 6.4.

6.1 AUTENTICAÇÃO

A *Microsoft* disponibiliza um modelo de aplicação *Swift* com a lógica de autenticação via *Entra ID* já implementada. No entanto, esse modelo utiliza o "*UIKit*"[1], e não o "SwiftUI"[2], *framework* de interfaces mais recente do *Swift*, lançado em 2019.

Por esse motivo, foi adotado um modelo desenvolvido pela comunidade que adapta a solução da *Microsoft* para *SwiftUI*: o "MSALSwiftUI"[3]. Esse modelo inclui uma tela de login que redireciona o usuário para o link de autenticação da *Microsoft* e, após o login bem-sucedido, retorna uma *struct* com o e-mail e o *token* de acesso do usuário. As informações da *struct* são salvar para consultas futuras à API da solução, e o usuário é redirecionado à tela de relatório do protótipo.

6.2 MODELO DE DADOS

Foram desenvolvidas duas classes para representar as entidades de dados do protótipo de aplicação móvel, "*Transaction*" e "*Category*". Ambas as classes implementam o protocolo "*Codable*" do *Swift*, que possibilita a codificação e decodificação JSON para simplificar o processo de comunicação com a API, evitando conversões manuais.

O banco de dados local utilizado foi uma versão do "*SQLite*"[4] para *Swift*. Como mostrado no Seção 4.5, o banco local terá duas tabelas, uma para transações e uma para categorias, sendo que os atributos das tabelas mantêm uma correspondência direta com os atributos das classes.

---

1 <https://developer.apple.com/documentation/uikit/>
2 <https://developer.apple.com/xcode/swiftui/>
3 <https://github.com/alschmut/MSALSwiftUI>
4 <https://github.com/stephencelis/SQLite.swift>

6.3 GERENCIAMENTO DE DADOS

O protótipo de aplicação foi separado em três classes para gerenciar os dados locais:

- **"*ApiManager*":** responsável por gerenciar toda a comunicação com a API da solução, abstraindo a lógica de manipulação de requisições HTTP. A classe tem como atributo o URL base da API e o "*accessToken*" que foi gerado após o login do usuário pelo fluxo de autenticação do modelo. Possui somente métodos genéricos que recebem o *endpoint*, método e *payload* para realizar a requisição.
- **"*DatabaseManager*":** classe responsável por realizar todo o gerenciamento do banco de dados local. Além de recuperar, incluir, excluir e atualizar registros das tabelas, também precisa controlar o arquivo do banco de dados, e criar as tabelas durante a primeira execução.
- **"*DataManager*":** centraliza todo gerenciamento de dados do protótipo. É responsável por armazenar os dados organizados para a visualização, sincronizar os dados locais com a nuvem, gerenciar o estado de carregamento das informações e fazer atualizações sobre os dados locais.

6.3.1 Classe *DataManager*

"*DataManager*" é a única classe de gerenciamento que é referenciada nas telas do protótipo, ou seja, todos as requisições de API e comandos no banco de dados passam pela classe "*DataManager*". Ela é instanciada no início da execução do aplicativo, logo após o *login* do usuário, e realiza a sincronização dos dados da nuvem com o banco de dados local.

Essa sincronização acontece através de uma carga completa, excluindo todos registros do banco de dados local e inserindo todos registros de transações e categorias retornados pelo *endpoint* "*/data*" da API da solução. Ao receber os registros e armazenar em seu banco local, organiza informações pré-calculadas em memória.

O protótipo de aplicação é composto por telas mensais, portanto, a *DataManager* utiliza outra classe para armazenar as informações necessárias divididas por mês. A *DataManager* utiliza um dicionário de pares chave-valor, onde a chave é uma *string* no formato "ano-mês"(por exemplo, "2024-11") e o valor é um objeto de uma classe chamada "*MonthlyData*". Esta classe, *MonthlyData*, possui como atributos a lista de transações, a soma dos gastos e das receitas, a lista de transações não categorizadas (consideradas como novas) e um dicionário onde cada categoria é uma chave e o valor gasto é o valor, sempre referentes ao mês específico.

Além do dicionário de dados mensais, a classe "*DataManager*" também possui os atributos:

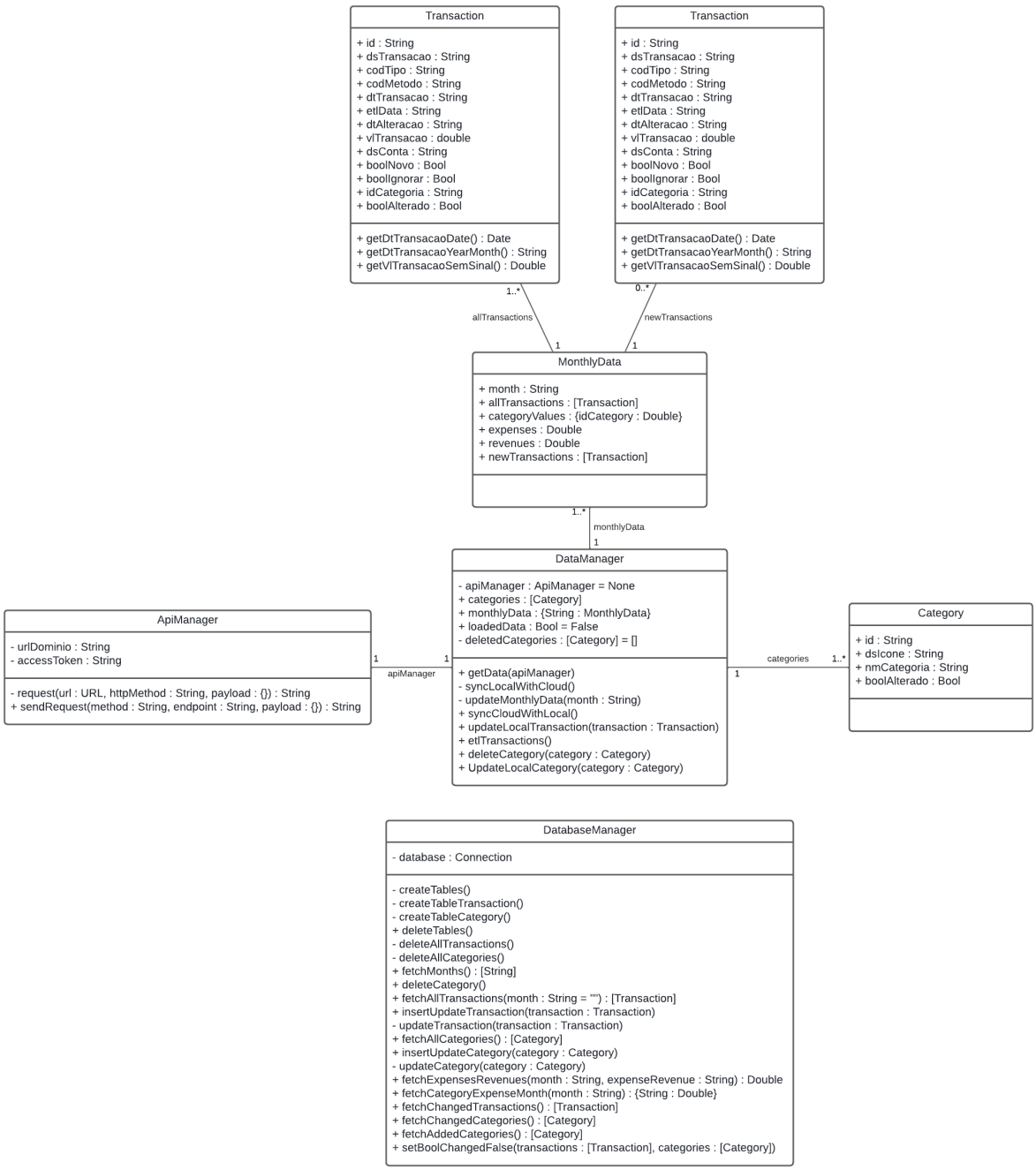
- Lista de objetos categoria, já que as categorias não variam de acordo com o mês, somente seus valores;
- Lista de categorias deletadas durante a sessão;
- *Booleano* que identifica se os dados estão disponíveis para serem visualizados na tela;
- Objeto da classe *ApiManager* para realizar as requisições para a API.

Uma das responsabilidades da classe é controlar as alterações nos dados que devem ser enviadas para a nuvem. Para isso, a *DataManager* utiliza, além da lista de categorias deletadas, a classe *DatabaseManager*, para realizar consultas buscando transações e categorias com os campos "bool_alterado" ou "bool_adicionado" verdadeiros. O envio das atualizações para a nuvem pode acontecer em dois momentos: na saída da tela de categorização de transações ou quando o aplicativo é colocado em segundo plano pelo usuário. Para enviar as atualizações utiliza a classe *ApiManager*, que faz requisições de "*POST*", "*PUT*" ou "*DELETE*" para manter os dados da nuvem sincronizados com o protótipo de aplicação.

6.3.2 Diagrama de classes

O diagrama de classes da solução está representado na Figura 15, destacando as relações e responsabilidades de cada classe. A classe central da solução é a "*DataManager*", que coordena as principais funcionalidades e interações entre os componentes. Ela mantém relação direta com as classes *ApiManager*, *Category* e *MonthlyData*. Já a classe *MonthlyData* apresenta duas relações distintas com a classe *Transaction*: uma lista que armazena todas as transações e outra específica para as transações novas. Por outro lado, a classe *DatabaseManager* é independente, sem relacionamentos diretos com outras classes, sendo instanciada de forma autônoma na inicialização da aplicação.

Figura 15 – Diagrama de classes



Fonte: O Autor (2024).

6.4 TELAS

O protótipo de aplicação é composto por três telas principais: relatório, categorização de novas transações e gerenciamento das categorias. Todas as telas buscaram seguir o padrão dos protótipos da Seção 4.4, com as mesmas funcionalidades e estilo, diferindo apenas o tamanhos de componente e fontes. Todas informações apresentadas nas figuras são dados reais vindos da Pluggy, extraídos das três instituições apresentadas na Seção 5.1: "*Nubank*", "Bradesco" e "XP *Banking*".

6.4.1 Relatório

A página inicial, apresentada na Figura 16, mostra o relatório mensal, que possui na área superior da tela uma seleção de meses no formato “ano-mês”, assim como o dicionário chave-valor da classe *DataManager* e um botão para sincronizar, que aciona o *pipeline* de ETL da API. Na área principal do relatório apresenta:

- O somatório de gastos e receitas que não foram marcadas como “ignoradas” do mês;
- A quantidade de transações não categorizadas, acompanhada da primeira transação não categorizada;
- O somatório de gastos não ignorados por categoria;
- Uma listagem das cinco últimas transações.

A Figura 16a mostra o relatório de um mês em que todas as transações estão categorizadas, enquanto a Figura 16b mostra um mês onde nenhuma transação foi categorizada. A partir da página inicial, é possível ser redirecionado para a tela de categorização de transações da Figura 18 através do cartão “Novas transações”, ou para a listagem de transações da Figura 17, clicando no “Ver tudo” do cartão de transações recentes.

