Desenvolvimento de uma Biblioteca para Automação de Sistemas Legados em uma Cooperativa de Crédito

Sidimar Salla[1], **William Moraes da Silva**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Campus Farroupilha

sidimarsalla@gmail.com, william.silva@farroupilha.ifrs.edu.br

Abstract. *This paper presents the development of a Python-based library designed to automate a legacy system used by a credit cooperative. The initiative addresses the need to optimize repetitive tasks, reduce operational errors and improve internal process efficiency, while considering the specific constraints of a traditional corporate environment. An incremental and iterative development methodology was adopted, preceded by an analysis of the cooperative's legacy system. The result was a library that is both easy to understand and simple to use, enabling developers to implement automation routines efficiently, even with minimal learning time. The proposed solution proved to be a promising tool to increase technical team productivity and facilitate the deployment of automated processes within the institution, offering a practical and accessible alternative for similar scenarios.*

Resumo. *Este artigo apresenta o desenvolvimento de uma biblioteca em Python voltada à automação de um sistema legado utilizado por uma cooperativa de crédito. A proposta surge da necessidade de otimizar tarefas repetitivas, reduzir erros operacionais e melhorar a eficiência dos processos internos, respeitando as particularidades de um ambiente corporativo tradicional. Para isso, adotou-se a metodologia de desenvolvimento incremental e iterativo, precedida por um estudo do funcionamento e das limitações do sistema legado em uso. O resultado obtido foi uma biblioteca de fácil compreensão e uso simplificado, permitindo que desenvolvedores, mesmo com pouco tempo de familiarização, consigam aplicar automações de forma eficiente. A solução demonstrou-se promissora para ampliar a produtividade da equipe técnica e facilitar a implementação de rotinas automatizadas dentro da instituição, representando uma alternativa viável e acessível para contextos semelhantes.*

1. Introdução

Nos últimos anos, o avanço das tecnologias de automação tem transformado significativamente a forma como organizações operam, especialmente em ambientes onde há sistemas legados e processos ainda fortemente dependentes de intervenções humanas (MELLO; SANTOS, 2020). No contexto das cooperativas de crédito, essas transformações ganham contornos ainda mais desafiadores, dado o compromisso dessas instituições com a transparência, a confiança e a participação dos associados (SILVA; PEREIRA, 2021). Tais valores impõem um equilíbrio delicado entre inovação tecnológica e a preservação dos princípios cooperativistas (OCB, 2022).

Na cooperativa em questão, situada na Serra Gaúcha, há um crescimento constante da base de associados e da complexidade operacional, fazendo com que soluções tecnológicas eficazes sejam necessárias (BANCO CENTRAL DO BRASIL, 2024). Dentre essas soluções, destaca-se o desenvolvimento de ferramentas de automação capazes de interagir com sistemas legados, a redução de erros operacionais, padronização de rotinas e o aumento da produtividade, sem comprometer a segurança da informação ou a governança institucional (MELLO; SANTOS, 2020).

A automação em cooperativas de crédito envolve ganhos de eficiência, redução de custos, necessidade de harmonizar tecnologia, segurança de dados (em conformidade com a LGPD) e ideais cooperativistas. A capacidade de equilibrar inovação tecnológica e manutenção dos princípios fundamentais do cooperativismo é determinante para que essas instituições acompanhem a evolução do mercado, atendam às expectativas dos associados e mantenham seu impacto social e econômico (OCB, 2022; BRASIL, 2018).

O presente artigo propõe o desenvolvimento de uma biblioteca em Python voltada à automação de processos em sistemas legados, com ênfase na simplicidade de uso, segurança e compatibilidade com o ambiente operacional da cooperativa de crédito utilizada como referência para o trabalho. A proposta busca tornar o desenvolvimento de automações mais simples e garantir rastreabilidade dos processos automatizados, criptografia de dados sensíveis e adaptabilidade a diferentes fluxos de trabalho internos.

A motivação para o desenvolvimento deste projeto decorre de experiências pessoais vivenciadas ao iniciar em uma cooperativa na Serra Gaúcha, onde constatei a ausência de padrões definidos para automações e a falta de uma biblioteca com métodos eficientes que facilitassem o desenvolvimento dessas automações.

Este artigo está estruturado da seguinte forma: na seção 2, apresenta-se o referencial teórico, que aborda temas como o sistema financeiro cooperativo, seus impactos, o uso de terminais no Windows, o desenvolvimento de bibliotecas, a LGPD e trabalhos relacionados. A seção 3 descreve os objetivos do projeto. Na seção 4, discorre-se sobre a metodologia adotada. A seção 5 detalha o processo de implementação. A seção 6 expõe as limitações encontradas durante o desenvolvimento. Por fim, a seção 7 apresenta as conclusões do trabalho.

2. Referencial Teórico

2.1. Sistema Financeiro Cooperativo

As cooperativas de crédito têm suas origens na Europa do século XIX, quando Friedrich Wilhelm Raiffeisen, a fim de auxiliar pequenos produtores e trabalhadores rurais com o acesso a serviços financeiros, organizou grupos de auxílio mútuo que permitiam aos associados poupar coletivamente e obter pequenos empréstimos a juros acessíveis, minimizando a dependência de agiotas e instituições financeiras convencionais, assim criando cooperativas de crédito rural sem fins lucrativos, com gestão democrática, responsabilidade ilimitada entre os associados, área de atuação restrita geograficamente e distribuição dos resultados proporcional à participação (RAIFFEISEN, 1970; BEIRUTH, 2014; BANCO CENTRAL DO BRASIL, 2017).

A partir disso, o surgimento do movimento cooperativista de crédito no Brasil se deu ligado à atuação do padre jesuíta Theodor Amstad, reconhecido como precursor deste

movimento no país. Em 1902, na localidade de Linha Imperial, no município de Nova Petrópolis, Rio Grande do Sul, Amstad fundou a primeira cooperativa de crédito rural brasileira. Ela foi estruturada com base nos princípios do sistema Raiffeisen, de origem alemã, o qual priorizava a solidariedade, a autogestão e a ajuda mútua entre os membros das comunidades rurais. Sua iniciativa representou um marco no processo de inclusão financeira de pequenos agricultores e imigrantes na região sul do Brasil, contribuindo para o fortalecimento da economia local e a construção de laços comunitários sustentáveis (SCHNEIDER 2002; RIBEIRO, 2011).

De acordo com o Banco Central do Brasil (2017), a Confederação Brasileira das Cooperativas de Crédito (CONFEBRAS, 2020) e estudos acadêmicos como os de Schneider (2002) e Beiruth (2014), a atuação de Amstad estabeleceu as bases do sistema cooperativo de crédito brasileiro, o qual, atualmente, constitui uma alternativa consolidada e resiliente frente ao sistema financeiro convencional. Além disso, pesquisas históricas destacam que o modelo cooperativista, promovido por Amstad, fomentou o desenvolvimento regional por meio da educação financeira, da democratização do acesso ao crédito e da participação ativa dos associados na gestão das cooperativas (RIBEIRO, 2011; LIMA, 2019).

A evolução normativa e operacional das cooperativas de crédito faz com que essas sejam definidas pelo Banco Central do Brasil como "instituição financeira formada pela associação de pessoas para prestar serviços financeiros exclusivamente aos seus associados" (BANCO CENTRAL DO BRASIL, 2023).

No contexto regional da Serra Gaúcha, a trajetória das cooperativas de crédito acompanha o fortalecimento do cooperativismo como vetor de desenvolvimento socioeconômico. Estudos e relatórios locais mostram que, nos últimos anos, o setor teve expansão significativa: em 2021 registrou-se crescimento de cerca de 16% no faturamento das cooperativas da região, alcançando R$ 5,3 bilhões, pautado sobretudo pela relevância do agronegócio local e pela diversificação de serviços financeiros oferecidos (RBS, 2022).

