

Aplicativo para Contagem de Macro-nutrientes e Dosagem de Insulina para Pessoas Diabéticas Implementado com Programação Orientada à Objeto em Python

1st João Pedro Santos

Escola de Engenharia UFMG

Universidade Federal de Minas Gerais

Belo horizonte, Brasil

joaopedrosantosdebrito@gmail.com

2nd Júlia Diniz

Escola de Engenharia UFMG

Universidade Federal de Minas Gerais

Belo Horizonte, Brasil

juliadinizrodrigues@gmail.com

3rd Ohana Souza

Escola de Engenharia UFMG

Universidade Federal de Minas Gerais

Belo Horizonte, Brasil

ohanasouza04@gmail.com

Abstract—Diante do crescente aumento no número de pessoas diagnosticadas com diabetes, torna-se essencial o desenvolvimento de recursos tecnológicos que atendam às necessidades específicas dessa parcela da população. Observa-se uma carência de aplicativos capazes de integrar, de maneira direta, o cálculo de carboidratos e o cálculo das doses de insulina, o que compromete a praticidade no manejo da doença. Para suprir essa necessidade, foi desenvolvido um aplicativo que combina essas funcionalidades de forma integrada e compacta, proporcionando uma experiência mais acessível e eficiente para o usuário. O aplicativo realiza o cálculo da dose de insulina necessária para cada refeição, avalia os carboidratos consumidos e armazena essas informações em um histórico, possibilitando acompanhamento e controle a longo prazo.

Index Terms—diabetes, insulina, aplicativo

I. INTRODUÇÃO

O projeto foi desenvolvido utilizando o paradigma de Programação Orientada a Objetos (POO), uma abordagem amplamente reconhecida por sua eficiência na organização de sistemas complexos. Baseando-se nos pilares de encapsulamento, herança, polimorfismo e abstração, conforme discutido em [1] e [2], o uso de POO permitiu criar um sistema modular, reutilizável e de fácil manutenção. Esses princípios organizam o código em componentes independentes, promovendo escalabilidade e simplificação no desenvolvimento e na manutenção do software.

A linguagem escolhida foi o Python, que oferece suporte nativo a todos os conceitos fundamentais de POO, além de uma sintaxe clara e intuitiva, conforme discutido em [1]. A interface gráfica do projeto foi implementada com o Tkinter, uma biblioteca oficial do Python. Criado por Fredrik Lundh como adaptador do toolkit Tcl/Tk, o Tkinter destaca-se por sua simplicidade e flexibilidade no desenvolvimento de interfaces gráficas [3].

O relatório será dividido nas seguintes seções: uma visão geral do projeto, a descrição dos requisitos e funcionalidades

implementadas, uma análise dos principais componentes do código, explicações detalhadas sobre o uso das bibliotecas e ferramentas, e, por fim, uma conclusão que resume as conquistas e as possibilidades de melhorias futuras.

II. ESTRUTURA DO CÓDIGO E APLICAÇÃO DE POO

A Programação Orientada a Objetos (POO) foi amplamente utilizada neste projeto para promover a modularidade, a reutilização de código e a organização estrutural do sistema. As classes foram projetadas para representar entidades do domínio do problema, como Usuário, Alimento, Refeição e diferentes tipos de Insulina, encapsulando dados e comportamentos relacionados. O polimorfismo foi aplicado para permitir que métodos como `calculaDosagem` fossem redefinidos de maneira específica em subclasses, enquanto a herança foi utilizada para compartilhar comportamentos e atributos comuns entre classes relacionadas. Além disso, o encapsulamento garantiu a proteção de dados sensíveis por meio do controle de acesso a atributos e métodos, enquanto a abstração simplificou a complexidade do sistema, expondo apenas as funcionalidades essenciais. Esses princípios de POO foram essenciais para construir um sistema flexível, escalável e de fácil manutenção.

A. Fluxograma

Figura 7

B. Bibliotecas

Abaixo estão as bibliotecas utilizadas no desenvolvimento do projeto.

- **Tkinter:** Biblioteca nativa da linguagem Python utilizada para o desenvolvimento da interface gráfica do projeto, proporcionando componentes visuais como botões, caixas de texto e janelas [3].
- **Datetime:** Biblioteca padrão do Python utilizada para manipulação de datas e horários, permitindo funcionalidades

como registro de timestamps, cálculo de diferenças de tempo e formatações específicas [4].

- **bcrypt:** Biblioteca utilizada para realizar a criptografia de senhas, garantindo a segurança dos dados sensíveis armazenados no sistema [5].
- **re:** Biblioteca padrão do Python utilizada para manipulação e busca de padrões em strings por meio de expressões regulares, essencial para validação e extração de dados [6].
- **csv:** Biblioteca padrão do Python usada para leitura e escrita de arquivos CSV (Comma-Separated Values), facilitando o gerenciamento de dados estruturados em formato de tabela [7].
- **sys:** Biblioteca padrão do Python que fornece acesso a variáveis e funções relacionadas ao ambiente de execução do programa, como manipulação de argumentos da linha de comando e controle do fluxo do sistema [8].
- **tkcalendar:** Extensão do Tkinter utilizada para criar widgets interativos de calendário, permitindo seleção de datas de forma visual na interface gráfica [9].
- **os:** Biblioteca padrão do Python usada para interagir com o sistema operacional, permitindo manipulação de arquivos e diretórios, além de obter informações sobre o ambiente [10].