6.4.2 Listagem de transações

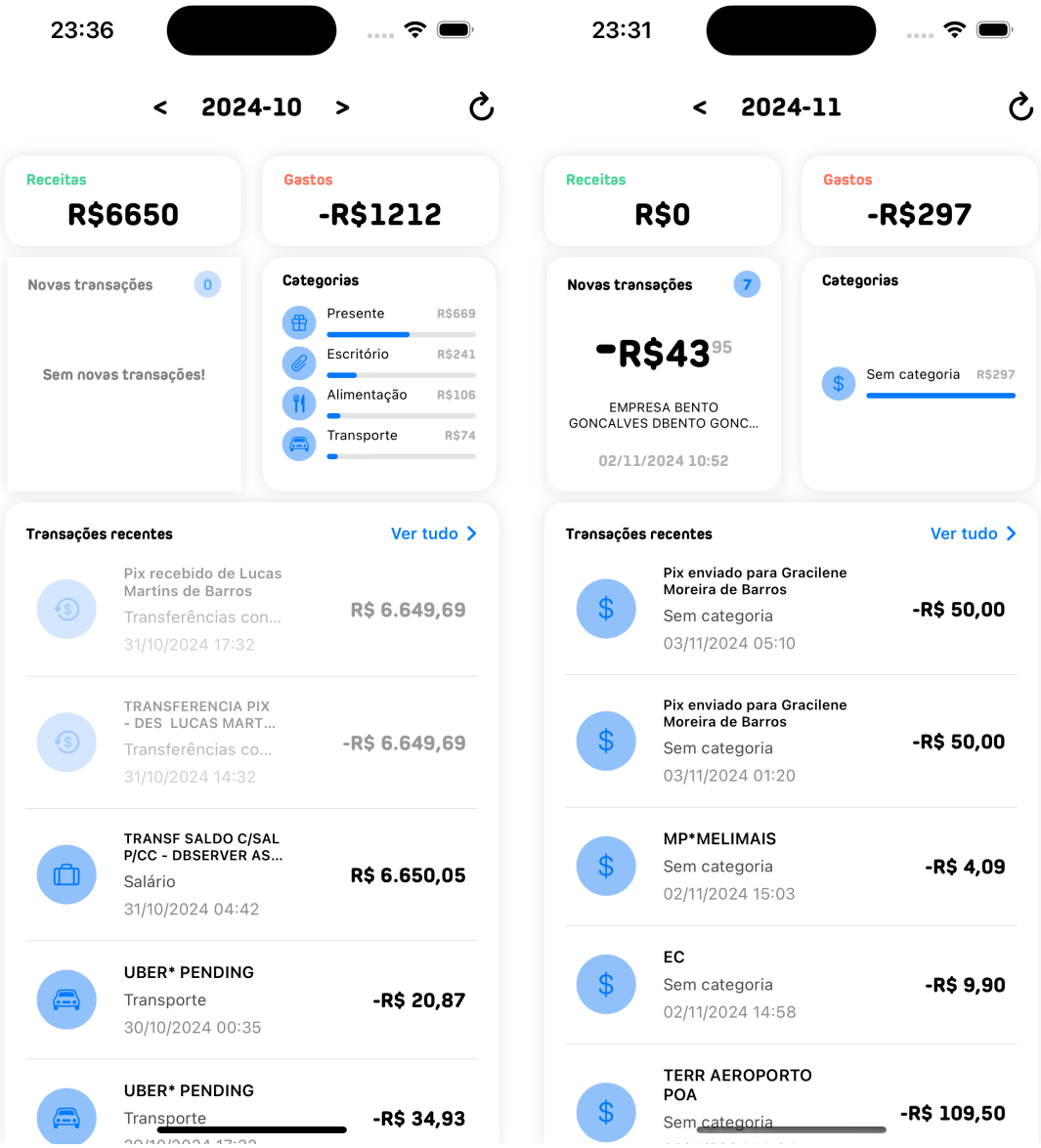
A listagem de transações, apresentada na Figura 17a, mostra todas as transações, marcadas como ignoradas ou não, do mês que estava selecionado. Cada transação tem sua descrição, valor, data, categoria e ícone de categoria. Transações marcadas como ignoradas são mostradas com a opacidade mais baixa (texto acinzentado).

É possível abrir um menu de contexto para editar os atributos de categoria e ignorado de uma transação, como mostrado na Figura 17b. Com o menu aberto, é possível realizar duas ações: “Ignorar” que irá marcar a transação como ignorada; e “Editar categoria”, que irá levar o usuário para a tela de categorização (Figura 18) da transação selecionada.

6.4.3 Categorização

A Figura 18 mostra a tela utilizada para categorizar as transações. No cabeçalho da tela é possível voltar para a página inicial ou ir para a tela de gerenciamento de categorias. A transação que está sendo categorizada é apresentada em um cartão com as informações de valor, descrição, data e conta origem. Além das informações, o cartão também possui o ícone de um olho, que possibilita marcar a transação como ignorada. A redução da opacidade dos textos representa que a transação em questão está ignorada, assim como feito na listagem de transações (Figura 17a).

Figura 16 – Desenvolvimento: relatório

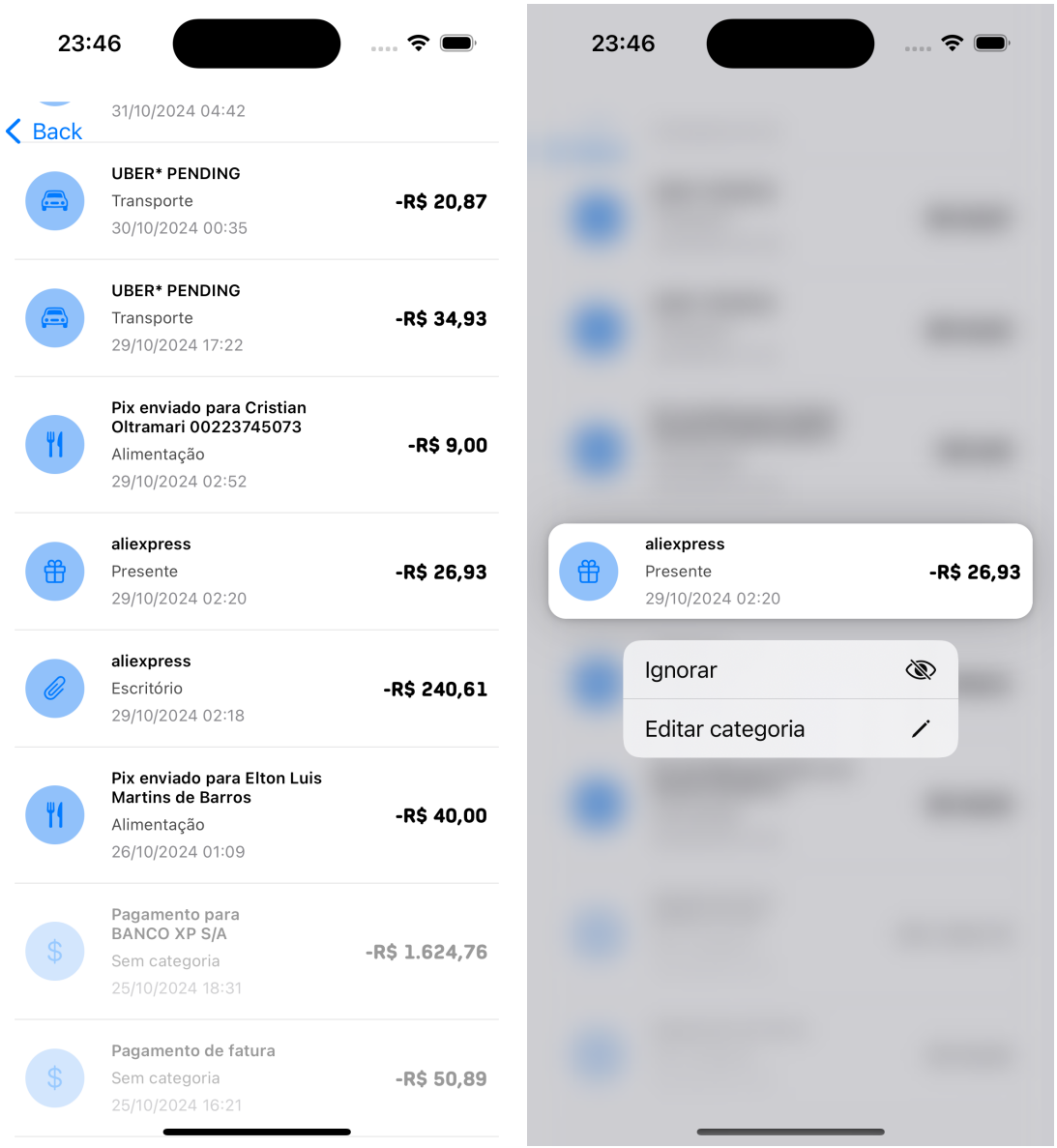


(a) Relatório - mês fechado (b) Relatório - mês corrente

Fonte: O Autor (2024).

Abaixo do cartão da transação existe uma lista com todas as categorias criadas, seus ícones e nomes. Ao usuário clicar em uma das categorias, estará atribuindo a transação em questão à categoria. Quando o usuário categoriza a transação, a classe *DataManager* envia a alteração para o banco de dados local, mudando a categoria, “bool_alterado” para verdadeiro, “bool_novo” para falso e “bool_ignorado” para o que estava selecionado quando o usuário confirmou a categoria. Já o envio da atualização para a nuvem acontece da forma explicaa na Seção 6.3.1.

Figura 17 – Desenvolvimento: listagem de transações



(a) Lista de transações (b) Menu de contexto de uma transação

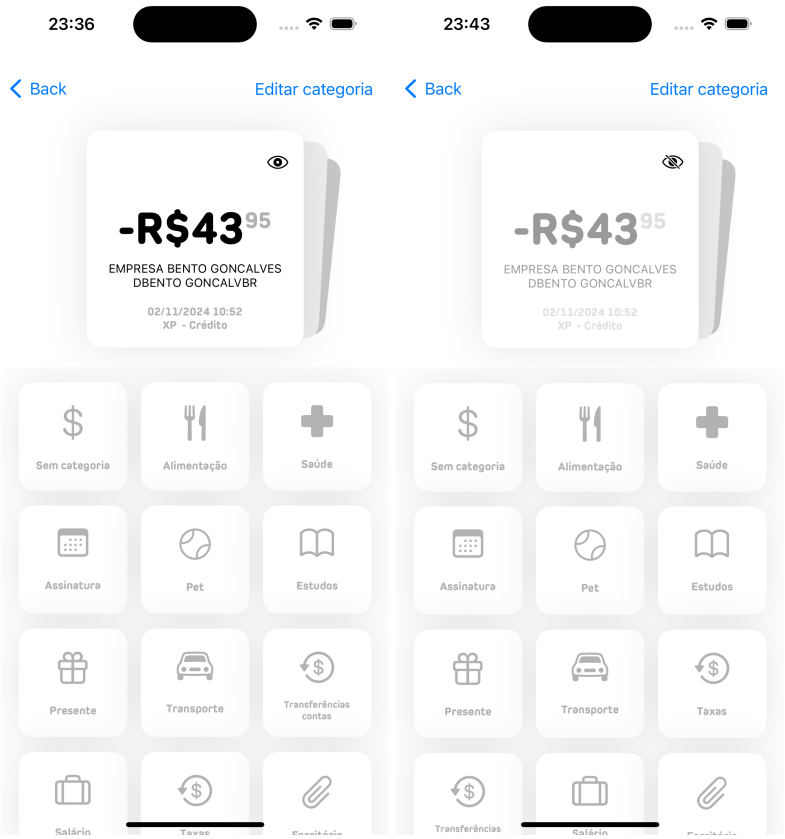
Fonte: O Autor (2024).