Além disso, observa-se o aumento contínuo do número de cooperativas e de associados, refletindo a adesão de comunidades à proposta de crédito solidário e à gestão colaborativa. Isso contribui para elevar a participação dessas instituições no Sistema Nacional de Cooperativismo de Crédito e para ampliar a capilaridade do acesso financeiro nas localidades (BANCO CENTRAL DO BRASIL, 2024)

2.2. Impactos da automação nos processos

Embora o crescimento do setor de cooperativas de crédito na Serra Gaúcha represente um avanço histórico em termos de inclusão e expansão financeira, ele também acarreta desafios operacionais relevantes. O principal deles é a necessidade de aprimorar a eficiência operacional para suportar o aumento constante do número de associados, garantindo ao mesmo tempo a manutenção da qualidade do atendimento. De fato, conforme aponta o Banco Central do Brasil, o segmento de cooperativas de crédito vem melhorando sua eficiência operacional e ganhando escala, mas essa expansão exige investimentos em processos e tecnologia para sustentar o crescimento sem degradar a experiência dos cooperados (BANCO CENTRAL DO BRASIL, 2021). Nesse mesmo sentido, a Organização das Cooperativas Brasileiras destaca que o avanço das cooperativas impõe a necessidade de modernização tecnológica, ampliação da infraestrutura e capacitação das equipes para

garantir um atendimento eficiente e personalizado (OCB, 2023). Complementarmente, o relatório de sustentabilidade da Sicredi reforça que a ampliação da base de associados demanda melhorias contínuas na gestão de processos e na inovação das ferramentas de atendimento (SICREDI, 2022).

Além disso, a concorrência crescente de bancos tradicionais e fintechs pressiona as cooperativas a inovar continuamente em produtos e canais de atendimento. Nesse contexto, a automação de processos surge como um caminho estratégico para otimizar custos e liberar a equipe para atividades de maior valor agregado. Segundo a Organização das Cooperativas Brasileiras, as cooperativas de crédito precisam adotar soluções tecnológicas como automação e inteligência artificial para manter a competitividade e garantir a sustentabilidade operacional em um mercado cada vez mais digital e competitivo (OCB, 2023).

Estudos da Capgemini (2016) indicam que a adoção de soluções de *Robotic Process Automation* (RPA) pode representar economia de custos significativa, chegando a uma fração de 1/3 a 1/5 do custo de um funcionário em tempo integral (FTE) em determinadas atividades repetitivas.

Do mesmo modo, Lacity e Willcocks (2015) observam que um único "robô" de *software* (bot), sequência de etapas pré definidas que executam uma tarefa repetitiva dentro do sistema, pode executar tarefas estruturadas equivalentes ao trabalho de dois a cinco colaboradores humanos, destacando o potencial de ampliação de capacidade sem aumento proporcional de equipe.

No âmbito prático, a automação melhora imediatamente a eficiência operacional ao automatizar tarefas rotineiras e repetitivas que anteriormente demandavam esforços humanos consideráveis. Isso não apenas reduz custos operacionais, mas também permite que as cooperativas processem um volume maior de transações em menos tempo, elevando a agilidade no atendimento e, consequentemente, a satisfação dos associados. Segundo o Banco Central do Brasil (2021), a automação de processos é um dos pilares para o ganho de escala e eficiência no segmento cooperativista. Outro benefício reside na redução de erros: processos automatizados diminuem a probabilidade de falhas humanas, garantindo consistência, segurança e reforçando a confiança dos cooperados (OCB, 2023; BRITO et al., 2021).

Com as tarefas operacionais aliviadas, os profissionais podem dedicar-se a atividades estratégicas, como o desenvolvimento de novos produtos, expansão de mercado e parcerias, favorecendo a competitividade frente a outros bancos e fintechs. Segundo a Deloitte (2020), a automação libera recursos humanos para tarefas de maior valor agregado, promovendo inovação e fortalecendo a posição das instituições financeiras em um mercado cada vez mais dinâmico e competitivo.

2.3. Terminais de Windows

O terminal no ambiente Windows tem evoluído consideravelmente ao longo das últimas décadas, deixando de ser uma interface meramente textual e passando a incorporar recursos gráficos e interativos que o aproximam das necessidades de automação modernas. Em essência, o terminal pode ser compreendido como um intermediador entre o usuário e o sistema operacional, permitindo o envio de comandos e a recepção de respostas via interface de linha de comando (CLI – *Command Line Interface*).

Historicamente, o Windows utilizou o *Prompt* de Comando (CMD) como seu terminal padrão. Baseado na arquitetura do MS-DOS, o CMD ainda é amplamente utilizado para tarefas simples de automação, *scripts* em lote (.bat) e interações básicas com o sistema de arquivos. No entanto, sua limitação estrutural, baseada na manipulação de texto puro, restringe sua capacidade de lidar com dados complexos e integrados (OLIVEIRA, 1997).

Com a introdução do *prompt* de comando PowerShell em 2006, houve um avanço significativo na capacidade de automação e controle do sistema via terminal. Diferentemente do CMD, o PowerShell permite a manipulação de dados estruturados por meio de *cmdlets*, comandos especializados usados no PowerShell, o shell de linha de comando e linguagem de script da Microsoft, e o uso de pipelines, forma de encadear *cmdlets* que transferem objetos entre comandos e não apenas texto (MICROSOFT, 2023).

2.4. Desenvolvimento de bibliotecas

O desenvolvimento de bibliotecas em Python constitui uma prática fundamental para a criação de soluções reutilizáveis, escaláveis e modulares. Bibliotecas são conjuntos de módulos organizados e empacotados com o objetivo de oferecer funcionalidades específicas para reutilização em diversos contextos de programação. No ecossistema Python, essa prática é fortemente incentivada por sua sintaxe simples e a ampla adoção da linguagem na comunidade científica e no setor corporativo (LUTZ, 2013).

A criação de bibliotecas promove a padronização do código, além da manutenção e extensibilidade das soluções. Ao encapsular funcionalidades comuns, as bibliotecas reduzem a duplicação de código, facilitam a integração com outros sistemas e permitem o versionamento de funcionalidades (GAMMA et al., 1995; FOWLER, 2019). Isso é especialmente relevante em ambientes de automação, como o de instituições financeiras, onde diferentes aplicações podem depender de um mesmo núcleo de automação para interagir com sistemas legados, exigindo reuso de componentes confiáveis, segurança e compatibilidade contínua (SOMMERVILLE, 2011; MEDEIROS; SANTOS, 2021).

A linguagem Python oferece ferramentas nativas para a estruturação e distribuição de bibliotecas, como o uso do arquivo *setup.py* com o pacote *setuptools*, que permite a instalação da biblioteca por meio de gerenciadores como o *pip*. Além disso, o repositório público PyPI (*Python Package Index*) serve como principal canal de distribuição de bibliotecas *open-source*, incentivando a colaboração entre desenvolvedores e a disseminação de boas práticas de codificação (VAN ROSSUM E DRAKE, 2001). O processo de desenvolvimento de bibliotecas envolve etapas como: definição dos requisitos, organização do código em módulos, documentação, testes automatizados e integração contínua. Essas práticas contribuem para o alinhamento com os princípios da engenharia de *software*, tais como coesão, acoplamento fraco e reutilização (MARTIN, 2009).

