Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

HIGOR DOS SANTOS MARTINS

APLICATIVO MÓVEL PARA AUXILIAR NA PRÁTICA DE EXERCÍCIOS FÍSICOS.

Assis/SP
2024

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

HIGOR DOS SANTOS MARTINS

APLICATIVO MÓVEL PARA AUXILIAR NA PRÁTICA DE EXERCÍCIOS FÍSICOS

Projeto de pesquisa apresentado ao Curso de Análise e Desenvolvimento de Sistemas do Instituto Municipal de Ensino Superior de Assis – IMESA e à Fundação Educacional do Município de Assis – FEMA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando(a): Higor dos Santos Martins
Orientador(a): Me. Guilherme de Cleva Farto

Assis/SP
2024

AGRADECIMENTOS

Gostaria de iniciar agradecendo meu grande amigo, Ozeas Carvalho, que foi quem me direcionou para ingressar na área de tecnologia e, que além disso, me incentiva e me ajuda progredir.

Também agradeço a minha esposa, Érika Martins, e minha filhas, Johanna e Clarissa, por me darem suporte e ânimo durante todo o processo da minha graduação e para além. Ainda no âmbito familiar, agradeço a minha mãe, Sirlene Martins, por todo apoio, muitas vezes, até financeiro para que eu realizasse o curso.

Por fim, agradeço ao corpo docente da FEMA, em especial, à coordenadora Diomara Barros e ao meu professor orientador Guilherme de Cleva Farto, por todo o período da graduação, todos os ensinamentos e pelo suporte na realização deste trabalho.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

RESUMO

Fornecer maneiras que incentivem uma melhor qualidade de vida é algo de suma importância. Desse modo, o presente trabalho, motivado pelo alto uso de aparelhos móveis da população brasileira, propõe o desenvolvimento de uma aplicação móvel híbrida, de arquitetura Cliente-Servidor, implementando o padrão de comunicação REST. O Cliente será desenvolvido utilizando o framework Flutter da linguagem Dart, que é mantida pela Google, a escolha do Flutter é justificada por sua natureza hibrida, que otimiza o desenvolvimento e manutenção de aplicações multiplataformas. Enquanto do lado do servidor, onde será contida as regras de negócio e comunicação com o banco de dados, o Postgresql - uma opção SGBD de modelo relacional - será implementado utilizando o framework Phoenix da linguagem Elixir, que utiliza o paradigma funcional, uma abordagem muito interessante para a construção de APIs. O aplicativo traz consigo a proposta de ser um facilitador na rotina de treinos do usuário, possibilitando que o mesmo possa criar rotinas de treinos, definir metas a serem alcançadas, preparar treinos. Tudo isso com o objetivo de auxiliar e motivar os usuários a adotarem um estilo de vida mais ativo e saudável.

Palavras-chave: Aplicativo móvel híbrido, API-Rest, Cliente-Servidor, Flutter, Dart, Elixir, Phoenix, atividade física, qualidade de vida.

ABSTRACT

Providing ways to encourage a better quality of life is of utmost importance. Therefore, this work, motivated by the high use of mobile devices by the Brazilian population, proposes the development of a hybrid mobile application with a Client-Server architecture, implementing the REST communication pattern. The Client will be developed using the Flutter framework of the Dart language, maintained by Google. The choice of Flutter is justified by its hybrid nature, which optimizes the development and maintenance of cross-platform applications. On the server side, where business rules and communication with the database will be handled, PostgreSQL—a relational database management system (RDBMS)—will be implemented using the Phoenix framework of the Elixir language, which employs the functional paradigm, a very interesting approach for building APIs. The application aims to be a facilitator in the user's training routine, allowing the user to create training routines, set goals to be achieved, and prepare training sessions. All of this is intended to assist and motivate users to adopt a more active and healthy lifestyle.

Keywords: Hybrid mobile application, REST API, Client-Server, Flutter, Dart, Elixir, Phoenix, physical activity, quality of life.