6.4.4 Gerenciamento de categorias

A Figura 19 mostra as telas usadas para gerenciar as categorias do protótipo de aplicação. A Figura 19a é composta por uma lista padrão da *SwiftUI*, com o ícone e nome de cada uma das categorias criadas (com exceção da categoria padrão chamada "Sem categoria" que não pode ser editada).

Na lista é possível usar o gesto padrão de arrastar para um dos lados para realizar a ação de *delete*. Quando uma categoria é deletada todas as transações dessa categoria são direcionadas para a categoria padrão ("Sem categoria") e a categoria é incluída na lista de categorias deletadas da classe *DataManager*, que, posteriormente, são enviadas para o endpoint "*/category*" usando

Figura 18 – Desenvolvimento: categorização de transação



(a) Transação não ignorada (b) Transação ignorada

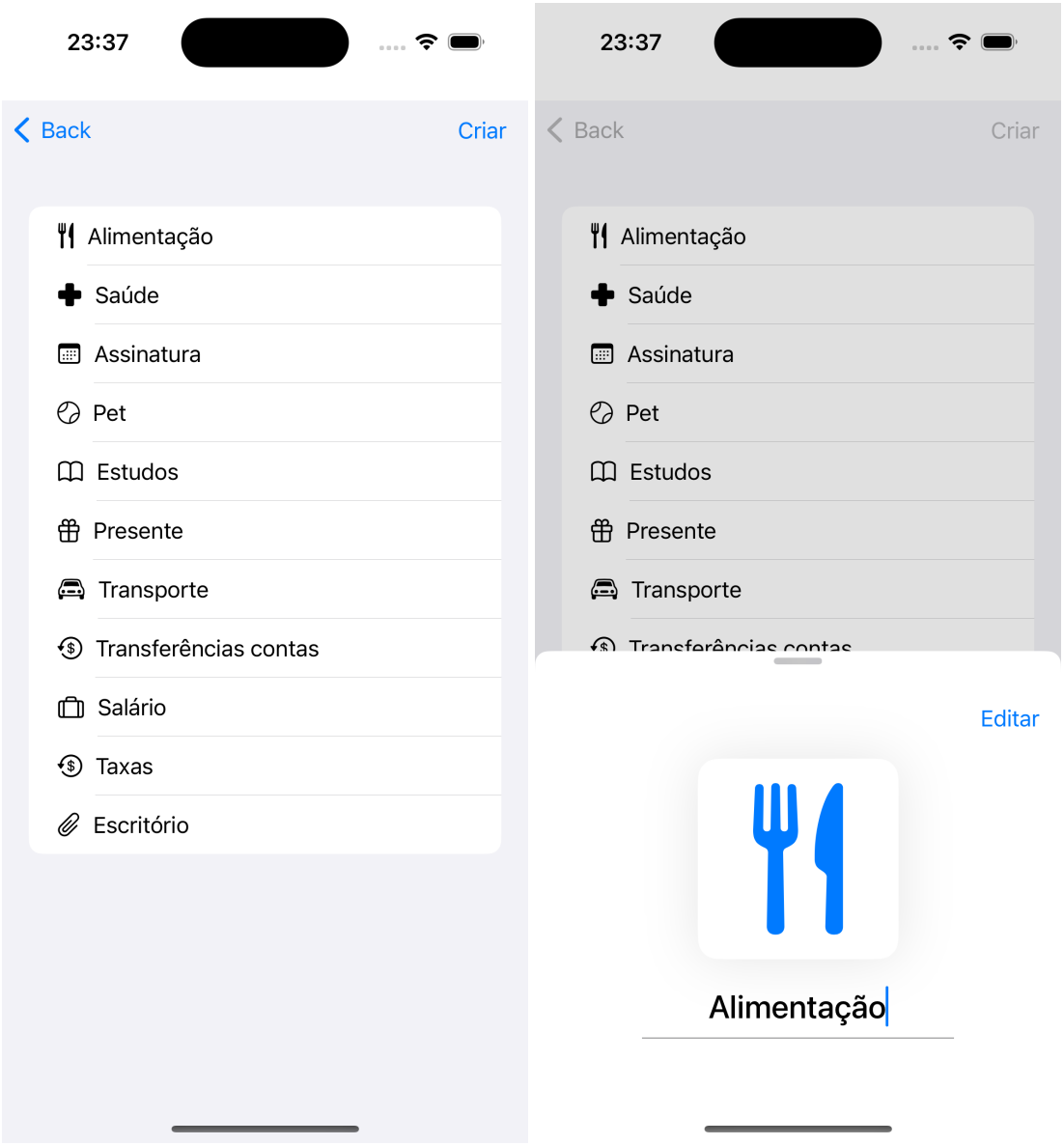
Fonte: O Autor (2024).

o método "*DELETE*".

Além de excluir, é possível criar novas categorias usando o botão "Criar" do cabeçalho ou editar, clicando em uma das categorias da lista. Tanto na edição quanto criação é aberta uma "*sheet*"[5] do *SwiftUI* com o ícone e nome da categoria - quando se trata de uma criação mostra o ícone padrão de cifrão e o nome em branco. Quando uma categoria é criada, é marcada com o "bool_adicionado" verdadeiro e enviada para o *endpoint* "*/category*" com o método "*POST*", já quando é alterada, é marcada com o "bool_alterado" e é enviada para o mesmo *endpoint*, porém com o método "*PUT*".

[5] <https://developer.apple.com/documentation/swiftui/view/sheet(ispresented:ondismiss:content:)>

Figura 19 – Desenvolvimento: gerenciamento de categorias



(a) Listagem de categorias (b) Edição de categoria

Fonte: O Autor (2024).

7 TESTES

Os testes foram divididos entre a validação da integração de dados (Seção 7.1), ou seja, se os dados das instituições financeiras estão sendo extraídos e carregados corretamente para o banco de dados em nuvem da solução; e os testes do protótipo de aplicação (Seção 7.2), que valida se os dados estão sendo mostrados corretamente no protótipo de visualização de dados.

7.1 VALIDAÇÃO DA INTEGRAÇÃO DE DADOS

Os testes desta seção abrangem a parte da solução responsável por executar o *pipeline* de integração de dados e enviar os dados para o protótipo de aplicação. Para isso, foi separado entre: testes de autenticação, que validam se somente usuários autorizados estão com acesso aos dados da solução (Seção 7.1.1); e a validação de dados, com objetivo de comparar o que foi extraído da *Pluggy* com os dados apresentados pelas instituições financeiras em extratos bancárias e em seus aplicativos individuais (Seção 7.1.2).

7.1.1 Autenticação

Para validar se a autenticação está funcionando corretamente, foram realizadas quatro requisições na API da solução:

1. Com usuário válido dentro da organização do *Entra ID*;
2. Sem nenhum tipo de autenticação definida.
3. Usando um *token* inválido;
4. Usuário logado porém com conta de fora da organização;

Todas requisições realizadas foram a partir da plataforma *Postman*[1], no *endpoint* "*/category*" utilizando o método "*GET*" sem nenhum corpo de requisição, somente os cabeçalhos padrão do *Postman* para o tipo de autenticação. A geração do *token* de todos os testes (que utilizaram *token* gerado) foram realizadas pelo mesmo endereço, *ID* de organização, cliente, segredo e escopo.

Entre os testes realizados, somente o caso número um teve sucesso para obter a resposta da requisição. O teste número dois, assim como o teste de número três, retornaram código de resposta 401 acompanhado por uma mensagem de "Token inválido". Enquanto o teste número quatro retornou erro ainda ao tentar realizar o *login* através do *link* de autenticação na organização. As evidências dos testes realizados podem ser vistos no Apêndice H

[1] <https://www.postman.com>

7.1.2 Validação de dados

Foram realizados testes individuais em cada uma das instituições financeiras, comparando a fatura de um mês, gerada através do aplicativo da instituição com os dados que foram carregados para o banco de dados. Esta comparação foi realizada entre os campos: descrição, valor e data do lançamento. Entre os testes realizados, os lançamentos apresentados no banco de dados fecharam com os lançamentos da fatura das três contas - XP, *Nubank* e Bradesco.

Um exemplo de como foi realizada a comparação para a conta corrente *Nubank*, no mês de outubro de 2024, pode ser visto na Figura 20. A Figura 20a mostra uma consulta no banco de dados, selecionando todas as transações da conta com nome "Nubank - conta" no mês de outubro de 2024. Enquanto a Figura 20b mostra o extrato da conta vindo diretamente do aplicativo do *Nubank*.

É possível identificar que a quantidade de lançamentos é a mesma nos dois casos, com datas e valores iguais. Já as descrições, apesar de não serem exatamente iguais ao extrato, ainda contêm a mesma informação, sendo o suficiente para identificar a transação. A diferença nas descrições acontece porque o *Nubank* não disponibiliza a descrição exatamente igual ao extrato para o *Open Finance*. As outras contas bancárias obtiveram resultados parecidos, algumas com diferenças na descrição, mas com os valores, datas e sentido da descrição sempre iguais ao extrato.

Todos os testes que envolvem realizar consultas no banco de dados foram realizados através da mesma ferramenta de consultas de bancos de dados, o "*TablePlus*"[2]. A conexão com o banco de dados em nuvem foi realizada através da *string* de conexão usada pela API.

7.2 VALIDAÇÃO DO PROTÓTIPO

A validação do protótipo se deu por dois testes principais: o teste de dados, para validar se os dados apresentados pelo protótipo estão de acordo com os dados extraídos pelo *pipeline* de integração; e o teste das modificações de informações controladas pelo protótipo. Para validar se as informações alteradas pelo protótipo de aplicação estão sendo refletidos corretamente para a nuvem foram realizados diversos testes categorizando transações, alterando o atributo de "ignorar" e realizando todas operações de categoria, criações, edições e exclusões.

Para validar se as telas do protótipo estão mostrando os dados corretamente, de acordo com o apresentado pelas instituições financeiras, foram realizados quatro testes:

1. **Somatório de gastos e receitas:** validar se os valores apresentados nos dois primeiros cartões do relatório estão de acordo com os valores do banco de dados.