C. Estrutura de Classes

Nesta seção, são descritas as principais classes desenvolvidas no projeto, com suas respectivas responsabilidades e métodos implementados, evidenciando a aplicação dos conceitos de Programação Orientada a Objetos.

- **Interface_Insulina:** A classe Interface_Insulina foi projetada como uma interface abstrata para padronizar os métodos relacionados aos diferentes tipos de insulina, definindo a estrutura básica para as classes específicas, como o método calculaDosagem, que deve ser obrigatoriamente implementado por suas subclasses. Por ser uma interface, ela utiliza conceitos fundamentais da Programação Orientada a Objetos (POO), como abstração, ao especificar apenas a estrutura de métodos sem implementar suas funcionalidades; polimorfismo, permitindo que diferentes subclasses forneçam implementações variadas para o método calculaDosagem, enquanto são tratadas de forma uniforme; e encapsulamento, garantindo consistência no comportamento das subclasses. Essa abordagem modular facilita a expansão do sistema, permitindo a adição de novos tipos de insulina sem a necessidade de modificar o código existente.
- **Tipos de Insulina:** As classes Asparge, Humalog, NPH e Glargina representam os diferentes tipos de insulina incluídos no aplicativo. Cada uma redefine o método calculaDosagem, incorporando os parâmetros específicos de cada insulina para o cálculo das doses. Esse comportamento demonstra o uso do polimorfismo,

pois o mesmo método, calculaDosagem, é implementado de forma distinta em cada classe, permitindo que cada tipo de insulina calcule as doses de acordo com suas próprias regras. Além disso, essas classes incluem um mecanismo para disparar um alarme caso a dose calculada exceda o limite máximo seguro, garantindo maior segurança para o usuário.

- **Nutrientes:** A classe Nutrientes foi desenvolvida para gerenciar o total de nutrientes acumulados de uma lista de alimentos. Ela disponibiliza métodos para adicionar os valores nutricionais correspondentes a diferentes alimentos e exibir o total acumulado de nutrientes consumidos, incluindo carboidratos, lipídios, proteínas e fibras. Além disso, a classe interage com o banco de dados para armazenar, recuperar e atualizar informações nutricionais de forma persistente. Por exemplo, ao adicionar um alimento, os valores nutricionais são atualizados tanto na memória quanto no banco de dados, garantindo que os dados permaneçam consistentes. Também é possível carregar os registros existentes do banco de dados ao iniciar a aplicação, permitindo que os cálculos de nutrientes comecem de um estado previamente salvo. Essa integração proporciona maior confiabilidade e persistência no gerenciamento de dados nutricionais.
- **Calculadora_Insulina:** A classe Calculadora_Insulina é responsável por determinar a dose correspondente para cada tipo de insulina, com base nos valores de entrada fornecidos pelo usuário. Ela utiliza o conceito de encapsulação para proteger os dados sensíveis e garantir que os cálculos sejam realizados de forma segura. Métodos como calcular_dose e configurar_alarme permitem abstrair a lógica de cálculo e o gerenciamento de alarmes, oferecendo uma interface simplificada ao usuário.
- **Calculadora_Nutrientes:** A classe Calculadora_Nutrientes calcula parâmetros importantes para o gerenciamento alimentar, como o gasto calórico diário, o metabolismo basal e a quantidade ideal de carboidratos, lipídios, proteínas e fibras. Ela emprega a abstração para encapsular os cálculos complexos relacionados à nutrição e organiza os dados fornecidos pelo usuário, como peso, altura, idade e nível de atividade física, em atributos. Métodos como calcula_gcd e calcula_macros permitem isolar e reutilizar a lógica de cálculo de maneira eficiente, promovendo flexibilidade e personalização.
- **Alimento** A classe Alimento foi projetada para gerenciar informações sobre alimentos e seus nutrientes. Seu principal objetivo é permitir que o usuário adicione alimentos com base em uma descrição e a quantidade em gramas, e calcule os nutrientes correspondentes a essa porção, utilizando dados armazenados no banco de dados.