1.INTRODUÇÃO

O sedentarismo, considerado como o mal do século, está associado a doenças crônicas que resultam em uma mortalidade aumentada (BOTH e CHAKRAVARTHY, 2002). Sendo a quarta principal causa de morte no mundo, indivíduos pouco ativos apresentam risco de 20% a 30% maior de morte em comparação a indivíduos fisicamente ativos (STEIN, 2019).

Ademais, o sedentarismo aliado a maus hábitos nutricionais representam o principal fator de risco para o desenvolvimento da obesidade (PEREIRA, 2003). O que por sua vez, resulta na perda de qualidade de vida e adoecimento da população, segundo relatório divulgado pela OMS (Organização Mundial da Saúde) em 2022, cerca de 500 milhões de pessoas podem desenvolver doenças não transmissíveis tais como: doenças cardíacas, obesidade, diabetes, depressão, ansiedade, entre outras, devido à inatividade física.

Com o advento da tecnologia a realização de atividades físicas em períodos de descanso do trabalho ou da escola tem sido substituída por entretenimento eletrônico, sendo essa, uma das causas da obesidade e do sedentarismo, é sabido que existem outros fatores que levam a essa condição. Contudo, destacam-se causas preponderantes do desenvolvimento da obesidade e sedentarismo, os maus hábitos alimentares e a falta de atividade física (WANNMACHER, 2016).

Desse modo, a introdução da prática de atividades físicas na rotina de um indivíduo é uma ferramenta poderosa para a prevenção e reversão da condição de obesidade e sedentarismo e para ser considerada uma pessoa fisicamente ativa é necessária a

realização de ao menos, 150 minutos de atividades físicas semanais, que podem ser divididas em pelo menos 3 dias na semana e com seções mínimas de 30 minutos (PISA, 2017).

Por conta disso, esse projeto visa desenvolver uma aplicação utilizando o framework da linguagem Dart chamado Flutter, que será responsável pelo Front End da aplicação, esse Front End irá consumir uma API Rest, desenvolvida em Elixir que será responsável pelas regras de negócio e a interação com a base de dados da aplicação. Essa aplicação tem como principal função auxiliar o usuário na preparação e realização de exercícios físicos, sendo assim, uma ferramenta para auxiliar no combate à obesidade e ao sedentarismo.

1.1. OBJETIVO GERAL

Desenvolver um aplicativo móvel em *Flutter* e *Elixir* que auxilie o usuário a desenvolver e executar planos de treino, sejam em academias ou ao ar livre.

Estudo da tecnologia *Flutter* e *Elixir*;

Pesquisa de usabilidade e acessibilidade para implementação do projeto;

Análise e modelagem do aplicativo proposto;

Desenvolvimento do aplicativo proposto;

Testes/validação das funcionalidades implementadas;

1.2. PÚBLICO ALVO

O público-alvo deste projeto é composto por pessoas que desejam inserir na sua rotina a prática de atividade física, seja ela realizada em academias ou ao ar livre em locais como praças, parques ou até mesmo em casa.

Um dos problemas enfrentados por esse público é a falta de motivação para realizar exercícios, muitas vezes dada pela dificuldade de elaborar treinos estimulantes. Nesse sentido, o projeto busca oferecer uma alternativa para o planejamento e execução de treinos através de um app que seja intuitivo e atrativo aos usuários, a fim de usar um aparelho eletrônico, que por muitas vezes é um fator que estimula o sedentarismo, como uma ferramenta auxiliar na prática de atividades físicas.

1.3. JUSTIFICATIVA

Os canais e influenciadores de prática de atividades físicas, principalmente voltadas ao universo da musculação, estão ganhando grande notoriedade nos dias atuais no Brasil e no Mundo. Um exemplo disso foram as lives que transmitiram o evento mais importante do meio do fisiculturismo, denominado Mister Olympia, que tiveram milhões de espectadores.