[2] <https://tableplus.com>

Figura 20 – Validação de dados: *Nubank* - conta corrente

(a) Validação de dados: banco de dados em nuvem

```
SELECT
     ft.dt_transacao
    ,ft.ds_transacao
    ,ft.vl_transacao
    ,dc.ds_conta
FROM transactions.f_transacao AS ft
LEFT JOIN transactions.d_conta AS dc
    ON ft.cod_conta = dc.cod_conta
WHERE
        YEAR(ft.dt_transacao) = 2024
    AND MONTH(ft.dt_transacao) = 10
    AND dc.ds_conta LIKE 'Nubank - conta'
ORDER BY ft.dt_transacao ASC
```

line 15, column 1, location 321

	dt_transacao	ds_transacao	vl_transacao	ds_conta
1	2024-10-01 19:57:23.163	Transferência Recebida|LUCAS MARTINS DE BARROS	1280.00	Nubank - conta
2	2024-10-01 19:57:51.167	Pagamento efetuado	-1281.85	Nubank - conta
3	2024-10-03 15:40:54.920	Transferência enviada|Douglas Lucietto	-50.00	Nubank - conta
4	2024-10-14 12:20:32.007	Transferência Recebida|FERNANDA DE SOUZA LOPES	40.00	Nubank - conta
5	2024-10-17 21:44:25.090	Transferência enviada|Douglas Lucietto	-90.00	Nubank - conta
6	2024-10-22 22:04:48.047	Transferência enviada|Gracilene Moreira de Barros	-50.00	Nubank - conta
7	2024-10-25 13:21:34.027	Pagamento de fatura	-50.89	Nubank - conta
8	2024-10-31 19:14:53.557	Transferência enviada|Douglas Lucietto	-50.00	Nubank - conta

(b) Validação de dados: extrato

NU_437153165_01OUT2024_31OUT2024

Data	Valor	Identificador	Descrição
01/10/2024	1280.00	66fc7e53-e782-41e1-9afd-b7c44fa01903	Transferência recebida pelo Pix - LUCAS MARTINS DE BARROS - ···.640.880-·· - BCO BRADESCO S.A. (0237) Agência: 3269 Conta: 69772-9
01/10/2024	-1281.85	66fc7e6f-3ce9-410d-acb5-df5a1c512885	Pagamento de boleto efetuado - FUNDACAO UNIVERSIDADE DE CAXIAS DO SUL
03/10/2024	-50.00	66fee536-6c7f-4b7b-ba0e-153c43f33c68	Transferência enviada pelo Pix - Douglas Lucietto - ···.990.220-·· - NU PAGAMENTOS - IP (0260) Agência: 1 Conta: 95232620-3
14/10/2024	40.00	670d0c8f-03e3-4f6e-b7c6-c54ccb1dd5a8	Transferência recebida pelo Pix - FERNANDA DE SOUZA LOPES - ···.660.570-·· - COOP SICREDI SERRANA RS Agência: 167 Conta: 88046-5
17/10/2024	-90.00	67118539-2021-4356-b7eb-edc92dbbe06c	Transferência enviada pelo Pix - Douglas Lucietto - ···.990.220-·· - NU PAGAMENTOS - IP (0260) Agência: 1 Conta: 95232620-3
22/10/2024	-50.00	67182180-a33a-43c7-80d4-e1761046bfe8	Transferência enviada pelo Pix - Gracilene Moreira de Barros - ···.849.810-·· - BCO ITAUCARD S.A. Agência: 500 Conta: 24587630-0
25/10/2024	-50.89	671b9b5d-3fc6-4bcf-befd-f87d2c67d62b	Pagamento de fatura
31/10/2024	-50.00	6723d72d-3484-45d4-8da6-639b7cbab5ac	Transferência enviada pelo Pix - Douglas Lucietto - ···.990.220-·· - NU PAGAMENTOS - IP (0260) Agência: 1 Conta: 95232620-3

Fonte: O Autor (2024).

2. **Transações não categorizadas:** validar a quantidade de transações novas, quais são e qual é a primeira transação nova.

3. **Gastos por categoria:** validar se os valores gastos por categoria estão de acordo com os valores no banco de dados.

4. **Transações recentes:** se as cinco transações recentes do relatório da aplicação estão apresentando a descrição, data, valor, nome da categoria e ícone corretamente.

Os testes foram realizados nas telas dos meses de novembro e outubro de 2024, comparando os valores mostrados nas telas do protótipo com o banco de dados em nuvem, ou seja, validando também possíveis problemas na inserção ou consulta do banco de dados local e nos retornos dos *endpoints* da API. Os resultados das consultas no banco de dados e a tela de relatório do protótipo foram exatamente iguais, mostrando que:

- A API da solução está enviando os dados corretamente através dos *endpoints*;

- O protótipo de aplicação está salvando os dados recebidos da API em seu banco de dados local de forma correta;
- As transformações realizadas pelo protótipo de aplicação durante a exibição (regra de separação por mês e formatações de datas para textos) não fizeram com que os dados divergissem.

As evidências dos testes, assim como as consultas realizadas no banco de dados em nuvem, são apresentadas no Apêndice I.

8 CONSIDERAÇÕES FINAIS

Este trabalho visava resolver o problema da integração de dados financeiros para facilitar o controle financeiro de um indivíduo. Para o desenvolvimento da solução, se fez necessário fundamentar conceitos que envolvem o tema de integração de dados, como a engenharia de dados e o *Open Finance* como um padrão de dados financeiros. A partir da fundamentação teórica, foi proposta uma solução, definindo as tecnologias, a arquitetura a ser utilizada e estabelecendo os requisitos funcionais e interfaces necessárias para implementação.

Com a solução proposta, iniciou-se a etapa de desenvolvimento, primeiramente realizando a integração dos dados financeiros, que incluiu a configuração da plataforma fornecedora dos dados, construção de um *pipeline* de ETL, configuração de um ambiente em nuvem e desenvolvimento de uma API para comunicação com o protótipo de aplicação. Em seguida, foi desenvolvido o protótipo de aplicação móvel, abordando desde a estruturação e gerenciamento dos dados até a implementação das telas e funcionalidades principais. Por fim, foram realizados testes para validar os dados extraídos, o funcionamento da integração e a eficácia do protótipo desenvolvido.

8.1 CONCLUSÃO

Com base nos estudos realizados e solução desenvolvida ao longo deste trabalho, pode-se concluir que é viável integrar os dados financeiros de uma pessoa mesmo que estejam distribuídos em mais de uma instituição. Porém, mesmo fazendo uso de um padrão definido para homogenizar o acesso aos dados das diferentes instituições financeiras no país, existem dificuldades causadas pelas diferenças nas implementações do padrão pelas diferentes instituições.

O *Open Finance* é um grande facilitador para a integração dos dados, tanto definindo uma forma de obter os dados diretamente das instituições, quanto definindo um padrão entre as instituições financeiras, reduzindo a quantidade de transformações necessárias para integração das informações. Além disso, a plataforma *Pluggy* foi essencial para o desenvolvimento por viabilizar o acesso aos dados do *Open Finance*.

Apesar dos padrões impostos pelo *Open Finance*, observou-se que a padronização entre as instituições é limitada aos nomes dos campos e seus tipos. Os campos de texto livre, como a descrição da transação (campo "*/data/transactionName*" no *Open Finance*, equivalente ao "ds_transacao" da solução), não tem seu conteúdo padronizado. Isso fez com que se tornasse necessário, por exemplo, realizar uma transformação para padronizar a descrição das transações de um dos bancos.

Outra dificuldade durante a integração dos dados foi com o *Id* das transações, vindo da

Pluggy, que pode mudar de acordo com as mudanças de outros campos, como a descrição. Esse problema foi contornado realizando um processo de *merge* personalizado que se assemelha à uma carga completa. Porém, seria possível resolver o problema de outras formas, por exemplo, recebendo um gatilho da própria *Pluggy* quando uma transação é excluída para replicar a exclusão no banco de dados da solução.

8.2 TRABALHOS FUTUROS

Como sugestões de melhoria e continuidade do trabalho, é possível:

- **Tornar a solução multiusuário:** para agregar essa funcionalidade não seria possível utilizar a versão grátis da *Pluggy* (Meu *Pluggy*), seria necessário ter um fluxo de autenticação na *Pluggy* através do protótipo, adicionar uma tabela para controle de usuários no banco de dados e realizar o controle das requisições da API para retornar somente dados do usuário que está realizando a requisição.

- **Realizar carga incremental no banco de dados local do protótipo:** esta funcionalidade iria reduzir a necessidade de processamento tanto do protótipo de aplicação quanto da API. Para realizar a carga incremental seria necessário adicionar um campo de data de alteração no banco de dados em nuvem, assim como registrar um histórico das transações e categorias deletadas. O *endpoint* "*/data*" poderia ser alterado para receber uma data de última atualização, que retorne somente os dados atualizados (e excluídos) após a última atualização do protótipo.

- **Carga incremental do *pipeline* de integração:** seria necessário adicionar um *endpoint* para receber as transações excluídas da *Pluggy* (através de um *webhook*[1]), possibilitando a realização de extrações incrementais. Além de excluir as transações necessárias ainda seria necessário realizar a busca por similaridade para não perder as informações de categoria e "ignorar", controladas pelo protótipo.

- **Adicionar outras informações do *Open Finance*:** através da *Pluggy* e do *Open Finance* é possível adicionar outras informações das contas bancárias, como por exemplo saldos e informações de investimentos.

- **Desacoplamento do fornecedor de dados:** da forma que o *pipeline* de ETL foi desenvolvido existe um acoplamento direto com a *Pluggy*. Para facilitar a troca de fornecedor ou aceitar mais de um fornecedor para contas diferentes poderia ser aplicado o padrão de Injeção de Dependência, com auxílio de uma interface abstrata que defina operações esperadas de qualquer fornecedor (GAMMA *et al.*, 2000).

[1] <https://docs.pluggy.ai/docs/webhooks>

APÊNDICE A – EXEMPLO *ENDPOINT ACCOUNTS*

O Código 1 representa o retorno do *endpoint* "/accounts" da *Pluggy*, passando o *ItemId* da conexão com a "XP Banking" - "3ee6505a-4029-4b7a-9f4f-905175d8ac4f". Os itens "taxNumber" e "transferNumber" do exemplo de retorno foram censurados.

Importante ressaltar que, para o mesmo *itemId* foram recuperados dois *Id's* diferentes, representando os dois produtos vinculados à esse *itemId*, um do tipo "*BANK*" (conta corrente) e o outro do tipo "*CREDIT*" (cartão de crédito). Estes dois Id's serão usados para obter as transações, que são vinculadas especificamente aos produtos (como descrito na Seção 5.4.1). Os dois outros *Item's* usados no desenvolvimento ("Nubank" e "Bradesco") também contêm um produto de conta corrente e outro de cartão de crédito, assim como o "XP".