- **Refeicao** A classe *Refeicao* é responsável por gerenciar um conjunto de alimentos que compõem uma refeição. Ela utiliza o conceito de agregação, permitindo que objetos da classe *Alimento* sejam adicionados e gerenciados dentro de uma refeição. Métodos como *adicionar_alimento* e *calcular_nutrientes_totais* abstraem as operações para somar os nutrientes dos alimentos incluídos. Essa abordagem promove modularidade, permitindo que refeições sejam compostas de maneira estruturada e organizada.
- **Historico** A classe *Historico* é uma interface base para as classes relacionadas ao registro de dados históricos, como alimentos e refeições. Ela faz uso de herança, fornecendo métodos e atributos que podem ser reutilizados ou sobrescritos pelas subclasses. Essa organização reduz a duplicação de código e facilita a implementação de funcionalidades específicas nas subclasses.
- **Historico_alimentos** A classe *Historico_Alimentos* herda de *Historico* e é responsável por gerenciar o registro e a consulta de dados relacionados ao histórico de alimentos consumidos por usuários, utilizando o banco de dados Supabase. Por meio do polimorfismo, ela pode implementar métodos personalizados para exibir ou armazenar dados de forma eficiente, enquanto reaproveita a funcionalidade básica fornecida pela classe base.
- **Historico_Refeicao** Derivada da classe base *Historico*, a classe *Historico_Refeicao* gerencia o armazenamento e a exibição de registros de refeições e alimentos em um banco de dados Supabase. Ela emprega o polimorfismo para adaptar métodos herdados e criar funcionalidades específicas, como a adição de novas refeições ao histórico e a exibição de registros com base em intervalos de datas fornecidos pelo usuário. Além disso, utiliza o conceito de associação, interagindo diretamente com objetos das classes *Refeicao* e *Alimento*.
- **Usuario:** A classe *Usuario* tem como principal objetivo gerenciar a criação e autenticação de usuários no sistema. Utilizando atributos privados, ela protege informações sensíveis, como senha e e-mail, enquanto métodos como *insere_usuario* e *autenticacao_usuario* implementam a lógica de validação e verificação, garantindo segurança e exclusividade. O uso de abstração permite que a complexidade do processo de autenticação seja encapsulada, facilitando a interação do usuário com o sistema.
- **Perfil Medico:** A classe *PerfilMedico* é responsável por gerenciar informações relacionadas ao perfil médico do usuário. Essa classe armazena dados relevantes,

como tipo de diabetes, doses padrão de insulina, e recomendações nutricionais específicas no banco de dados Supabase. O perfil médico é utilizado para personalizar as funcionalidades do aplicativo, garantindo que cada usuário tenha acesso a uma experiência adaptada às suas necessidades individuais. A classe *PerfilMedico* é responsável por gerenciar informações relacionadas ao perfil médico do usuário. Essa classe armazena dados relevantes, como tipo de diabetes, doses padrão de insulina, e recomendações nutricionais específicas no banco de dados Supabase. O perfil médico é utilizado para personalizar as funcionalidades do aplicativo, garantindo que cada usuário tenha acesso a uma experiência adaptada às suas necessidades individuais.

- **Verificadora** A classe *Verificadora* possui como principal funcionalidade validar se um valor fornecido pode ser convertido para um tipo numérico específico, como float ou int. Para isso, ela utiliza o método estático *verificar_inteiro*, que tenta realizar a conversão e retorna True em caso de sucesso ou False se ocorrer um erro. O uso do método estático reflete o conceito de encapsulamento da Programação Orientada a Objetos (POO), pois centraliza e organiza a lógica de validação dentro de uma estrutura clara e reutilizável, sem a necessidade de instanciar a classe. Além disso, ao ser estático, o método pode ser acessado diretamente pela classe, reforçando a modularidade e a eficiência no desenvolvimento.

D. Diagrama UML

Figura 8

E. Usuario

A classe *Usuario* gerencia informações como nome, e-mail, senha e preferências dos usuários, implementando funcionalidades essenciais, como registro, login e armazenamento seguro de dados. Utiliza encapsulamento para proteger atributos sensíveis, como senhas, que são criptografadas com *bcrypt*. Também verifica duplicações no banco de dados para garantir a integridade. Essa abordagem modular e orientada a objetos promove segurança, organização e facilita a expansão do sistema, exemplificando os conceitos de encapsulamento e abstração.

Algorithm 1 Algoritmo de autenticação e inserção de usuário.

```

0: procedure AUTENTICAÇÃO DE USUÁRIO(email, senha_inserida)
0:   response  $\leftarrow$  consultar tabela "Usuarios"
0:   por senha com email
0:   if response.data  $\neq \emptyset$  then
0:     senha_armazenada  $\leftarrow$  response.data[0]['senha']
0:     if bcrypt.checkpw(senha_inserida, senha_armazenada)
0:       then
0:         Imprimir "Login realizado com sucesso!"
0:         Retornar True
0:       else
0:         Imprimir "Senha Inválida"
0:         Retornar False
0:     else
0:       Imprimir "Usuário não encontrado!"
0:       Retornar False
0: procedure INSERIR USUÁRIO(email, senha)
0:   padrao_email  $\leftarrow$  regex para validação de email
0:   if não email combina com padrao_email then
0:     Imprimir "Formato de e-mail inválido."
0:     Retornar
0:   response  $\leftarrow$  consultar tabela "Usuarios"
0:   por email
0:   if response.data  $\neq \emptyset$  then
0:     Imprimir "Erro: Este email já está em uso."
0:     Retornar
0:   salt  $\leftarrow$  bcrypt.gensalt()
0:   hashed  $\leftarrow$  bcrypt.hashpw(senha, salt)
0:   hashed_senha  $\leftarrow$  hashed.decode("utf-8")
0:   response  $\leftarrow$  inserir email e hashed_senha
0:   na tabela "Usuarios"
0:   if response.data  $\neq \emptyset$  then
0:     Imprimir "Usuário cadastrado com sucesso!"
0:     Retornar True
0:   else if response.error  $\neq \emptyset$  then
0:     Imprimir "Erro cadastrando o usuário: response.error.message"
0:     Retornar False
0:   else
0:     Imprimir "Erro desconhecido."
0:     Retornar False
0:   =0