Em contextos específicos, como o de automação para sistemas legados, o uso de bibliotecas customizadas permite abstrair comandos repetitivos, interações com interfaces gráficas e comunicação com bancos de dados. Essa abordagem modularizada é essencial para a construção de *frameworks* de automação que sejam adaptáveis às mudanças e restrições dos sistemas utilizados em ambientes corporativos. Segundo Mello e Santos (2020), a utilização de bibliotecas específicas e estruturadas facilita a manutenção e a escalabilidade das automações, principalmente quando aplicadas a sistemas legados com

restrições técnicas e operacionais.

2.5. Lei Geral de Proteção de Dados (LGPD)

A crescente digitalização das relações sociais e comerciais intensificou a coleta e o processamento de dados pessoais, sendo necessária a criação de normativas que assegurem a privacidade e os direitos fundamentais dos cidadãos. Nesse contexto, foi sancionada no Brasil a Lei nº 13.709/2018, conhecida como Lei Geral de Proteção de Dados Pessoais (LGPD), com o objetivo de estabelecer diretrizes claras para o tratamento de dados pessoais por pessoas naturais e jurídicas, tanto no setor público quanto no privado (BRASIL, 2018).

A LGPD foi inspirada em regulamentações internacionais, especialmente o Regulamento Geral sobre a Proteção de Dados (General Data Protection Regulation – GDPR) da União Europeia, adotando diretrizes semelhantes no que diz respeito à transparência, segurança e responsabilidade no tratamento de dados pessoais (DONEDA, 2021; PINHEIRO, 2020). A legislação brasileira define como dado pessoal qualquer informação relacionada a uma pessoa natural identificada ou identificável, e como dado sensível aqueles dados que revelem origem racial ou étnica, convicção religiosa, opinião política, filiação a sindicato ou a organização de caráter religioso, filosófico ou político, bem como dados referentes à saúde, à vida sexual, ao dado genético ou biométrico, quando vinculado a uma pessoa natural (BRASIL, 2018).

Entre os princípios da LGPD, destacam-se: a finalidade, que exige que o tratamento de dados tenha propósitos legítimos, específicos e explícitos; a necessidade, que limita a coleta ao mínimo necessário; a segurança, que impõe a adoção de medidas técnicas e administrativas para proteger os dados contra acessos não autorizados e vazamentos (DONEDA, 2019).

A lei também prevê os direitos dos titulares, como o acesso, correção, exclusão e portabilidade de seus dados, além da possibilidade de revogação do consentimento a qualquer momento. Para a fiscalização e aplicação da LGPD, foi criada a Autoridade Nacional de Proteção de Dados (ANPD), responsável por zelar, implementar e fiscalizar o cumprimento da legislação no Brasil (BRASIL, 2020).

Nesse sentido, o desenvolvimento de sistemas, bibliotecas e ferramentas de automação também deve incorporar mecanismos de proteção de dados desde sua concepção, em conformidade com o princípio de *privacy by design*, garantindo que a segurança e a privacidade estejam embutidas no ciclo de vida do *software* (CAVOUKIAN, 2011).

2.6. Trabalhos prévios

Poucos trabalhos foram encontrados na literatura sobre a automação de processos em cooperativas de crédito. Entretanto, destaca-se o trabalho de Rovaris (2023), que tem como foco central a análise dos fatores que influenciam a decisão de adoção da tecnologia *Robotic Process Automation* (RPA) em cooperativas de crédito, utilizando como base a teoria TOE (*Technology-Organization-Environment*). Sua abordagem consiste em mapear métricas e motivadores através de revisão sistemática da literatura e estudo de caso com diversas cooperativas, visando compreender os aspectos organizacionais, tecnológicos e ambientais que impactam essa decisão. Diferentemente disso, o presente trabalho não

se debruça sobre o processo decisório de adoção, mas sim sobre a criação prática de uma biblioteca em Python que efetivamente implementa a automação de processos em sistemas legados de uma cooperativa específica.

Enquanto Rovaris (2023) estrutura seu estudo a partir de entrevistas, observações e coleta documental para analisar o cenário de adoção da RPA em múltiplas unidades do sistema cooperativo, o foco deste trabalho está na construção técnica de uma ferramenta de automação customizada. Esta pesquisa propõe uma solução de baixo custo e de fácil utilização, voltada à realidade operacional de uma cooperativa da Serra Gaúcha, com ênfase na compatibilidade com sistemas legados, segurança dos dados e simplicidade de desenvolvimento. Assim, ao contrário do enfoque teórico e estratégico da autora, este estudo adota uma abordagem prática, incremental e aplicada, com objetivo de oferecer resultados tangíveis e utilizáveis diretamente no cotidiano institucional.

Por fim, enquanto a contribuição prática do trabalho de Rovaris (2023) se dá por meio de um *framework* que pode orientar futuras decisões de adoção tecnológica em cooperativas, este estudo entrega um produto funcional e reutilizável: uma biblioteca modularizada que permite aos desenvolvedores implementar automações com facilidade. Assim, a principal diferença entre os trabalhos reside no escopo e na aplicação: Rovaris contribui com um modelo analítico para orientar decisões estratégicas, enquanto este trabalho oferece uma ferramenta técnica implementável, contribuindo diretamente para a transformação digital operacional das cooperativas.

3. Objetivos

3.1. Objetivo geral

Desenvolver uma biblioteca em Python destinada à automação de processos em um sistema legado de uma cooperativa de crédito.

3.2. Objetivos específicos

- Desenvolver um algoritmo para automatizar processos em um sistema legado.
- Integrar a biblioteca com o sistema legado.Permitir interação com múltiplas janelas simultaneamente.
- Preconizar por uma estrutura escalável e de fácil manutenção.
- Desenvolver uma interface simples e acessível.
- Garantir a aplicação das normas definidas pela LGPD.

4. Metodologia

Para alcançar os objetivos propostos, optou-se por uma abordagem prática baseada no desenvolvimento incremental e iterativo da solução. Para tal, foi realizado um estudo detalhado da documentação do sistema legado, fornecida pela cooperativa, a qual apresenta explicações sobre o funcionamento operacional do sistema. A partir disso foi identificado que o sistema legado possuía uma maneira de ser iniciado utilizando um comando, evitando toda a interação com a sua interface gráfica de login, facilitando ainda mais o processo de inicialização. Além disso, foram realizadas conversas com os responsáveis pelo sistema para sanar dúvidas sobre o comando e o sistema legado. Com isso, foram levantados os requisitos descritos no quadro 1.

Tipo do Requisito	Requisito
Funcionais	Validar se o nome do usuário foi informado.
	Validar a senha do usuário, gerando erro se: o primeiro caractere for "@" ou o terceiro caractere for "_".
	Validar se o nome do sistema a ser acessado foi informado.
	Validar o tipo do sistema (transacional ou relatório); usar "transacional" como padrão se não informado.
	Validar a unidade de atendimento (UA); usar "none" como padrão se não informado.
	Validar se o número da cooperativa (coop) foi informado.
	Gerar o comando de inicialização do sistema.
	Manipular janelas do sistema (abrir, focar, minimizar, etc).
	Realizar a leitura do conteúdo da tela.
	Enviar texto para o sistema.
	Enviar teclas para o sistema (atalhos, comandos).
	Executar interações apenas se um elemento específico for detectado na tela.
	Permitir informar o nome da janela a ser usada no comando de abertura.
	Identificar se o sistema não foi aberto corretamente e retornar uma resposta adequada.
	Identificar elementos específicos na tela.
	Registrar logs das operações para rastreabilidade.
	Executar o comando no terminal da máquina.
Não Funcionais	Utilizar criptografia para proteger dados sensíveis, como senhas e acessos.
	Manter registros de log organizados, acessíveis e auditáveis para análise posterior.