Código 1 – Retorno do *endpoint accounts* da *Pluggy*


```
{
    "total": 2,
    "totalPages": 1,
    "page": 1,
    "results": [
        {
            "id": "57dc15f3-0e55-44d8-a88c-f3cccb0eca0c",
            "type": "BANK",
            "subtype": "CHECKING_ACCOUNT",
            "name": "Banco XP S.A.",
            "balance": 2635.06,
            "currencyCode": "BRL",
            "itemId": "3ee6505a-4029-4b7a-9f4f-905175d8ac4f",
            "number": "00710485-8",
            "createdAt": "2024-09-13T20:08:59.775Z",
            "updatedAt": "2024-10-12T20:39:18.122Z",
            "marketingName": null,
            "taxNumber": "***.***.***-**",
            "owner": "LUCAS MARTINS DE BARROS",
            "bankData": {
                "transferNumber": "***/****/*********-*",
                "closingBalance": 2635.06,
                "automaticallyInvestedBalance": 2635.06,
                "overdraftContractedLimit": 0,
                "overdraftUsedLimit": 0,
                "unarrangedOverdraftAmount": 0
            },
            "creditData": null
        },
        {
```

```
                    "id": "240b5fb8-e25e-443b-b562-adcfa13fdeaf",
                    "type": "CREDIT",
                    "subtype": "CREDIT_CARD",
                    "name": "Cartao XP Visa Infinite",
                    "balance": 1548.76,
                    "currencyCode": "BRL",
                    "itemId": "3ee6505a-4029-4b7a-9f4f-905175d8ac4f",
                    "number": "8470",
                    "createdAt": "2024-09-13T20:09:00.508Z",
                    "updatedAt": "2024-10-12T20:39:18.272Z",
                    "marketingName": null,
                    "taxNumber": null,
                    "owner": "LUCAS MARTINS DE BARROS",
                    "bankData": null,
                    "creditData": {
                        "level": "INFINITE",
                        "brand": "VISA",
                        "balanceCloseDate": null,
                        "balanceDueDate": "2024-10-01",
                        "availableCreditLimit": 3451.24,
                        "balanceForeignCurrency": null,
                        "minimumPayment": 485.41,
                        "creditLimit": 5000,
                        "isLimitFlexible": false,
                        "holderType": null,
                        "status": null
                    }
                }
            ]
        }
```


Fonte: O Autor (2024)

APÊNDICE B – EXEMPLO DE UM SEGREDO DE CONTA

O Código 2 mostra os Id's contidos no segredo do cofre de chaves da conta da XP. O "contaId" é o *Id* do produto "BANK" retornado pelo *endpoint* "/accounts" (conforme representado pelo Apêndice A), enquanto o "creditoId" é o *Id* do produto "CREDIT", do mesmo *endpoint*.

Os três segredos foram criados manualmente no cofre de chaves de conta, contendo o nome da instituição financeira como título e um JSON de Id's, que foram obtidos através do *endpoint* "*accounts*" como conteúdo. Esse segredo será usado futuramente para extrair as transações de cada uma das contas, como será explicado na Seção 5.4.1.

Todos os segredos salvos nos cofres de chaves foram em texto livre. A Microsoft (2024), cita em sua documentação que camadas extras de proteção, como a criptografia por uma chave de proteção separada antes do armazenamento do segredo é recomendada somente para dados altamente confidenciais, o que não é o caso das informações deste trabalho.

Código 2 – Exemplo de segredo XP do cofre de chaves de conta

```
1  {
2      "itemId": "8acef5f6-6fa4-425d-b483-efb144abd739",
3      "contaId": "caa33696-ba3e-4f68-ac26-75769d34b000",
4      "creditoId": "0918080e-eb39-4acd-b382-d9618b6f93a2"
5  }
```

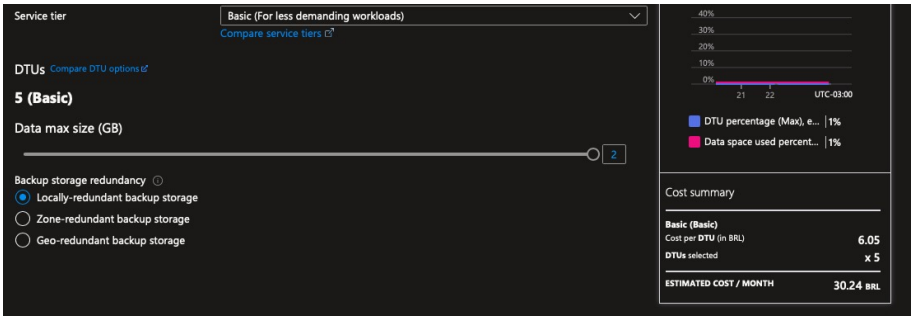
APÊNDICE C – CUSTOS DA INFRAESTRUTURA EM NUVEM

A *Azure* oferece 100 dólares de crédito para o uso em seus produtos através do plano "*Microsoft Azure for Students*"[1], que foi utilizado para manter os serviços deste trabalho. O banco de dados utilizado foi o "Básico" dos modelos de compra baseados em Unidade de Transação do Banco de Dados (DTU), que tem um custo aproximado de 30 reais mensais, conforme apresentado na Figura 21.

Enquanto o plano do serviço web utilizado inicialmente foi o "*Free* F1", que não tem nenhum custo, porém, com uma limitação de 60 minutos de processamento por dia. Durante o desenvolvimento, os 60 minutos de processamento diários não foram o suficiente, sendo necessário alterar para o plano "*Basic* B1", que não possui limite de tempo. A Figura 22 mostra o plano *Basic*, que tem um custo mensal de aproximadamente 74 reais, porém, como a aplicação não recebe solicitações frequentemente, o custo foi mais baixo.

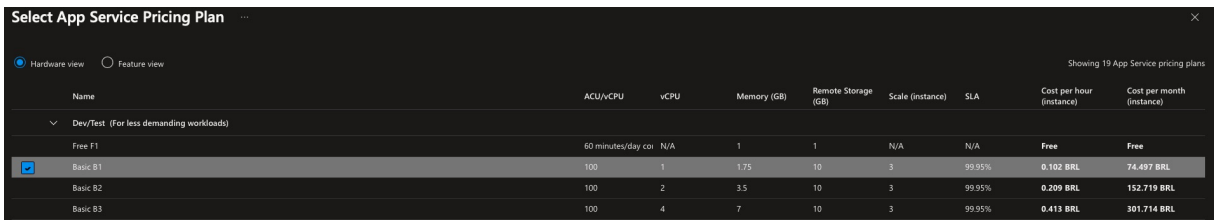
Os custos totais, durante todo o desenvolvimento, foram de 24 dólares para o banco de dados, 35 dólares para a aplicação web e 28 centavos de dólar para o cofre de chaves - que tem um custo por operação (consulta, edição ou criação de segredos). A contratação dos serviços foi entre os meses de março e novembro, totalizando 65 dos 100 dólares disponíveis, uma média de 7 dólares por mês.

Figura 21 – Plano de banco de dados



Fonte: O Autor (2024).

Figura 22 – Plano de aplicação web



Fonte: O Autor (2024).

[1] <https://azure.microsoft.com/pt-br/free/students>

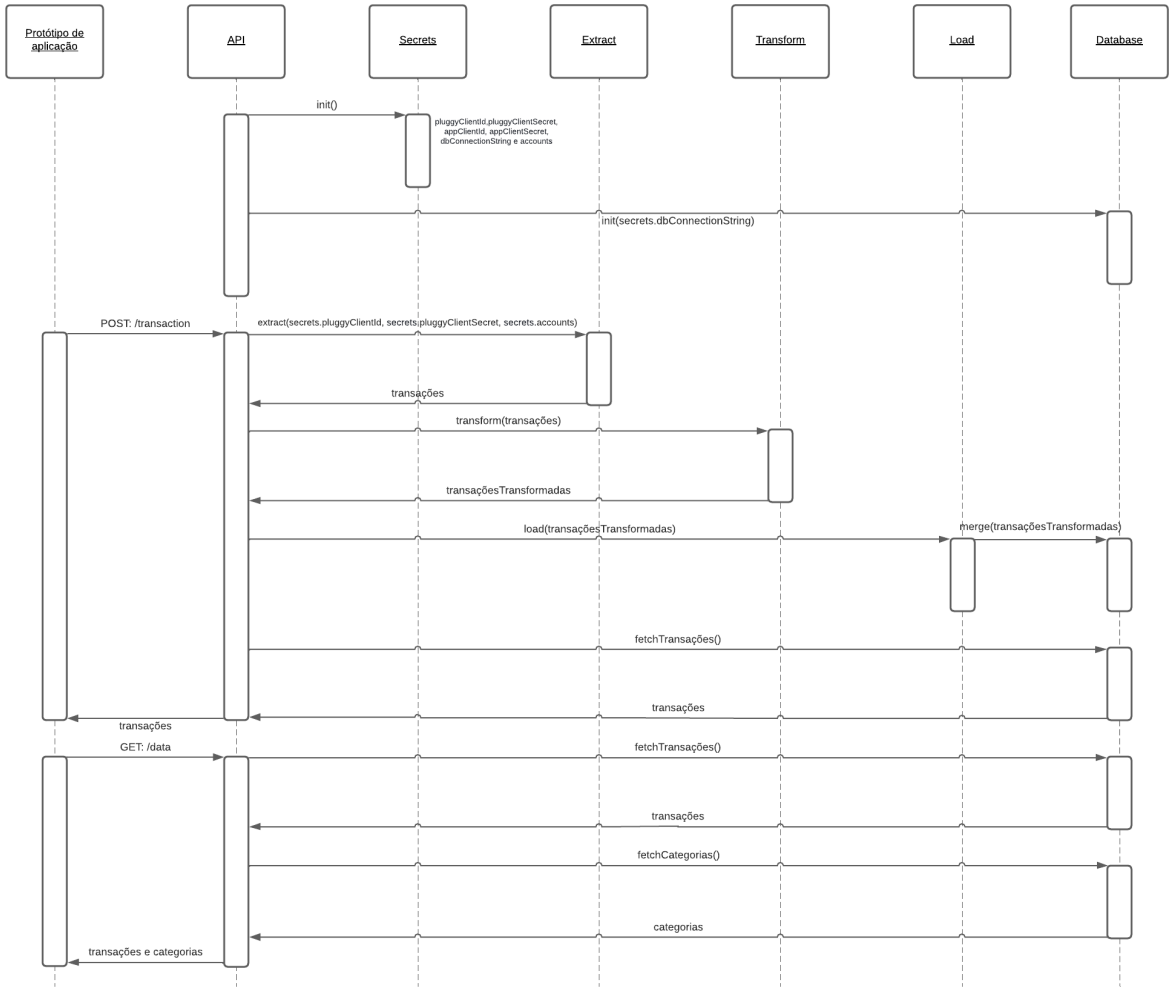
APÊNDICE D – DIAGRAMA DE SEQUÊNCIA DA API

A Figura 23 mostra um diagrama de sequência de uma requisição que aciona o *pipeline* de ETL e da requisição usada para obter todas informações necessárias (transações e categorias) para atualizar o banco de dados local do protótipo. O diagrama mostra uma representação do protótipo de aplicação, que estará realizando as requisições, a própria API, que irá receber a requisição e processar quais funções precisa realizar e cinco classes, que serão chamadas pela API da solução.

Inicialmente, antes de receber qualquer requisição, a API realiza a inicialização dos objetos "*Secrets*" e "*Database*", que serão usados durante toda a "vida" da API, conforme explicado na Seção 5.3. Após a inicialização dos dois objetos, a API chama a função de extração da classe "Extract", passando como parâmetros os segredos necessários para extrair os dados da *Pluggy*. Com a listagem de todas as transações da extração, realiza a transformação, recebendo os dados transformados, que, posteriormente, são enviados para a função "*load*", que realiza o *merge* no banco de dados. Após a finalização do processo de ETL, a API realiza a busca de todas as transações disponíveis no banco de dados e retorna a lista para o protótipo.

O *endpoint* "*/data*" aciona somente o objeto da classe "*Database*", primeiramente fazendo a busca por todas transações e depois a busca pelas categorias. Esses dados são retornados para o protótipo de aplicação em formato de JSON, composto por uma chave de transações, com a lista de objetos de transações e uma chave para categorias, com uma lista de objetos categoria.

Figura 23 – Diagrama de sequência da API



Fonte: O Autor (2024).