```

F. Insulina

As classes de insulinas no projeto representam diferentes tipos, como Asparge, Humalog, NPH e Glargina, e modelam suas características específicas para calcular doses de forma personalizada. Baseadas em uma classe abstrata comum, essas classes implementam o método calculaDosagem com regras adaptadas a cada tipo de insulina. O uso de polimorfismo permite que o sistema trabalhe com diferentes tipos de insulina de maneira uniforme, enquanto mecanismos de segurança validam as doses e alertam sobre limites perigosos. Essa abordagem garante flexibilidade, segurança e organização no gerenciamento do tratamento de diabetes.

III. BASE DE DADOS

O esquema de banco de dados do aplicativo foi desenvolvido utilizando o Supabase, uma plataforma que permite criar bancos de dados PostgreSQL em nuvem de forma rápida e acessível, além de incluir funcionalidades como autenticação, armazenamento e APIs em tempo real, tudo de forma gratuita e com uma interface intuitiva para manipulação e integração. O Supabase também disponibiliza uma biblioteca oficial para Python, que facilita a execução de queries, inserções e outras operações no banco de dados diretamente do código. Essa biblioteca simplifica o desenvolvimento ao permitir que as

Algorithm 2 Classe Asparge (Insulina de Ação Rápida)

```

0: procedure CALCULA DOSAGEM(peso, tipo_diabetes,
0:   dosagem_max, carboidratos, proteínas) {Calcula a
0:   dose de insulina necessária para a refeição}
0:   if tipo_diabetes = "Tipo1" then
0:     tdd  $\leftarrow$  peso  $\times$  0.55
0:   else if tipo_diabetes = "Tipo2" then
0:     tdd  $\leftarrow$  peso  $\times$  0.3
0:   else if tipo_diabetes = "Pr - diabetes" then
0:     tdd  $\leftarrow$  peso  $\times$  0.1
0:   else if tipo_diabetes = "Gestacional" then
0:     tdd  $\leftarrow$  peso  $\times$  0.6
0:   else
0:     Imprimir "Tipo de diabetes desconhecido."
0:     Retornar 0
0:   ic  $\leftarrow$  500/tdd
0:   glicose_proteina  $\leftarrow$  proteinas  $\times$  0.15
0:   carboidratos_totais  $\leftarrow$  carboidratos +
0:   glicose_proteina
0:   dose_insulina  $\leftarrow$  carboidratos_totais/ic
0:   dose_final  $\leftarrow$  min(dose_insulina, dosagem_max)
0:   Retornar round(dose_final, 6)
0: procedure VERIFICA ALARME(dose_calculada,
0:   dosagem_maxima) {Chama o método base para
0:   verificar alarmes de dosagem}
0:   Chamar super.VerificaAlarme(dose_calculada,
0:   dosagem_maxima)
0:   =0

```

interações com o banco de dados sejam feitas por meio de comandos de alta abstração, integrando eficientemente os dados com a lógica da aplicação.

Foram criadas 6 tabelas de referência para organizar e normalizar os dados fundamentais do sistema:

Alimentos: Contém dados retirados da Tabela Brasileira de Composição de Alimentos (TACO), desenvolvida pela Universidade Estadual de Campinas (UNICAMP) em colaboração com o Ministério da Saúde, é a principal referência nacional para dados nutricionais de alimentos. Ela fornece informações detalhadas sobre a composição de centenas de alimentos, incluindo conteúdo de macronutrientes, vitaminas, minerais e fibras. Sua utilização garante que o aplicativo trabalhe com dados confiáveis e atualizados, ajustados à realidade brasileira.

Atividades físicas: Lista os níveis de atividade física com suas respectivas chaves de identificação (id).

Refeicao: Lista cada tipo de refeição com suas respectivas chaves de identificação (id).

Sexos: Lista cada sexo com suas respectivas chaves de identificação (id).

Tipos diabetes: Lista cada tipo de diabetes disponível no aplicativo com suas respectivas chaves de identificação (id).

Tipos insulina: Lista cada tipo de insulina disponível no aplicativo com suas respectivas chaves de identificação (id).