Quadro 1: Requisitos Funcionais e Não Funcionais.
Fonte: Elaborado pelo autor.

Entre os requisitos funcionais, destacam-se a validação dos dados de entrada (usuário, senha, sistema, tipo de sistema, unidade de atendimento e número da cooperativa), a geração do comando de inicialização e a automação de interações com a interface do sistema. Já entre os requisitos não funcionais, destaca-se a presença da criptografia, que se enquadra nessa categoria por ser uma exigência aplicada especificamente no ambiente de produção. Esse requisito não interfere diretamente na funcionalidade da automação, mas é essencial para garantir a segurança das credenciais e a rastreabilidade das ações por meio de registros de log, atendendo às diretrizes de proteção de dados e auditoria.

Com base nisso, a biblioteca de automação passou a focar na construção e execução da linha de comando necessária para iniciar o sistema. Essa execução exige que a biblioteca seja capaz de receber e validar parâmetros, formatando-os corretamente

para garantir que o sistema seja aberto de forma eficiente e confiável. Reuniões com a equipe de desenvolvimento da cooperativa também contribuíram para a definição de que a biblioteca deveria interagir diretamente com a janela do sistema legado. Isso evita o uso de comandos de teclado e mouse baseados em coordenadas, que são mais suscetíveis a falhas devido a variações visuais na interface.

Durante o desenvolvimento, também foram conduzidas reuniões com desenvolvedores de outras cooperativas que utilizam sistemas similares. Nessas conversas, reforçou-se a importância de interações mais robustas com a interface e confirmou-se a necessidade da biblioteca identificar corretamente a janela do sistema aberto, detectando elementos específicos em tela e realizando ações condicionais com base nessa detecção.

Para o algoritmo, utilizou-se a linguagem de programação Python (versão 3.13.11), em conjunto com o ambiente virtual gerenciado pelo Pipenv, ferramenta nativa do Python. Como ambiente de desenvolvimento integrado (IDE), foi utilizado o Visual Studio Code, escolhido por sua ampla adoção na comunidade Python, facilidade de uso e grande volume de recursos e extensões disponíveis.

Optou-se por estruturar a classe principal com suporte ao protocolo de contexto do Python, com o objetivo de garantir que os recursos fossem corretamente liberados após a execução de tarefas críticas, promovendo maior robustez e confiabilidade no código.

A biblioteca Pywinauto foi utilizada para automação de interfaces gráficas no ambiente Windows, permitindo interação programática com janelas e controles de aplicativos. A Win32API e a Win32CON foram empregadas para chamadas de baixo nível ao sistema operacional, possibilitando o envio de comandos diretamente a janelas e processos, além do controle de eventos e teclas. A biblioteca Pydantic foi utilizada para a validação e estruturação de dados, garantindo a integridade das configurações e entradas recebidas pela biblioteca. Para o tratamento criptográfico das credenciais, foi utilizada a PyCryptodome, que oferece algoritmos robustos de criptografía e hash. A biblioteca OS, nativa do Python, foi utilizada para interação com o sistema de arquivos e execução de comandos no terminal, funções essenciais no contexto da automação.

Esse conjunto de decisões técnicas, alinhado aos requisitos definidos e às restrições operacionais e de segurança da cooperativa, orienta o desenvolvimento da biblioteca. O algoritmo busca uma solução automatizada, segura, rastreável e eficaz para a integração com o sistema legado da instituição.

5. Implementação

A implementação foi conduzida de forma incremental e iterativa, permitindo validações contínuas a cada novo componente desenvolvido. Essa abordagem favoreceu a identificação precoce de falhas e a adaptação rápida da arquitetura da solução conforme novas demandas foram surgindo durante o desenvolvimento.

No início do processo, foi estruturado o ambiente de desenvolvimento, o que incluiu a configuração do sistema legado e a instalação de todas as dependências e bibliotecas essenciais para o funcionamento do projeto. Essa etapa foi fundamental para assegurar que todas as ferramentas estivessem adequadamente preparadas para suportar as fases subsequentes do desenvolvimento, garantindo compatibilidade e desempenho satisfatório.

Posteriormente, deu-se início ao desenvolvimento da classe modelo, responsável

pela validação dos dados de entrada, conforme ilustrado na Figura 1. Para isso, utilizou-se a herança da classe BaseModel da biblioteca Pydantic, recurso que permite a criação de um modelo de dados fortemente tipado, com validação automática e conversão de tipos conforme os atributos definidos. Essa escolha tecnológica trouxe vantagens significativas, como a redução de erros decorrentes de dados inválidos, maior clareza no tratamento das informações e facilitação da manutenção futura do código.

```python
class ACClientAccessConfig(BaseModel):
    user: User
    system: Literal["siat", "siac", "sacg"]
    database: Literal["relatorio", "transacao"] = "transacao"
    num_coop: str
    ua: Optional[str] = None

    @field_validator('ua')
    def validate_ua(cls, ua: Optional[str]):
        if ua is not None:
            len_ua = len(ua)
            if len_ua > 2:
                raise ValueError('UA deve ter no máximo 2 caracteres.')
            elif not len_ua:
                raise ValueError('UA não pode ser vazio.')
            elif len_ua == 1:
                return ua.zfill(2)

        return ua

    @field_validator('num_coop')
    def validate_num_coop(cls, num_coop: str):
        if isinstance(num_coop, int):
            return str(num_coop).zfill(4)
        elif isinstance(num_coop, str) and len(num_coop) < 4:
            return num_coop.zfill(4)
        return num_coop
```

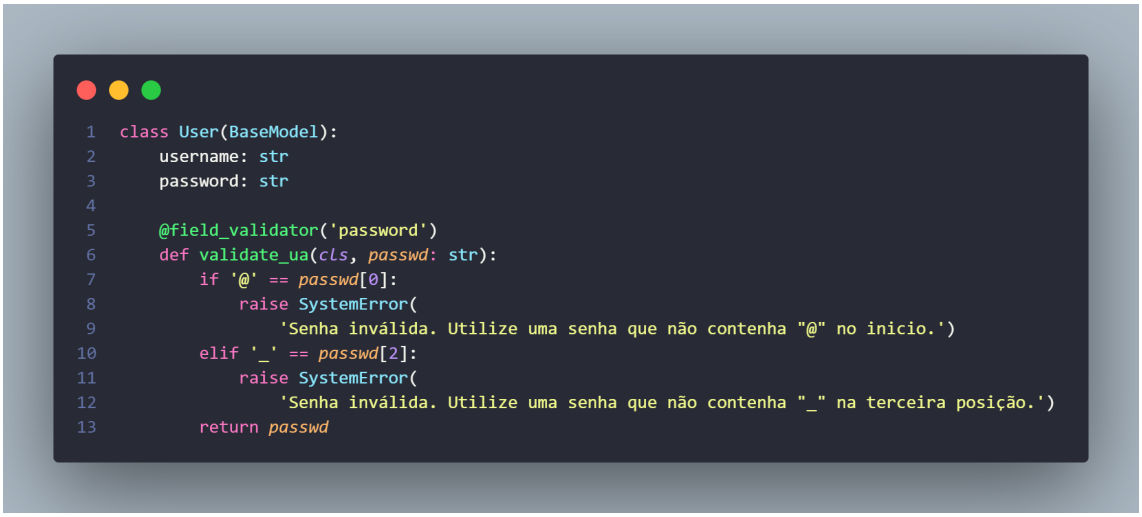
Figura 1: Classe para configuração do acesso ao sistema legado.
Fonte: Elaborado pelo autor.