APÊNDICE E – *ENDPOINTS* DA API DA SOLUÇÃO

Os *endpoints* disponíveis na API da solução, seus possíveis métodos de requisição, expectativa de corpo de requisição e de resposta estão representados na listagem a seguir:

1. ***/category***: gerencia todas as operações relacionadas a categorias, oferecendo os métodos:
    - ***GET***: recupera todas as categorias cadastradas.
        - Corpo da requisição: não necessário.
        - Resposta: lista de objetos, cada um contendo os campos “id”, “nm_categoria” e “ds_icone”, para representar cada categoria.
    - ***POST***: cria novas categorias.
        - Corpo da requisição: lista de objetos, cada um com os campos “id”, “nm_categoria” e “ds_icone”, representando cada categoria a ser criada.
        - Resposta: uma mensagem de confirmação com o total de categorias criadas.
    - ***PUT***: atualiza uma ou mais categorias existentes.
        - Corpo da requisição: uma lista de objetos com os campos “id”, “nm_categoria” e “ds_icone”, representando cada categoria que deve ser atualizada.
        - Resposta: uma mensagem de confirmação com o total de categorias atualizadas.
    - ***DELETE***: exclui uma ou mais categorias.
        - Corpo da requisição: uma lista de id’s das categorias que devem ser excluídas.
        - Resposta: uma mensagem de confirmação com o total de categorias excluídas.
2. ***/transaction***: gerencia todas as operações relacionadas a categorias, oferecendo os métodos:
    - ***GET***: recupera todas as transações do banco de dados.
        - Corpo da requisição: não necessário.
        - Resposta: lista de objetos, cada um contendo todos os campos de transações.
    - ***POST***: aciona o gatilho para realizar o processo de ETL, inserindo as novas transações.
        - Corpo da requisição: não necessário.
        - Resposta: lista de objetos de todas as transações disponíveis, assim como o método *GET*.
    - ***PUT***: altera os atributos gerenciados pelo protótipo de aplicação de uma ou mais transações existentes.

  - Corpo da requisição: uma lista de objetos com os campos “id”, “bool_novo”, “id_categoria” e “dt_alteracao”, representando as possíveis alterações de uma transação.
  - Resposta: uma mensagem de confirmação com o total de transações alteradas.

3. ***/data***: retorna todos os dados necessários para o protótipo de aplicação atualizar o banco de dados local. Este *endpoint* aceita somente requisições de *GET*:

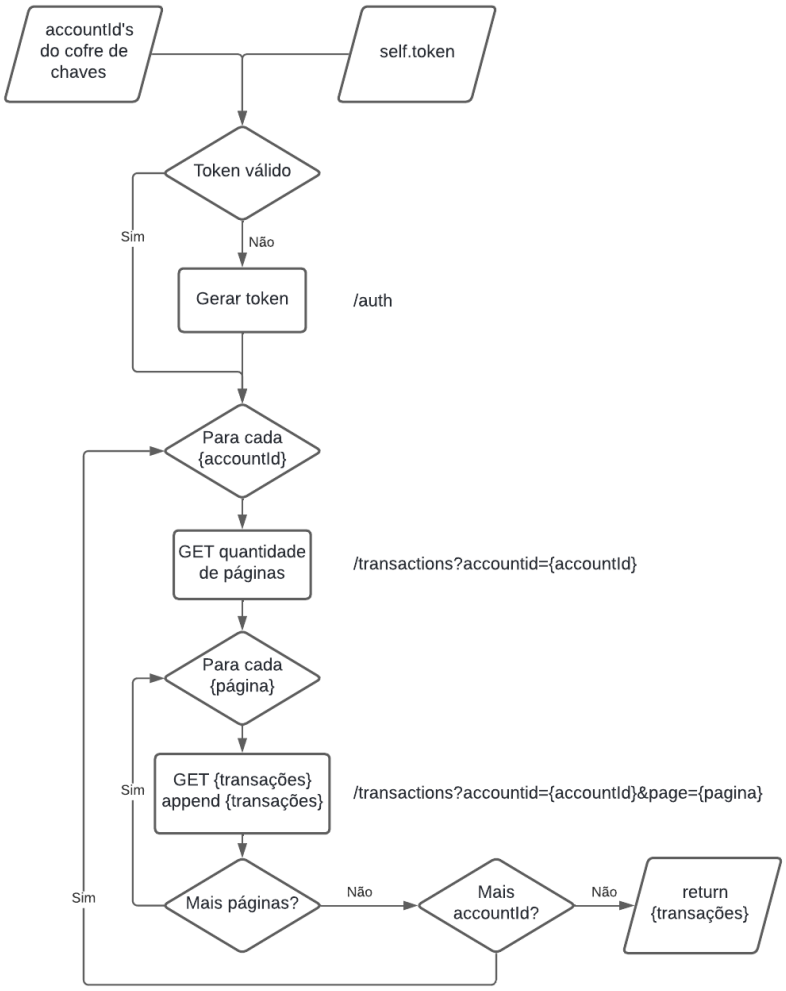
   - ***GET***: recupera a lista de todas as transações e categorias, ou seja, todas as informações necessárias para atualizar o banco de dados local do protótipo de aplicação.
     - Corpo da requisição: não necessário.
     - Resposta: dois objetos, “categories” e “transactions”, que contêm uma lista de objetos representando todas as categorias cadastradas ou todas as transações disponíveis, respectivamente.

APÊNDICE F – FLUXOGRAMA DE EXTRAÇÃO DE DADOS

A Figura 24 mostra o fluxo de alto nível do método da classe de extração responsável por extrair as transações da *Pluggy* de todos *accountId's* do cofre de chaves de contas. No fluxo, é recebido como entrada uma relação de todos *accountId's* disponíveis no cofre de chaves, além do *token*, que é atributo da classe de extração.

Primeiramente, é realizada uma validação do *token*, que, caso não seja válido, é gerado um novo utilizando o *clientId* e *clientSecret* da *Pluggy*, através do *endpoint* "/auth". Com um token válido, é realizado um laço, que, para cada conta, obtém a quantidade de páginas de transações, através do *endpoint* "/transactions", passando como cabeçalho da requisição o accountId em questão. Com o número de páginas, é realizado outro laço, que percorre cada uma das páginas do mesmo *endpoint* ("/*transactions*"), agregando todas as transações da página em uma lista de dicionários, que é retornada no final dos dois laços.

Figura 24 – Fluxograma de extração de dados



Fonte: O Autor (2024).

APÊNDICE G – EXEMPLO DE *ENDPOINT TRANSACTIONS*

O Código 3 mostra como exemplo uma transação de uma conta do tipo "*BANK*" retornada pelo *endpoint* "/transactions" da *Pluggy*. A transação se trata de uma transferência *Pix* recebida na conta da "XP", no valor de R$300,26. Enquanto o Código 4 mostra como exemplo uma transação de uma conta do tipo "*CREDIT*" do mesmo *endpoint*. Os itens "*documentNumber*", tanto de "payer" quanto "receiver" do retorno foram censurados.

Código 3 – Exemplo de transação de conta tipo *BANK*


```
{
    "id": "c59279ea-9c18-46db-b3e0-d1c3266cb56c",
    "description": "Pix recebido de Lucas Martins de Barros",
    "descriptionRaw": "Pix recebido de Lucas Martins de Barros",
    "currencyCode": "BRL",
    "amount": 300.26,
    "amountInAccountCurrency": null,
    "date": "2024-10-03T02:51:39.427Z",
    "category": "Same person transfer",
    "categoryId": "04000000",
    "balance": null,
    "accountId": "57dc15f3-0e55-44d8-a88c-f3cccb0eca0c",
    "providerCode": null,
    "status": "POSTED",
    "paymentData": {
      "payer": {
        "accountNumber": "00069772-9",
        "branchNumber": "3269",
        "documentNumber": {
          "type": "CPF",
          "value": "***.***.***-**"
        },
        "name": null,
        "routingNumber": null,
        "routingNumberISPB": null
      },
      "paymentMethod": "PIX",
      "reason": null,
      "receiver": {
        "accountNumber": null,
        "branchNumber": null,
        "documentNumber": {
          "type": "CPF",
          "value": "***.***.***-**"
        },
```

```
            "name": null,
            "routingNumber": null,
            "routingNumberISPB": null
          },
          "receiverReferenceId": null,
          "referenceNumber": null
        },
        "type": "CREDIT",
        "operationType": "PIX",
        "creditCardMetadata": null,
        "acquirerData": null,
        "createdAt": "2024-10-03T20:29:18.029Z",
        "updatedAt": "2024-10-03T20:29:18.029Z"
    }
```


Fonte: O Autor (2024)

Código 4 – Exemplo de transação de conta tipo *CREDIT*

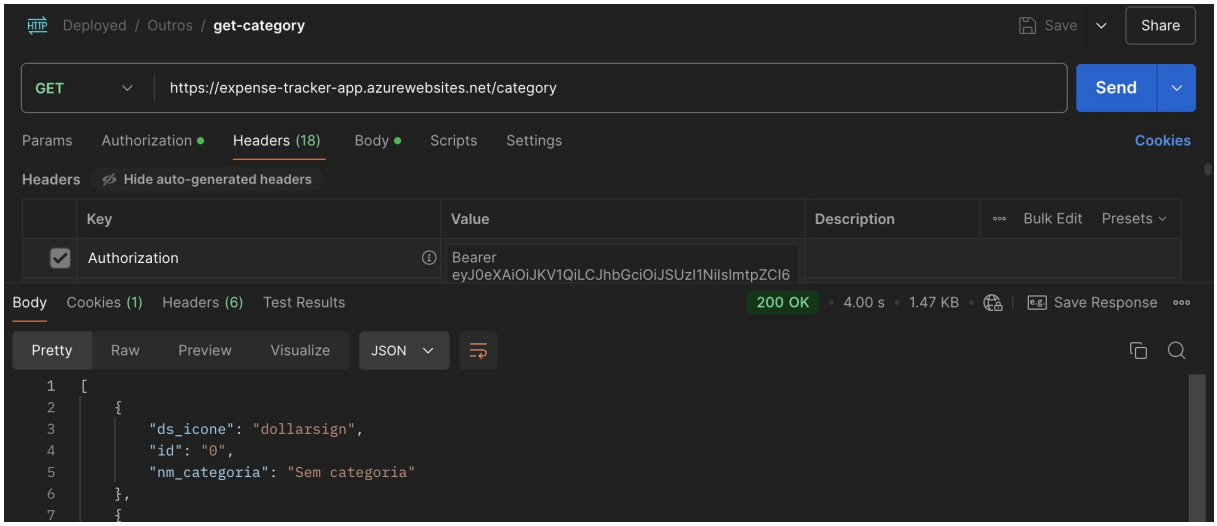

```
{
    "id": "41fbd40d-d0c8-4c6d-a25d-ecee70ac04ff",
    "description": "PAYPAL␣␣␣␣␣␣*MICROSOFT",
    "descriptionRaw": "PAYPAL␣␣␣␣␣␣*MICROSOFT",
    "currencyCode": "BRL",
    "amount": 35.99,
    "amountInAccountCurrency": null,
    "date": "2024-09-07T17:36:00.000Z",
    "category": "Services",
    "categoryId": "07000000",
    "balance": null,
    "accountId": "240b5fb8-e25e-443b-b562-adcfa13fdeaf",
    "providerCode": null,
    "status": "POSTED",
    "paymentData": null,
    "type": "DEBIT",
    "operationType": null,
    "creditCardMetadata": {
      "cardNumber": "8028",
      "payeeMCC": 7372,
      "billId": "a09e2bc4-7584-4fc7-bd35-2df9adef18fe"
    },
    "acquirerData": null,
    "merchant": null,
    "createdAt": "2024-09-13T20:09:00.836Z",
    "updatedAt": "2024-10-01T20:27:16.177Z"
}
```


Fonte: O Autor (2024)

APÊNDICE H – EVIDÊNCIAS DE TESTES DE AUTENTICAÇÃO

A Figura 25 mostra o resultado do teste número um de autenticação, utilizando um usuário da organização autenticado via "*Entra ID*". É possível visualizar que o código de resposta foi 200 e o corpo contém as categorias em formato de JSON, assim como esperado para o *endpoint*.