Para permitir a inserção e o gerenciamento de dados pelos

usuários, foram criadas 4 tabelas adicionais:

Usuários: Armazena informações dos usuários, incluindo emails e senhas. As senhas são armazenadas utilizando criptografia hash, garantindo maior segurança dos dados sensíveis.

Perfil médico: Reúne informações médicas dos usuários, como peso, altura, tipo de diabetes, nível de atividade física, tipo de insulina (quando aplicável) e outras informações relevantes para o cálculo de nutrientes e insulina.

Historico: Armazena o histórico de refeições salvas pelos usuários, com informações de nutrientes totais e insulina calculada, se aplicável.

Historico Alimentos: Armazena o histórico de alimentos salvos pelos usuários, com informações de nutrientes totais.

O banco de dados foi projetado seguindo o princípio da 4ª Forma Normal (4NF), que garante a eliminação de dependências multivaloradas, assegurando que cada dado seja armazenado de maneira singular e sem redundância. Essa abordagem é essencial para evitar inconsistências, melhorar a eficiência e facilitar a manutenção.

IV. RESULTADOS E INTERFACE GRÁFICA

O desenvolvimento do aplicativo apresentou resultados expressivos, oferecendo uma solução funcional e intuitiva para o público diabético e também para usuários que desejam monitorar as propriedades nutricionais dos alimentos consumidos. Ele calcula de forma precisa a dose de insulina ideal com base nos alimentos ingeridos em cada refeição e fornece informações detalhadas sobre macronutrientes, como fibras e carboidratos. Além disso, permite ao usuário configurar uma dosagem máxima de insulina por refeição, ajudando a evitar riscos associados à subdosagem ou hiperdosagem, como hipoglicemia e hiperglicemia. Um recurso adicional relevante é o alerta emitido quando o usuário registra uma ingestão de carboidratos muito alta, o que possibilita ajustar a refeição ou monitorar a glicemia com mais cuidado após a aplicação.

Um dos aspectos mais positivos do aplicativo é sua acessibilidade. A interface foi desenvolvida com foco na simplicidade e objetividade, tornando-o utilizável por pessoas de diferentes idades e níveis de experiência com tecnologia. O design claro, com botões intuitivos e mensagens diretas, assegura que tanto crianças quanto pessoas mais velhas possam utilizá-lo com facilidade, ampliando sua aplicabilidade e impacto positivo.

No âmbito técnico, o projeto permitiu a aplicação prática de conceitos avançados de Programação Orientada a Objetos (POO), como encapsulamento, polimorfismo e tratamento de exceções. Ferramentas como tkinter e tkcalendar foram essenciais para a criação de uma interface gráfica funcional, enquanto a integração com o banco de dados Supabase possibilitou a gestão eficiente das informações do usuário, histórico de refeições e alimentos consumidos. O uso de bibliotecas como Pillow (PIL) aprimorou a apresentação visual do aplicativo, e módulos personalizados, como Calculadora_Insulina e as classes específicas de insulina (e.g., Asparge, Humalog, Glargina, entre outras), garantiram cálculos precisos e suporte especializado para diferentes perfis de diabetes.

Um dos pontos que precisam ser aprimorados no aplicativo está relacionado ao cadastro de perfis médicos para usuários que não tomam insulina. Atualmente, essa configuração pode gerar alguns bugs durante a utilização. Contudo, optou-se por manter essa funcionalidade ativa para permitir futuras melhorias e adaptações, com o objetivo de tornar o aplicativo ainda mais inclusivo para diferentes perfis de usuários.

Embora o aplicativo já seja funcional e apresente bons resultados, algumas melhorias poderiam ser implementadas para torná-lo ainda mais completo. Entre elas, destacam-se:

- A possibilidade de editar o perfil médico cadastrado;
- A funcionalidade para excluir alimentos ou refeições do histórico;
- A inclusão de filtros no histórico para exibir macronutrientes específicos conforme a necessidade do usuário;
- A integração com mais tipos de insulina e a possibilidade de gerenciar o uso de múltiplas insulinas simultaneamente.

A. Contexto geral do aplicativo

No cenário atual, há uma escassez de recursos tecnológicos acessíveis e eficientes para o gerenciamento de condições crônicas como o diabetes. Aplicativos existentes no mercado, como Glic [11], mySugr [12] e Minsulin [13], muitas vezes não atendem às necessidades de todas as faixas etárias ou níveis de conhecimento tecnológico. Além disso, esses aplicativos frequentemente não fornecem informações detalhadas sobre os dados nutricionais dos alimentos consumidos, limitando a capacidade dos usuários de monitorar completamente sua dieta.

Nosso aplicativo foi desenvolvido com uma proposta diferenciada, utilizando bibliotecas gratuitas e de fácil acesso, focando não apenas no cálculo preciso da dose de insulina, mas também na oferta de dados nutricionais detalhados dos alimentos ingeridos. Isso permite um acompanhamento mais completo e personalizado para os usuários, facilitando o controle e o gerenciamento do diabetes de forma acessível e eficiente para todos.