Outra classe criada no desenvolvimento do projeto, ilustrada na Figura 2, foi a classe “Usuário”, que desempenha um papel fundamental como modelo validador dos dados de autenticação. Seu principal objetivo é garantir que os campos essenciais para a abertura do sistema sejam devidamente preenchidos e atendam aos critérios de segurança estabelecidos.

Para isso, a classe realiza verificações para assegurar que esses dados não estejam vazios e que a senha contenha somente caracteres permitidos, proibindo a utilização de caracteres especiais que possam comprometer a segurança ou a estabilidade do sistema. Sempre que a validação identifica a presença de caracteres não autorizados ou

informações faltantes, a classe gera um erro, impedindo o prosseguimento do processo de autenticação.

Essa funcionalidade é crucial para manter a integridade e a confiabilidade dos dados fornecidos pelo usuário, prevenindo falhas de segurança e assegurando que o sistema opere conforme as especificações.



```python
class User(BaseModel):
    username: str
    password: str

    @field_validator('password')
    def validate_ua(cls, passwd: str):
        if '@' == passwd[0]:
            raise SystemError(
                'Senha inválida. Utilize uma senha que não contenha "@" no inicio.')
        elif '_' == passwd[2]:
            raise SystemError(
                'Senha inválida. Utilize uma senha que não contenha "_" na terceira posição.')
        return passwd
```

Figura 2: Classe que valida usuario e senha.
Fonte: Elaborado pelo autor.

Pode-se observar que essa classe possui as validações de senha apontadas no quadro 1. Se for informada uma sequência de caracteres com essas exceções, o código retorna o erro correspondente nos logs.

Após a criação do modelo de dados com suas respectivas validações, são configuradas todas as constantes do projeto, como o caminho para o arquivo .BAT, o número de tentativas para leitura da tela, entre outras. Com os modelos e o arquivo de constantes criados, iniciou-se o desenvolvimento da classe responsável por toda a lógica de manipulação e interação com o sistema. Essa classe recebe os dados para inicialização do projeto por meio de parâmetros. Em sua função de inicialização, foram criadas algumas variáveis internas para armazenar informações utilizadas ao longo da execução da classe. Uma dessas variáveis é responsável pela validação dos dados (Figura 3), de forma que, ao identificar qualquer inconsistência, a execução seja encerrada imediatamente.

Para facilitar o uso seguro desses recursos e garantir o encerramento adequado das conexões com o sistema legado, a classe foi implementada com os métodos especiais *__enter__* e *__exit__*. Esses métodos possibilitam que a classe seja utilizada dentro do gerenciador de contexto *with*, um recurso nativo da linguagem Python que automatiza o gerenciamento de recursos, como abertura e fechamento de arquivos ou conexões. Essa abordagem não só torna o código mais legível e organizado, mas também assegura que os recursos sejam liberados corretamente mesmo em caso de exceções ou erros inesperados durante a execução, aumentando a robustez e segurança do sistema.

Durante o desenvolvimento dos módulos de contexto, foi iniciada, em paralelo, a criação da função responsável pela geração do comando de inicialização do sistema (Figura 3). Essa função tem papel central no processo de automação, pois constrói di-

namicamente o comando que será enviado para iniciar a interação com o sistema legado. Para tanto, ela utiliza as informações validadas anteriormente, garantindo que o comando gerado esteja sempre correto e parametrizado conforme as necessidades do projeto. O desenvolvimento dessa função envolveu atenção especial para o tratamento dos dados e a construção precisa do comando, fatores essenciais para o funcionamento eficiente e seguro da automação.

```python
def __build_command(self) -> str:
    logging.info("Construindo o comando para abrir o sistema.")
    command = (
        f'"{self.__configs.ACCLIENT_BAT_PATH}" '
        f'-u {self.__ac_client_access_config.user.username} '
        f'-p {self.__ac_client_access_config.user.password} '
        'sis-cmd '
        f'-s {self.__ac_client_access_config.system} '
        f'-c {self.__ac_client_access_config.num_coop} '
        '-x -l -v '
        f'TE_TITLE={self.__window_title}'
    )

    if self.__ac_client_access_config.ua is not None:
        logging.info("Setando unidade de atendimento.")
        command += f" --ua {self.__ac_client_access_config.ua}"

    if self.__ac_client_access_config.database == "relatorio":
        logging.info("Setando sistema de relatório.")
        command += " -b Relatórios"

    logging.debug("Comando construído: %s", command)
    return command
```

Figura 3: Função que cria o comando para inicialização do sistema.
Fonte: Elaborado pelo autor.

A função em questão utiliza todas as informações previamente inicializadas pela função apresentada na Figura 1, com o objetivo de construir e retornar uma string que contém o comando completo, devidamente parametrizado para garantir sua execução correta no ambiente desejado. O processo de criação desse comando envolve uma série de validações rigorosas dos dados de entrada, utilizando estruturas condicionais para verificar a consistência e a adequação dos parâmetros fornecidos. Além disso, a manipulação strings assegura que o comando final esteja formatado conforme os requisitos do sistema, minimizando a ocorrência de erros durante a execução e aumentando a confiabilidade do processo.

Após a construção e o armazenamento do comando em uma variável privada, foi

desenvolvida a função responsável por abrir o sistema, conforme ilustrado na Figura 4. Essa função foi projetada para realizar um número limitado de tentativas de abertura do sistema, buscando garantir maior robustez diante de falhas temporárias ou problemas na inicialização. Caso o sistema não seja aberto dentro das tentativas estipuladas, a função retorna um erro específico, notificando que não foi possível estabelecer a conexão com o sistema. Essa abordagem preventiva é fundamental para informar o usuário ou outros módulos sobre a falha, possibilitando a tomada de ações corretivas ou o registro adequado do problema para futuras análises.

```python
def open_system(self) -> None:
    """
    Abre o sistema AC Client.
    """
    logging.info("Abrindo o sistema AC Client.")
    attempts = self.__configs.ACCLIENT_ATTEMPTS_FOR_OPEN

    for count in range(attempts):
        try:
            logging.info("Tentativa %d de abrir o sistema.", count + 1)
            os.system(self.__command_for_open)
            time.sleep(3)
            self.__connect_window()

            if not self.find_text_in_screen_text("Retorna ao Sistema Operacional", 30):
                self.close_system()
                raise RuntimeError(
                    "Erro ao abrir o sistema: 'Retorna ao Sistema Operacional' não encontrado.")

            logging.info("Sistema aberto com sucesso.")
            break

        except RuntimeError as e:
            logging.error("Erro ao abrir o sistema: %s", e)
            self.app = None
            self.window = None

    else:
        logging.critical(
            "Não foi possível abrir o sistema %s após %d tentativas.",
            self.__ac_client_access_config.system,
            attempts
        )
        raise RuntimeError(
            f"Não foi possível abrir o sistema {self.__ac_client_access_config.system} "
            f"após {attempts} tentativas."
        )
```

Figura 4: Classe modelo para inicialização do sistema.
Fonte: Elaborado pelo autor.

A execução do comando no terminal foi realizada utilizando a biblioteca os, que permite a interação direta com o sistema operacional, assim utilizando um terminal para executar o comando. Após a execução desse comando, é acionada uma função desenvolvida para estabelecer a conexão com a janela do sistema que foi aberta, utilizando a biblioteca Pywinauto para controlar e manipular essa janela. Para garantir que o sistema tenha tempo suficiente para iniciar corretamente, a função aguarda um intervalo de três segundos antes de iniciar o processo de conexão com a janela.