Figura 25 – Teste de autenticação: usuário válido



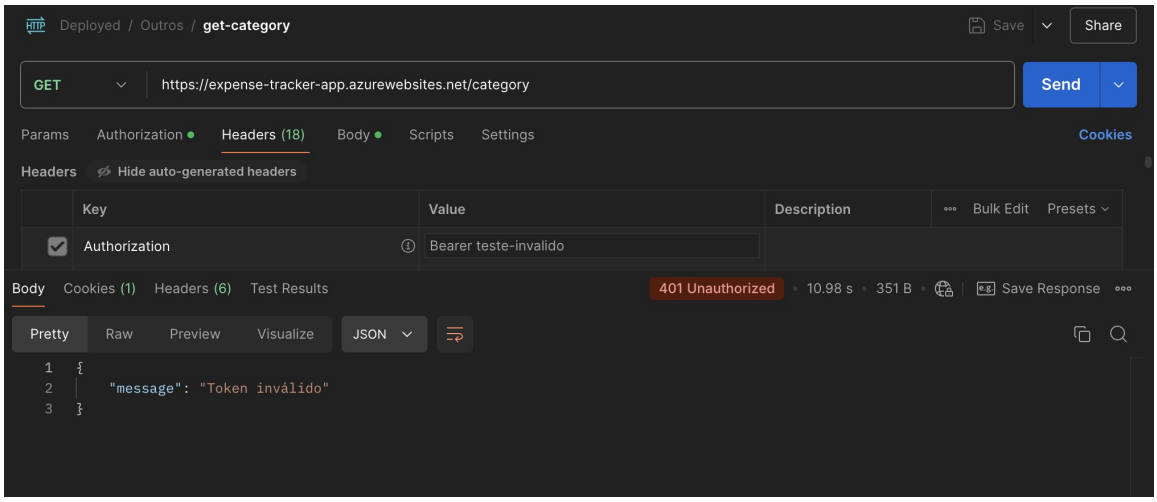
Fonte: O Autor (2024).

Enquanto isso, a Figura 26 mostra os testes dois (Figura 26a) e três (Figura 26b). O teste de número três contém "Bearer token-invalido" no cabeçalho de autenticação, onde deveria estar o *token*; e o teste de número quatro está sem o cabeçalho de autenticação (representado pelo tipo de autenticação selecionado como "Sem autenticação"). Ambos os testes retornaram erro de "*Token* inválido".

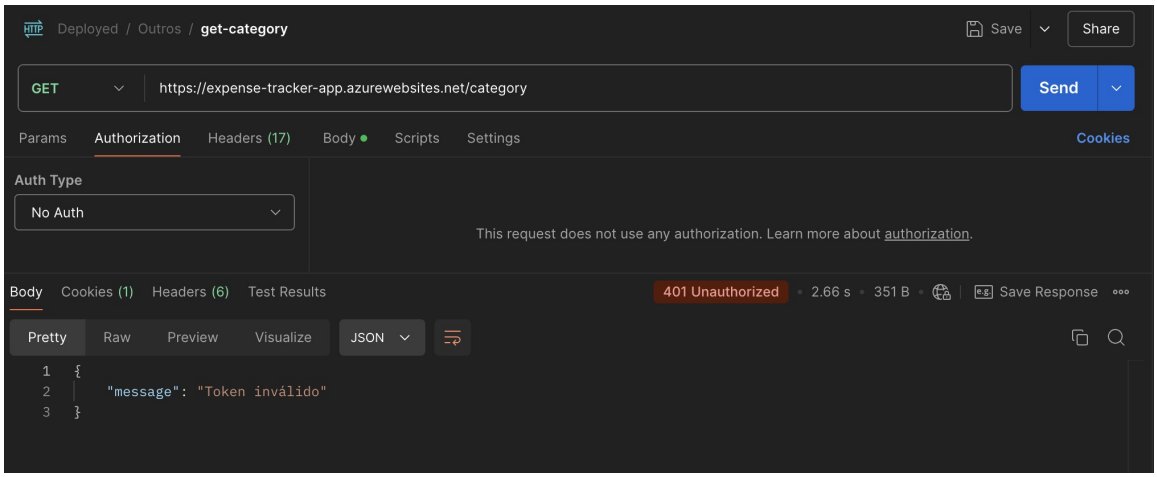
A Figura 27 mostra o erro ao tentar realizar o *login* no *link* da *Microsoft* para a organização utilizando um usuário que não pertence à organização da aplicação web. Com isso, não é gerado um *token* e o usuário não consegue acessar a API da solução.

Figura 26 – Testes de autenticação: *tokens* inválidos

(a) Teste de autenticação: *token* inválido

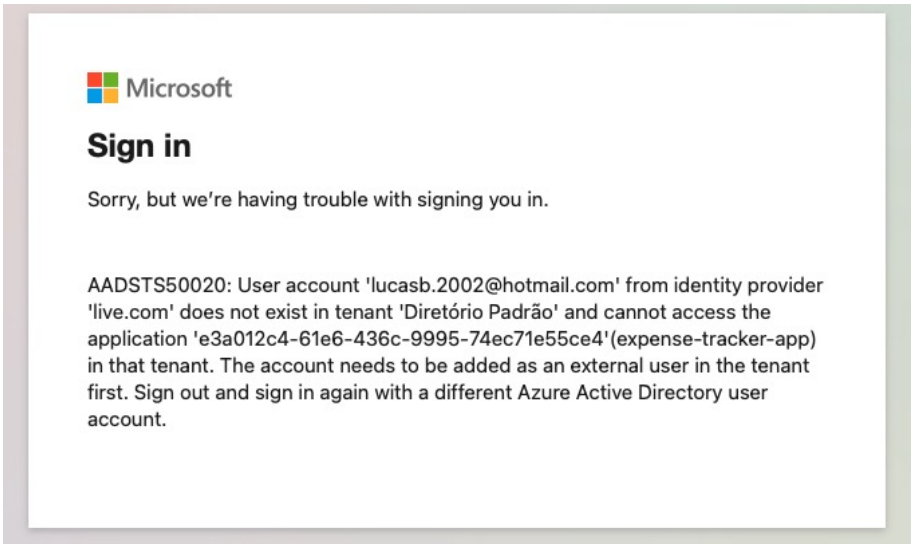


(b) Teste de autenticação: sem método de autenticação



Fonte: O Autor (2024).

Figura 27 – Teste de autenticação: usuário fora da organização



Fonte: O Autor (2024).

APÊNDICE I – EVIDÊNCIAS DE TESTES NAS TELAS DO PROTÓTIPO

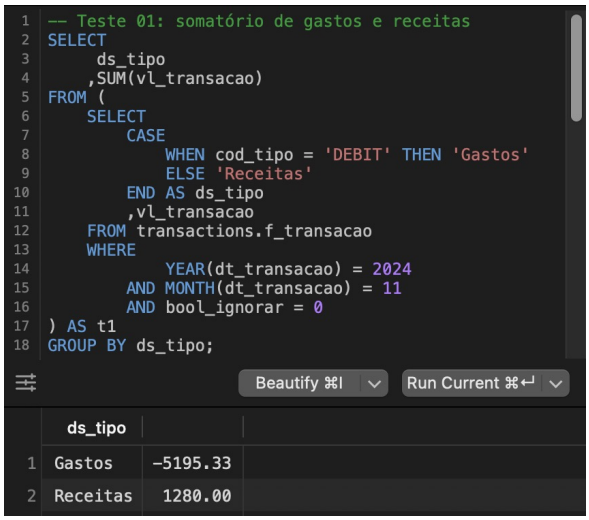
Para validar se o protótipo está mostrando os dados de forma esperada, como estão no banco de dados em nuvem foram realizados os quatro testes citados no Seção 7.2. A Figura 28 mostra a tela de novembro no momento dos testes, enquanto a Figura 29 mostra as consultas realizadas e seus resultados, que foram utilizados para comparação com a tela do protótipo.

A Figura 29a mostra a consulta para encontrar o valor de gastos e receitas não "ignorados" no mês de novembro - os resultados fecham com os valores mostrados no protótipo. A Figura 29c soma os valores de despesas não ignoradas no mês de novembro, agrupando por categoria - os valores arredondados são exatamente iguais ao cartão de gastos por categoria do protótipo de aplicação. A Figura 29b mostra a consulta realizada para obter as transações não categorizadas do mês, que retornou quatro transações, sendo a mais antiga com descrição de "TERR AEROPORTO POA", assim como mostrado no protótipo. O teste das transações recentes, da Figura 29d, mostra que as cinco transações mais recentes estão iguais tanto no banco de dados em nuvem quanto no protótipo, suas descrições, valores, categorias e datas.

Figura 28 – Teste do protótipo: relatório

Fonte: O Autor (2024).