V. CONCLUSÃO

O desenvolvimento do aplicativo demonstrou resultados significativos, consolidando-se como uma solução inovadora e acessível para o gerenciamento do diabetes e o monitoramento das propriedades nutricionais dos alimentos. A combinação de uma interface intuitiva com funcionalidades técnicas avançadas garante precisão no cálculo de doses de insulina e fornece dados detalhados sobre macronutrientes, tornando-o uma ferramenta prática e eficiente para diferentes perfis de usuários.

Além disso, o projeto destacou a relevância da aplicação de conceitos de Programação Orientada a Objetos e integração com ferramentas modernas, como o banco de dados Supabase e bibliotecas Python específicas, para a criação de soluções tecnológicas robustas. Apesar de algumas limitações, como a necessidade de aprimorar o suporte para perfis médicos sem

insulina, o aplicativo se apresenta como um sistema funcional e adaptável, com um grande potencial de evolução.

Com as melhorias planejadas, como a edição de perfis médicos, a exclusão de itens do histórico e a inclusão de filtros nutricionais, o aplicativo pode se consolidar ainda mais como uma referência no segmento. Ao abordar as lacunas identificadas em soluções já existentes no mercado, ele se posiciona como uma ferramenta acessível e eficiente para atender às necessidades de uma ampla gama de usuários, promovendo bem-estar e autonomia no gerenciamento do diabetes. O projeto foi desenvolvido com base em exemplos práticos disponíveis em um repositório GitHub.¹

REFERENCES

- [1] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.
- [2] C. Larman, *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Pearson Education India, 2005.
- [3] P. S. Foundation, *Tkinter: GUI Programming in Python*, Documentação oficial do Python, 2025, disponível em: <https://docs.python.org/3/library/tkinter.html>.
- [4] —, *Datetime: Basic Date and Time Types*, Documentação oficial do Python, 2025, disponível em: <https://docs.python.org/3/library/datetime.html>.
- [5] D. M. et al., *bcrypt: Modern password hashing for your software*, PyPI - Python Package Index, 2025, disponível em: <https://pypi.org/project/bcrypt/>.
- [6] P. S. Foundation, *re: Regular Expression Operations*, Documentação oficial do Python, 2025, disponível em: <https://docs.python.org/3/library/re.html>.
- [7] —, *csv: CSV File Reading and Writing*, Documentação oficial do Python, 2025, disponível em: <https://docs.python.org/3/library/csv.html>.
- [8] —, *sys: System-specific Parameters and Functions*, Documentação oficial do Python, 2025, disponível em: <https://docs.python.org/3/library/sys.html>.
- [9] J. Malard, *tkcalendar: Calendar and DateEntry widgets for Tkinter*, PyPI - Python Package Index, 2025, disponível em: <https://pypi.org/project/tkcalendar/>.
- [10] P. S. Foundation, *os: Miscellaneous Operating System Interfaces*, Documentação oficial do Python, 2025, disponível em: <https://docs.python.org/3/library/os.html>.
- [11] Glic, “Glic - controle de diabetes,” Aplicativo, versão 1.0, 2025, disponível em: <https://www.gliconline.com.br>. Acesso em: 14 jan. 2025.
- [12] mySugr, “mysugr - gerenciador de diabetes,” Aplicativo, versão 3.0, 2025, disponível em: <https://mysugr.com>. Acesso em: 14 jan. 2025.
- [13] Minsulin, “Minsulin - gerenciamento de insulina,” Aplicativo, versão 2.5, 2025, disponível em: <https://minsulin.com.br>. Acesso em: 14 jan. 2025.

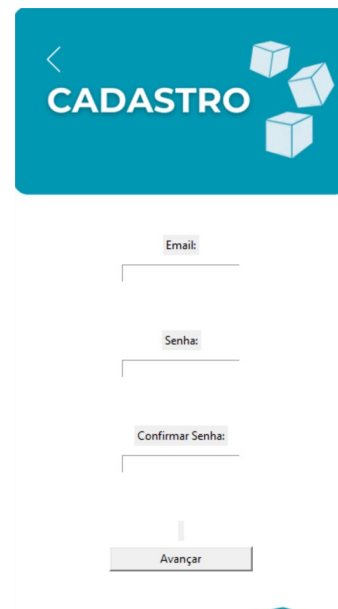


Fig. 1. Tela de cadastro.

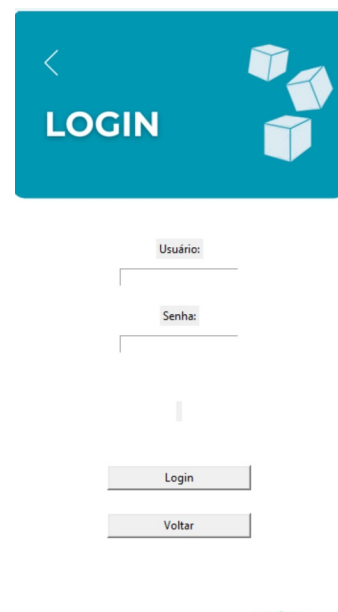


Fig. 2. Tela de login.