Caso a conexão seja estabelecida com sucesso, os detalhes da janela são armazenados em uma variável privada, que será utilizada posteriormente nas funções responsáveis pelo envio de comandos e pela captura do texto exibido na tela (conforme ilustrado na Figura 5). Para realizar essa conexão, emprega-se a biblioteca Pywinauto, que oferece recursos para manipular janelas e controles de aplicações em execução no ambiente Windows, permitindo a automação precisa e confiável das interações com a interface gráfica.

```python
def __connect_window(self) -> None:
        """
        Conecta à janela da aplicação.
        """
        logging.info("Conectando à janela da aplicação.")
        self.app = Application(backend="uia").connect(
            title=self.__window_title)
        self.window = self.app.window(
            title=self.__window_title).wait('exists', timeout=30)
        logging.info("Conexão com a janela estabelecida com sucesso.")
```

Figura 5: Função para se conectar na janela do sistema legado.
Fonte: Elaborado pelo autor.

Essa mesma função para abertura do sistema é executada dentro da função *__enter__* para que seja executado de forma automática quando for executado no modo de contexto, ou seja, com o comando with.

Em seguida, foi desenvolvida a função de encerramento do sistema (Figura 6), responsável por fechar a janela que está em execução. Contudo, isso o encerramento ocorre apenas se há alguma aberta, ou seja, ele só executa o comando de encerramento caso a variável não estiver nula.

```python
def close_system(self) -> None:
    """
    Fecha o sistema AC Client.
    """
    logging.info("Fechando o sistema AC Client.")
    if self.window is not None:
        self.window.close()
        logging.info("Sistema fechado com sucesso.")
    else:
        logging.warning("Tentativa de fechar o sistema, mas a janela não estava aberta.")
```

Figura 6: Função que finaliza o sistema.
Fonte: Elaborado pelo autor.

Esta função é executada na *__exit__*, ou seja, toda a ver que o usuário sair do contexto, o sistema é fechado automaticamente, evitando que o sistema fique aberto na máquina e economizando recursos de máquina.

A função demonstrada na Figura 7, captura todo o conteúdo que está sendo mostrado no terminal do sistema, retornando uma string com o conteúdo. Essa função é tem grande impacto, pois permite realizar a verificação se o usuário está no campo certo para que o input seja assertivo em qualquer interação com o sistema.

```python
def get_screen_text(self) -> str:
        """
        Pega o texto da tela do sistema.

        Returns:
            str: Texto da tela do sistema.

        Exemplo de uso:
            screen_text = ac_client.get_screen_text()
        """
        logging.info("Obtendo o texto da tela do sistema.")
        if self.app is None:
            logging.error("A aplicação AC Client não está conectada (self.app é None).")
            raise RuntimeError("A aplicação AC Client não está conectada.")
        window_specification = self.app.top_window()
        window_object = window_specification.child_window(
            title="Text Area", control_type="Document").wrapper_object()
        text_screen = window_object.iface_text.DocumentRange.GetText(-1)
        logging.info("Texto da tela obtido com sucesso.")
        logging.debug("Texto da tela obtido: %s", text_screen)
        return text_screen
```

Figura 7: Função que captura o texto da tela.
Fonte: Elaborado pelo autor.

A Figura 8 exibe um trecho fundamental do código, responsável pela tarefa crucial de envio de comandos diretamente à janela do sistema. Para viabilizar essa funcionalidade, foram adotadas as bibliotecas especializadas Win32API e Win32CON, as quais são amplamente utilizadas para permitir uma interação mais profunda e eficaz com o sistema operacional Windows. A biblioteca Win32API, em particular, possibilita o acesso a um conjunto extenso de funções nativas da API do Windows que não estão disponíveis diretamente através da linguagem Python, o que a torna indispensável para a execução de comandos em cenários onde a janela do sistema está maximizada ou mesmo operando em segundo plano. Essa capacidade amplia significativamente a escalabilidade e flexibilidade da automação, ao possibilitar que múltiplos processos possam ser executados simultaneamente na mesma máquina, sem que haja interferência ou perda de desempenho.

Além disso, com o suporte robusto oferecido pela Win32API, os comandos são encaminhados de forma precisa para a janela que já foi previamente vinculada utilizando a biblioteca Pywinauto, responsável pelo gerenciamento das janelas e controles do ambiente Windows. De forma complementar, a biblioteca Win32CON fornece um conjunto de constantes essenciais para o envio de comandos de teclas específicas, englo-

bando desde caracteres alfabéticos simples até comandos especiais, como a tecla ”*Shift*”e outras combinações de teclas. Para assegurar que esses comandos sejam transmitidos corretamente e direcionados exatamente à janela desejada, utilizou-se a função PostMessage, a qual recebe como parâmetro o identificador da janela-alvo, garantindo assim uma comunicação eficiente e confiável entre o sistema de automação e a interface do usuário, sem riscos de desvios ou erros na execução.

```python
def send_input(self, input_value: str) -> None:
        """
        Envia texto ou comando de tecla para a janela do sistema.

        Args:
            input_value (str): Texto a ser enviado ou nome da tecla mapeada.

        Exemplo de uso:
            ac_client.send_input("texto a ser enviado")
            ac_client.send_input("enter")
        """
        logging.info("Enviando input: '%s'", input_value)
        if self.app is None:
            logging.error("A aplicação AC Client não está conectada (self.app é None).")
            raise RuntimeError("A aplicação AC Client não está conectada.")
        hwnd = self.app.top_window().wrapper_object().handle

        # Verifica se é uma tecla mapeada
        if input_value.lower() in self.__configs.ACLIENT_KEY_MAP:
            win32api.PostMessage(   #  ignore: 1101
                hwnd, win32con.WM_KEYDOWN,
                self.__configs.ACLIENT_KEY_MAP[input_value.lower()], 0
            )
            win32api.PostMessage(
                hwnd, win32con.WM_KEYUP,
                self.__configs.ACLIENT_KEY_MAP[input_value.lower()], 0
            )
            logging.info("Comando/tecla '%s' enviado com sucesso.", input_value)
        else:
            # Envia como texto comum
            for letter in input_value:
                if not letter.isspace():
                    win32api.PostMessage(hwnd, win32con.WM_CHAR, ord(letter), 0)
                else:
                    win32api.PostMessage(hwnd, win32con.WM_CHAR, win32con.VK_SPACE, 0)
                time.sleep(0.05)
            logging.info("Texto enviado com sucesso.")
```

Figura 8: Função para envio de comandos/texto.
Fonte: Elaborado pelo autor.

Antes da etapa de validação com os desenvolvedores das cooperativas, foram realizados testes prévios por mim, com o objetivo de verificar o correto funcionamento das funcionalidades implementadas. Esses testes serviram como base inicial para avaliar aspectos como a eficiência da automação, a compatibilidade com o sistema legado e a facilidade de uso da biblioteca, pontos esses levantados na introdução deste trabalho. Embora os resultados tenham sido observados e relatados, o registro formal foi feito internamente na empresa, não sendo possível apresentar telas ou exportar esses dados neste

trabalho por motivos de confidencialidade e proteção de informações sensíveis.

A validação das funcionalidades foi feita por desenvolvedores de várias cooperativas do sistema onde cada desenvolvedor criou algumas automações para validarem as funções que serviram como casos de testes práticos para verificar o correto funcionamento de cada recurso implementado na biblioteca. A cada nova automação criada, foram testadas situações reais de uso, permitindo identificar possíveis falhas, ajustar comportamentos e assegurar que os requisitos funcionais e não funcionais estavam sendo atendidos conforme o esperado. Essa abordagem garantiu maior confiabilidade à solução e contribuiu para a evolução contínua do projeto.