Figura 29 – Testes do protótipo: banco de dados em nuvem

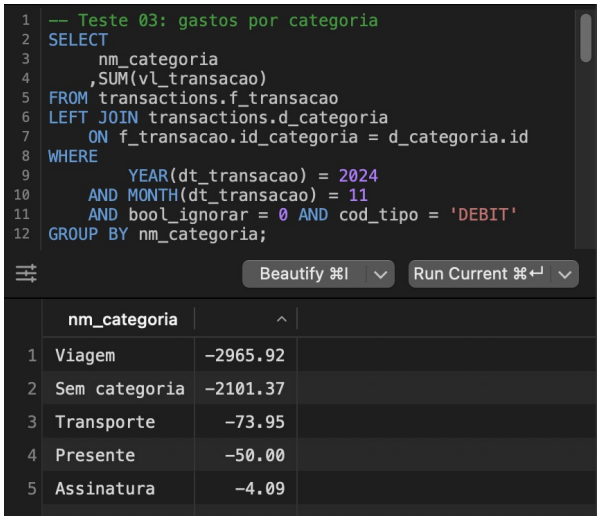


```sql
-- Teste 01: somatório de gastos e receitas
SELECT
     ds_tipo
    ,SUM(vl_transacao)
FROM (
    SELECT
        CASE
            WHEN cod_tipo = 'DEBIT' THEN 'Gastos'
            ELSE 'Receitas'
        END AS ds_tipo
        ,vl_transacao
    FROM transactions.f_transacao
    WHERE
            YEAR(dt_transacao) = 2024
        AND MONTH(dt_transacao) = 11
        AND bool_ignorar = 0
) AS t1
GROUP BY ds_tipo;
```

Beautify ⌘I | Run Current ⌘↵

	ds_tipo	
1	Gastos	-5195.33
2	Receitas	1280.00

(a) Somatório de gastos e receitas

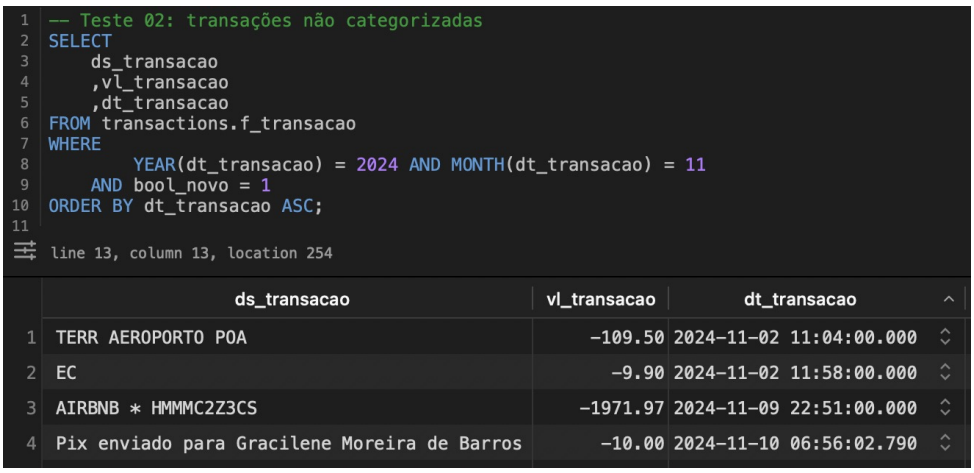


```sql
-- Teste 03: gastos por categoria
SELECT
     nm_categoria
    ,SUM(vl_transacao)
FROM transactions.f_transacao
LEFT JOIN transactions.d_categoria
    ON f_transacao.id_categoria = d_categoria.id
WHERE
        YEAR(dt_transacao) = 2024
    AND MONTH(dt_transacao) = 11
    AND bool_ignorar = 0 AND cod_tipo = 'DEBIT'
GROUP BY nm_categoria;
```

Beautify ⌘I | Run Current ⌘↵

	nm_categoria	
1	Viagem	-2965.92
2	Sem categoria	-2101.37
3	Transporte	-73.95
4	Presente	-50.00
5	Assinatura	-4.09

(b) Gastos por categoria

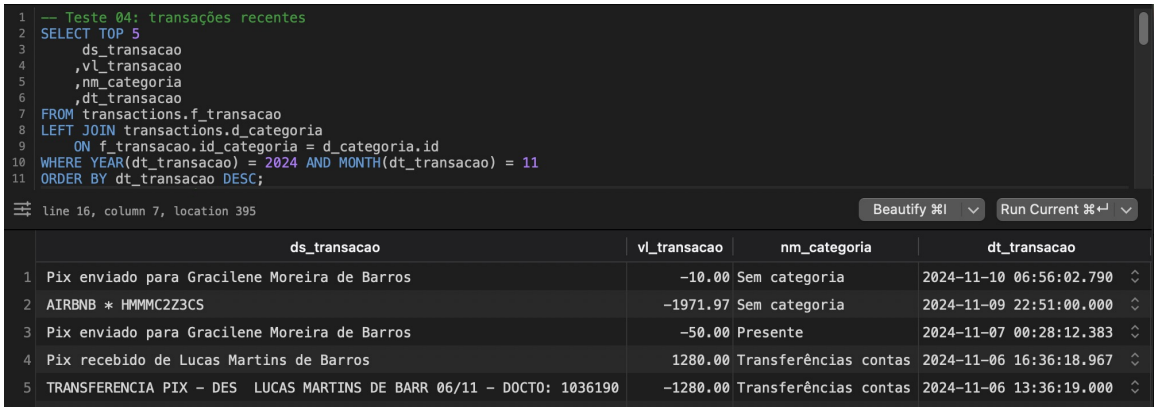


```sql
-- Teste 02: transações não categorizadas
SELECT
    ds_transacao
    ,vl_transacao
    ,dt_transacao
FROM transactions.f_transacao
WHERE
        YEAR(dt_transacao) = 2024 AND MONTH(dt_transacao) = 11
    AND bool_novo = 1
ORDER BY dt_transacao ASC;
```

line 13, column 13, location 254

	ds_transacao	vl_transacao	dt_transacao
1	TERR AEROPORTO POA	-109.50	2024-11-02 11:04:00.000
2	EC	-9.90	2024-11-02 11:58:00.000
3	AIRBNB * HMMMC2Z3CS	-1971.97	2024-11-09 22:51:00.000
4	Pix enviado para Gracilene Moreira de Barros	-10.00	2024-11-10 06:56:02.790

(c) Transações não categorizadas



```sql
-- Teste 04: transações recentes
SELECT TOP 5
     ds_transacao
    ,vl_transacao
    ,nm_categoria
    ,dt_transacao
FROM transactions.f_transacao
LEFT JOIN transactions.d_categoria
    ON f_transacao.id_categoria = d_categoria.id
WHERE YEAR(dt_transacao) = 2024 AND MONTH(dt_transacao) = 11
ORDER BY dt_transacao DESC;
```

line 16, column 7, location 395 | Beautify ⌘I | Run Current ⌘↵

	ds_transacao	vl_transacao	nm_categoria	dt_transacao
1	Pix enviado para Gracilene Moreira de Barros	-10.00	Sem categoria	2024-11-10 06:56:02.790
2	AIRBNB * HMMMC2Z3CS	-1971.97	Sem categoria	2024-11-09 22:51:00.000
3	Pix enviado para Gracilene Moreira de Barros	-50.00	Presente	2024-11-07 00:28:12.383
4	Pix recebido de Lucas Martins de Barros	1280.00	Transferências contas	2024-11-06 16:36:18.967
5	TRANSFERENCIA PIX - DES LUCAS MARTINS DE BARR 06/11 - DOCTO: 1036190	-1280.00	Transferências contas	2024-11-06 13:36:19.000

(d) Transações recentes

Fonte: O Autor (2024).

REFERÊNCIAS

ASSIS JOãO V. O. RODRIGUES, G. H. N. G. H. F. Bill: Desenvolvimento e validação de uma aplicação móvel para gerenciamento financeiro pessoal. **Revista Eletrônica de Computação Aplicada**, v. 1, n. 1, 2020. Disponível em: <http://periodicos.unifacef.com.br/reca/article/view/2039/1440>.

BACEN. **Qual a diferença entre Open Banking e Open Finance?** 2022. <www.openfinancebrasil.org.br/2022/11/17/qual-a-diferenca-entre-open-banking-e-open-finance/> [Acesso em: 19 de abr. de 2024.].

____. **Conheça o Open Finance**. 2023. <www.openfinancebrasil.org.br/conheca-o-open-finance/> [Acesso em: 13 de mai. de 2024].

____. **Documentação técnica - Open Finance**. 2024. <www.openfinancebrasil.atlassian.net/wiki/spaces/OF/pages/> [Acesso em: 13 de mai. de 2024].

____. **Open Finance**. 2024. <www.openfinancebrasil.org.br> [Acesso em: 19 de abr. de 2024.].

BELVO. **Belvo**. 2024. <www.developers.belvo.com/reference/using-the-api-reference> [Acesso em: 19 de mai. de 2024].

BRADESCO. **Aplicativos Bradesco**. s.d. <https://banco.bradesco/aplicativo-bradesco> [Acesso em: 16 de nov. de 2024.].

BUCKLEY, R.; ARNER, D.; BARBERIS, J. The evolution of fintech: A new post-crisis paradigm? **Georgetown Journal of International Law**, v. 47, p. 1271–1319, 01 2016.

CHOUDHARY, R. A.; MUDI, T.; SHARMA, S. A comparative analysis of native and hybrid mobile application development. **International Journal of Interactive Mobile Technologies (iJIM)**, n. 5, 2020.

CÔRTES, P. **ADMINISTRAÇÃO DE SISTEMAS DE INFORMAÇÃO**. Saraiva Educação S.A., 2017. ISBN 9788502108561. Disponível em: <https://books.google.com.br/books?id=Tx1nDwAAQBAJ>.

DOAN, A.; HALEVY, A.; IVES, Z. **Principles of Data Integration**. Morgan Kaufmann, 2012. Disponível em: <https://doi.org/10.1016/C2011-0-06130-6>.

EL-SAPPAGH, S.; HENDAWI, A.; EL-BASTAWISSY, A. A proposed model for data warehouse etl processes. **Journal of King Saud University - Computer and Information Sciences**, v. 23, p. 91–104, 07 2011.

GAMMA, E. *et al.* **Padrões de Projetos: Soluções Reutilizáveis**. Bookman, 2000. ISBN 9788573076103. Disponível em: <https://books.google.com.br/books?id=V0Ey1KF3zwcC>.

HAMEED, M.; NAUMANN, F. Data preparation: A survey of commercial tools. **SIGMOD Rec.**, Association for Computing Machinery, New York, NY, USA, v. 49, n. 3, p. 18–29, dec 2020. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/3444831.3444835>.

HE, W.; XU, L. D. Integration of distributed enterprise applications: A survey. **IEEE Transactions on Industrial Informatics**, v. 10, n. 1, p. 35–42, 2014.

IDRIS, N.; FOOZY, C. F. M.; SHAMALA, P. A generic review of web technology: Django and flask. **International Journal of Advanced Science Computing and Engineering**, v. 2, n. 1, p. 34–40, 2020.

MICROSOFT. **Microsoft SQL database functions**. 2023. Disponível em: <https://learn.microsoft.com/en-us/sql/t-sql/functions/functions?view=sql-server-ver16>.

____. **Sobre os segredos do Azure Key Vault**. 2024. Disponível em: <https://learn.microsoft.com/pt-br/azure/key-vault/secrets/about-secrets>.

NAMBIAR, A.; MUNDRA, D. An overview of data warehouse and data lake in modern enterprise data management. **Big Data and Cognitive Computing**, v. 6, n. 4, 2022. ISSN 2504-2289. Disponível em: <https://www.mdpi.com/2504-2289/6/4/132>.

NWOKEJI, J.; MATOVU, R. **A Systematic Literature Review on Big Data Extraction, Transformation and Loading (ETL)**. [S.l.: s.n.], 2021. 308-324 p. ISBN 978-3-030-80125-0.

PLUGGY. **Pluggy**. 2024. <www.pluggy.ai/> [Acesso em: 18 de mai. de 2024].

____. **Pluggy**. 2024. <www.docs.pluggy.ai/docs> [Acesso em: 18 de mai. de 2024].

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software - 9.ed**. [S.l.]: McGraw Hill Brasil, 2021.

REIS, J.; HOUSLEY, M. **Fundamentals of Data Engineering: Plan and Build Robust Data Systems**. O'Reilly, 2022. ISBN 9781098108304. Disponível em: <https://books.google.com.br/books?id=Z_TFzgEACAAJ>.

VASSILIADIS, P.; SIMITSIS, A. Extraction, transformation, and loading. **Encyclopedia of Database Systems**, Citeseer, v. 10, 2009.