¹Repositório disponível em: <https://github.com/Ohana-Souza/POO.git>.

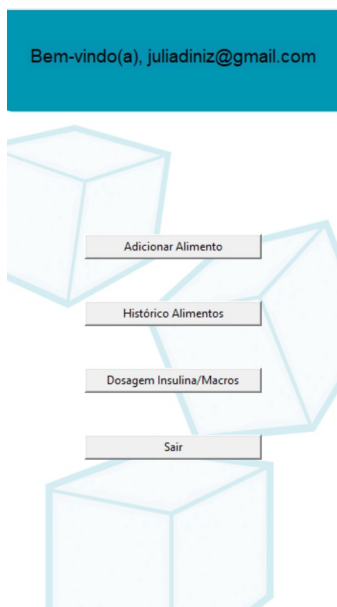


Fig. 3. Tela inicial.

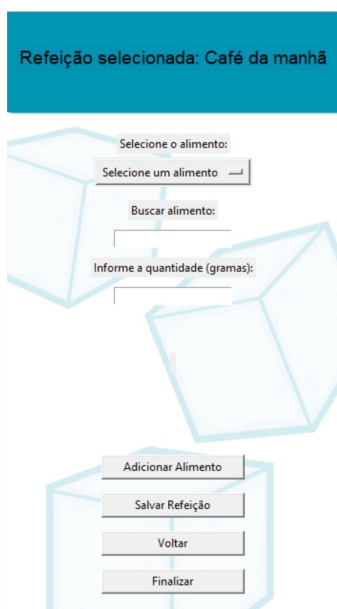


Fig. 4. Tela de seleção de refeição/alimento.

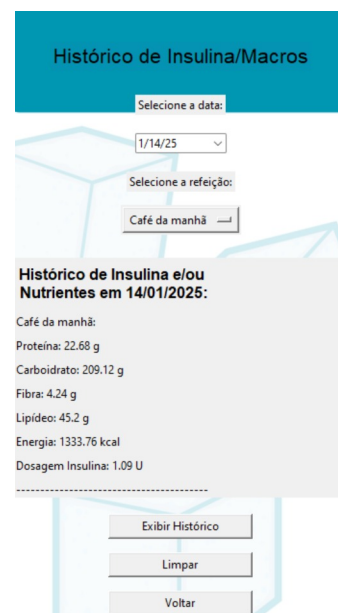


Fig. 5. Tela do histórico.

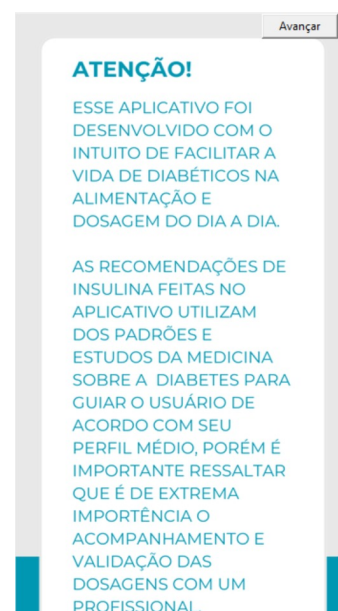


Fig. 6. Tela de aviso.