6. Limitações

Apesar dos requisitos propostos durante o desenvolvimento da biblioteca, é fundamental reconhecer algumas limitações que impactam sua aplicabilidade e desempenho em cenários específicos. Além disso, algumas restrições técnicas e funcionais ainda se fazem presentes.

Uma limitação já mencionada dentro do projeto relacionado à utilização de caracteres especiais no processo de autenticação via linha de comando. Durante os testes, verificou-se que o caractere "@" na primeira posição da senha e o caractere "_" na terceira posição não são interpretados corretamente pelo sistema, sendo ignorados e assim causando falha na autenticação. Essa limitação reduz a flexibilidade na criação de senhas e pode comprometer a execução automatizada caso essas combinações sejam utilizadas. É possível que outros caracteres ou padrões específicos também apresentem esse tipo de incompatibilidade, embora não tenham sido identificados nos testes realizados.

Uma das limitações mais significativas diz respeito à execução paralela do sistema automatizado. Embora a biblioteca tenha demonstrado capacidade de interagir com múltiplas instâncias do sistema abertas simultaneamente, o próprio sistema limita a três a quantidade de terminais que podem ser abertos com a mesma credencial de um único usuário em uma única máquina. Essa limitação já conhecida, compromete a escalabilidade da automação, especialmente em ambientes que demandam alto volume de execuções simultâneas. Assim, a reutilização de uma mesma conta de usuário em execuções paralelas torna-se inviável, exigindo alternativas como múltiplas credenciais ou máquinas dedicadas.

Outra limitação funcional observada durante o desenvolvimento está relacionada à ausência de suporte à simulação de combinações de teclas (atalhos). Embora esse recurso seja pouco utilizado atualmente, o sistema legado permite a configuração de atalhos programáveis que podem acelerar significativamente o acesso a funcionalidades específicas. A impossibilidade de automatizar essas combinações reduz o potencial de eficiência da biblioteca em certos fluxos de trabalho, representando um ponto crítico para futuras melhorias.

Além disso, destaca-se uma possível vulnerabilidade frente a atualizações do próprio sistema legado. A automação foi desenvolvida com base em uma versão específica do sistema, a qual recentemente passou a permitir sua inicialização via comandos. Caso futuras atualizações modifiquem esse processo de abertura ou os parâmetros necessários para sua execução, a biblioteca poderá se tornar incompatível ou exigir adaptações.

7. Conclusão

O objetivo deste estudo foi produzir uma biblioteca que fosse usada para automatizar sistemas legados de terminal de uma cooperativa de crédito. No conjunto dos objetivos traçados, os resultados mostram que a proposta pode ser uma opção de automação com tratamento de dados, escalabilidade e aplicação das normas instituídas na LGPD. Além disso, destaca-se a simplicidade que a biblioteca oferece para desenvolver novas automações, uma vez que a passagem de parâmetros permite que outros desenvolvedores apliquem nos sistemas legados, sem a necessidade de nova programação.

A estrutura modular da biblioteca, aliada à organização clara de seus componentes, mostrou-se fundamental para garantir escalabilidade, facilidade de manutenção e reutilização do código, o que reforça sua viabilidade de adoção em diferentes cenários operacionais da instituição. Essa arquitetura permitiu que o desenvolvimento das automações ocorresse de forma mais simples e padronizada, reduzindo significativamente o tempo necessário para construção de fluxos, o que impacta diretamente na produtividade das equipes envolvidas.

Durante os testes realizados *in loco* na instituição e também por desenvolvedores de outras cooperativas do sistema, foi possível observar de forma prática os benefícios propostos na fase inicial do projeto. A rastreabilidade dos processos automatizados foi plenamente atendida por meio dos registros de log estruturados, enquanto a criptografia de dados sensíveis garantiu conformidade com as políticas de segurança da informação. Além disso, a flexibilidade da biblioteca para se adaptar a diferentes fluxos de trabalho internos se confirmou como um diferencial, atendendo a diversas demandas operacionais com mínima necessidade de ajustes no código. Esses resultados, validados em diferentes ambientes, comprovam a eficácia da proposta e reforçam sua aderência às necessidades reais do contexto corporativo.

Ainda, a solução proposta está alinhada às exigências de segurança da informação, considerando práticas como armazenamento criptografado de credenciais e registros de logs para fins de auditoria, suprindo as demandas da instituição e da própria LGPD. A escolha por tecnologias amplamente consolidadas, como Python, também favorece a portabilidade e a integração com outras ferramentas corporativas.

No entanto, é importante reconhecer que algumas limitações foram identificadas nesta versão inicial, especialmente no que se refere a restrições técnicas impostas pelo sistema automatizado. Um exemplo é a limitação de múltiplas instâncias com a mesma credencial e a ausência de suporte completo para a simulação de combinações de teclas programáveis. Tais situações, embora não comprometam os objetivos iniciais, representam oportunidades relevantes para futuras melhorias.

Como perspectiva para trabalhos futuros, sugere-se ampliar a compatibilidade da biblioteca com diferentes versões do sistema legado, bem como a inclusão de um módulo para detecção automática de alterações estruturais no sistema que possam impactar o funcionamento das automações. Além disso, é possível que se evolua para uma estrutura de organização de usuários, permitindo que a própria biblioteca identifique quantos usuários semelhantes estão ativos na máquina e realize o gerenciamento adequado para o uso simultâneo. Também poderiam ser aplicadas técnicas de inteligência artificial para identificar padrões de interação do usuário com o sistema, otimizando ainda mais os fluxos de

automação.

Referências

BANCO CENTRAL DO BRASIL. **Cooperativismo de crédito no Brasil: história e desafios**. Brasília: Banco Central do Brasil, 2017.

BANCO CENTRAL DO BRASIL. **Levantamento sobre cooperativas de crédito: participação em municípios brasileiros**. 2024. Disponível em: https://www.bcb.gov.br/detalhenoticia/20247/noticia. Acesso em: 13 jun. 2025.

BANCO CENTRAL DO BRASIL. **O que é cooperativa de crédito?** Disponível em: https://www.bcb.gov.br/estabilidadefinanceira/cooperativacredito. Acesso em: 13 jun. 2025.

BANCO CENTRAL DO BRASIL. **Panorama do Sistema Nacional de Crédito Cooperativo 2023: crescimento e desafios**. Brasília: Banco Central do Brasil, 2024. Disponível em: https://www.bcb.gov.br/content/estabilidadefinanceira/coopcredpanorama/relatorio_panorama_cooperativas_2023_FINAL.pdf. Acesso em: 13 jun. 2025.

BANCO CENTRAL DO BRASIL. **Relatório de economia bancária 2021 – cooperativismo de crédito**. Brasília: Banco Central do Brasil, 2021. Disponível em: https://www.bcb.gov.br/publicacoes/relatorioeconomiabancaria. Acesso em: 18 jun. 2025.

Beiruth, A. X. O cooperativismo de crédito rural no Brasil: origens e impactos na inclusão financeira. **Revista de Economia e Sociologia Rural**, v. 52, n. 1, 2014.

BRASIL. Decreto n.º 10.474, de 26 de agosto de 2020. **Regulamenta a estrutura da Autoridade Nacional de Proteção de Dados (ANPD)**. Diário Oficial da União, Brasília, DF, 27 ago. 2020.

BRASIL. Lei n.º 13.709, de 14 de agosto de 2018. **Lei Geral de Proteção de Dados Pessoais (LGPD)**. Diário Oficial da União: seção 1, Brasília, DF, 15 ago. 2018. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709.htm. Acesso em: 17 jun. 2025.

Brito, M. J. et al. **Automação de processos e transformação digital no setor financeiro: impactos na eficiência e segurança. Revista de Administração e Inovação**, v. 18, n. 2, p. 85–98, 2021. Disponível em: https://doi.org/10.11606/rai. Acesso em: 18 jun. 2025.

Capgemini. **Intelligent automation: a game changer for business operations**. Capgemini Research Institute, 2016. Disponível em: https://www.capgemini.com/fr-fr/wp-content/uploads/sites/2/2020/06/A-game-changer-for-business-operations_2020.pdf. Acesso em: 13 jun. 2025.

Cavoukian, A. Cavoukian **Privacy by design: the 7 foundational principles**. Toronto: Information and Privacy Commissioner of Ontario, 2011. Disponível em: https://www.ipc.on.ca/wp-content/uploads/Resources/7foundationalprinciples.pdf. Acesso em: 10 jun. 2025.

CONFEBRAS – Confederação Brasileira das Cooperativas de Crédito. **História do cooperativismo de crédito no Brasil**. 2020. Disponível em: https://www.confebras.com.br. Acesso em: 18 jun. 2025.

DELOITTE. **Automação inteligente no setor financeiro: acelerando a transformação digital**. São Paulo: Deloitte Brasil, 2020. Disponível em: https://www2.deloitte.com/br/pt/pages/finance/articles/automacao-inteligente-no-setor-financeiro.html. Acesso em: 18 jun. 2025.

Doneda, D. **Da privacidade à proteção de dados pessoais**. 2. ed. Rio de Janeiro: Forense, 2019.

Ferreira, J. P.; ALMEIDA, R. Desenvolvimento de ferramentas low code para automação de processos corporativos. **Revista de Tecnologia e Sistemas**, v. 10, n. 2, p. 35–48, 2022.

Fowler, M. **Refatoração: aperfeiçoando o design de códigos existentes**. 2. ed. São Paulo: Novatec, 2019.

Gamma, E. et al. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 1995.

Lacity, M.; Willcocks, L. What knowledge workers stand to gain from automation. **Harvard Business Review**, 19 jun. 2015. Disponível em: https://hbr.org/2015/06/what-knowledge-workers-stand-to-gain-from-automation. Acesso em: 13 jun. 2025.

Lima, J. B. O papel do cooperativismo na inclusão social e no desenvolvimento regional. **Revista Pensamento & Realidade**, v. 34, n. 2, 2019.

Lutz, M. **Learning Python**. 5. ed. Sebastopol: O'Reilly Media, 2013.

Martin, R. C. **Clean code: a handbook of agile software craftsmanship**. Upper Saddle River: Prentice Hall, 2009.

Medeiros, L. A.; SANTOS, R. F. Desenvolvimento de bibliotecas de automação para sistemas legados: uma abordagem prática em instituições financeiras. **Revista Brasileira de Sistemas de Informação**, v. 17, n. 2, p. 45–60, 2021.

Mello, C. A. S.; SANTOS, M. **Automação de processos com robôs de software: fundamentos, aplicações e casos práticos**. São Paulo: Érica, 2020.

MICROSOFT. What is Windows Terminal?. **Microsoft Learn**, 2023. Disponível em: https://learn.microsoft.com/en-us/windows/terminal. Acesso em: 10 jun. 2025.

MICROSOFT DOCS. Overview of Windows Terminal features. **Microsoft Learn**, 2024. Disponível em: https://learn.microsoft.com/en-us/windows/terminal/features. Acesso em: 10 jun. 2025.

OCB – ORGANIZAÇÃO DAS COOPERATIVAS BRASILEIRAS. **Cooperativismo de crédito: desafios e perspectivas para a transformação digital**. Brasília: OCB, 2023. Disponível em: https://www.somoscooperativismo.coop.br. Acesso em: 18 jun. 2025.

OCB – ORGANIZAÇÃO DAS COOPERATIVAS BRASILEIRAS. **Panorama do cooperativismo de crédito**. Brasília: OCB, 2023. Disponível em: https://www.somoscooperativismo.coop.br. Acesso em: 18 jun. 2025.

OCB – ORGANIZAÇÃO DAS COOPERATIVAS BRASILEIRAS. **Relatório do cooperativismo brasileiro 2022**. Brasília: OCB, 2022. Disponível em: https://www.ocb.coop.br. Acesso em: 13 jun. 2025.

Oliveira, C. P. **Tecnologia e automação bancária**. 1997. Dissertação (Mestrado em Administração) – Fundação Getúlio Vargas, Rio de Janeiro. Disponível em: https://bibliotecadigital.fgv.br. Acesso em: 10 jun. 2025.

Pinheiro, P. P. **LGPD: comentada artigo por artigo**. São Paulo: Revista dos Tribunais, 2020.

PYCRYPTODOME. PyCryptodome: cryptographic library for Python. Disponível em: https://www.pycryptodome.org. Acesso em: 17 jun. 2025.

PYPA. Python Packaging User Guide. **Python Packaging Authority**, 2023. Disponível em: https://packaging.python.org. Acesso em: 10 jun. 2025.

PYTHON. os — Miscellaneous operating system interfaces. Disponível em: https://docs.python.org/3/library/os.html. Acesso em: 17 jun. 2025.

PYTHON. pywin32 – Python for Windows (Win32API and Win32CON). Disponível em: https://github.com/mhammond/pywin32. Acesso em: 17 jun. 2025.

PYTHON SOFTWARE FOUNDATION. Python Language Reference, version 3.13.11. Disponível em: https://www.python.org. Acesso em: 17 jun. 2025.

PYWINAUTO. Pywinauto: GUI Automation for Windows Applications. Disponível em: https://pywinauto.readthedocs.io. Acesso em: 17 jun. 2025.

Raiffeisen, F. W. **The credit unions: a German solution to rural poverty**. Mainz: Raiffeisen-Verlag, 1970.

RBS (GaúchaZH). Sítio histórico, instituições centenárias e a união como força motriz: um retrato do cooperativismo na Serra. Porto Alegre, 2022. Disponível em: https://gauchazh.clicrbs.com.br/pioneiro/economia/noticia/2022/07/sitio-historico-instituicoes-centenarias-e-a-uniao-como-forca-motriz-um-retrato-do-cooperativismo-na-serra-cl53yayi6000j019ipci25el3.html. Acesso em: 13 jun. 2025.

Ribeiro, M. A. Cooperativismo e desenvolvimento local: o caso das cooperativas de crédito rural no RS. **Revista de Administração Pública**, v. 45, n. 3, 2011.

Schneider, A. **Padre Theodor Amstad: apóstolo do cooperativismo**. Caxias do Sul: EDUCS, 2002.

SICREDI. **Relatório de sustentabilidade 2022**. Porto Alegre: Sicredi, 2022. Disponível em: https://www.sicredi.com.br/site/sobre-o-sicredi/relatorios. Acesso em: 18 jun. 2025.

Silva, M.; Pereira, L. Tecnologia e cooperativismo: desafios e oportunidades da transformação digital. **Revista Brasileira de Cooperativismo**, v. 8, n. 1, p. 15–28, 2021.

Sommerville, I. Sommerville **Engenharia de software**. 9. ed. São Paulo: Pearson, 2011.

Van Rossum, G.; Drake JR., F. L. **The Python language reference manual**. Network Theory Ltd., 2001.

WORLD COUNCIL OF CREDIT UNIONS. History. Disponível em: https://www.woccu.org/about/history. Acesso em: 13 jun. 2025.