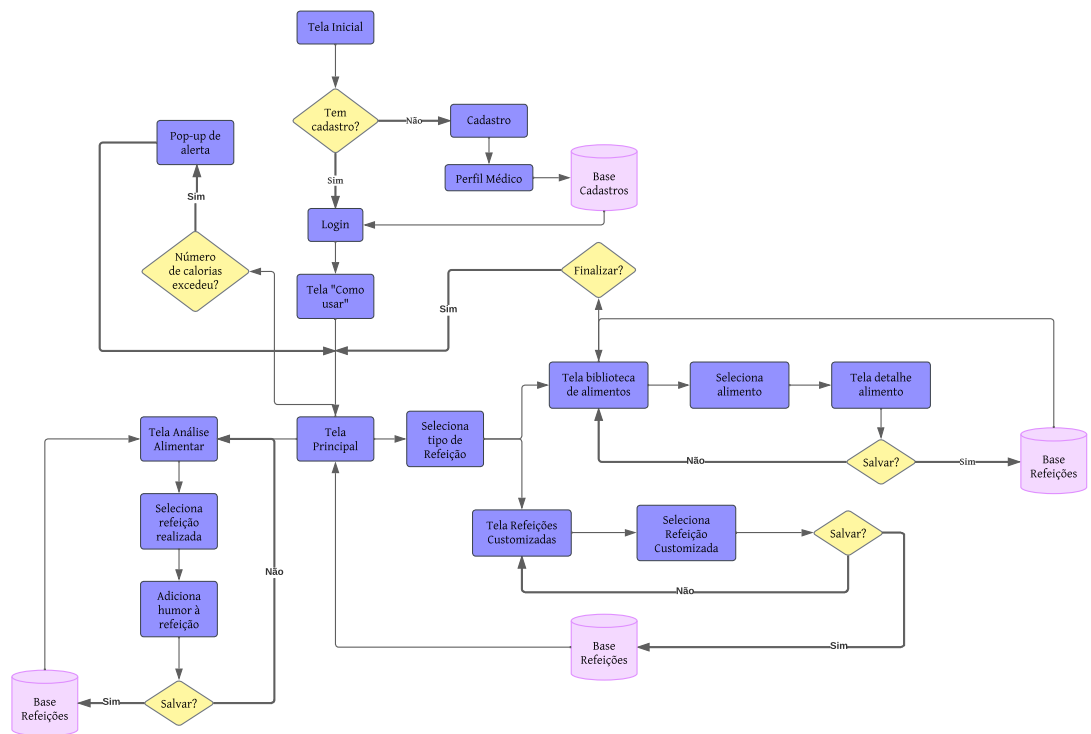


Fig. 7. Fluxuograma.

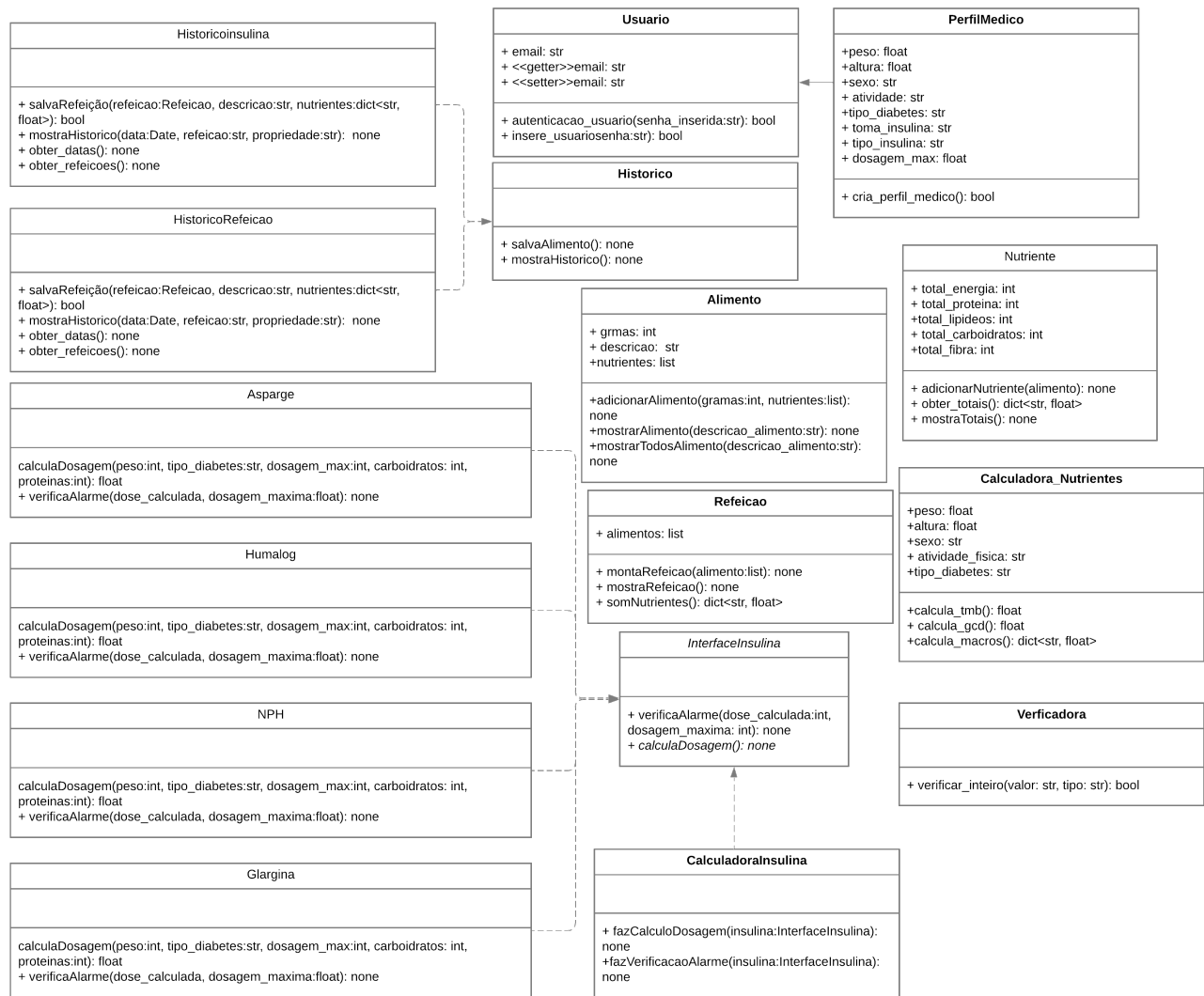


Fig. 8. Diagrama UML do projeto