No Brasil, influenciadores do meio como Renato Cariani estão alcançando milhões de pessoas e, por consequência, motivando muitas delas a começarem a se exercitar. Porém, nem sempre é fácil elaborar uma rotina de treinos, mesmo para aqueles que frequentam uma academia, pois, no modelo da maioria das academias brasileiras, nem sempre existe o acompanhamento devido de um personal e para aqueles que treinam em casa é ainda mais difícil. Esse pode ser um fator que acaba desestimulando e fazendo com que o indivíduo abandone a prática de atividades físicas.

Por outro lado, dispositivos móveis são amplamente utilizados nos dias de hoje, o Brasil tem 242 milhões de celulares inteligentes em uso no país, segundo um levantamento divulgado pela FGV, são mais aparelhos que habitantes e além disso o brasileiro passa em média 5.4 horas utilizando o aparelho, com aponta uma matéria do G1.

Devido a esses fatores, o presente trabalho propõe o desenvolvimento de uma aplicação que se aproveite do alto tempo que o brasileiro passa no smartphone e o auxilie na preparação e execução do treinamento. A aplicação irá utilizar Flutter - que é um framework que possibilita que o build da aplicação seja realizado tanto para Iphones quanto para Android, o que facilita o alcance de público e torna mais ágil o desenvolvimento - para desenvolver o Front End, que vai consumir uma API Rest, desenvolvida em Elixir - uma linguagem de paradigma funcional, escalável e com um suave de aprendizado.

2.MÉTODO DE DESENVOLVIMENTO

Durante todo o desenvolvimento do projeto, serão utilizados sites, cursos, artigos e vídeos tutoriais como ferramenta de auxílio, para utilização das ferramentas escolhidas. Além disso, no início do projeto será realizada um levantamento, análise e validação de

requisitos, a fim identificar necessidades e lapidar as ideias a serem desenvolvidas. Essa etapa será o alicerce para o desenvolvimento do projeto.

Continuando, serão criados os diagramas UML. Diagrama de caso de Uso, Diagrama de Classe e Diagrama de Atividades, essa é a etapa de modelagem do projeto e espera-se que ao final dela seja possível ter uma visão clara da arquitetura e funcionamento da aplicação.

Posteriormente, será iniciada a fase de desenvolvimento, norteada pela etapa passada. Utilizando a arquitetura de cliente-servidor, será realizado o desenvolvimento de uma aplicação móvel híbrida desenvolvida na linguagem Dart, utilizando o seu framework Flutter, que efetuará o papel de cliente na arquitetura. Já do lado servidor será desenvolvida uma API em Elixir, utilizando o framework Phoenix, onde será contida as regras de negócio da aplicação e persistência dos dados através do banco de dados relacional PostgreSQL. Por fim, o padrão REST realizará as trocas de mensagens entre Cliente e Servidor.

2.1. ARQUITETURA CLIENTE-SERVIDOR

Uma arquitetura Cliente-Servidor pode ser compreendida como um modelo computacional baseado em rede onde uma aplicação é dividida em dois processos distintos executados separadamente de forma independente, (OSÓRIO, 2023). Nesse cenário, o processo cliente efetua requisições ao servidor e aguarda que as mesmas sejam respondidas. Já o processo servidor processa as requisições recebidas e envia as respostas ao processo cliente, (Hura, 1995), essa forma de comunicação entre cliente servidor é denominado de *Request Response* e um dos protocolos mais comumente utilizados para que elas sejam implementadas é o protocolo HTTP da pilha TCP/IP, (Berners-Lee, 1996).

2.2. REST

REST (*REpresentation State Transfer*) é um estilo arquitetônico que fornece padrões que facilitam a comunicação entre sistemas *Web*. Sistemas compatíveis com REST são denominados RESTful, que são uma coleção de serviços REST, onde cada um é responsável por gerenciar determinados recursos, (SEGURA, 2018). O uso de recursos ao invés de comandos adicionam uma natureza apátrida aos sistemas que o

implementam para troca de informação, o que significa que o servidor não necessita de informações sobre a implementação e estado do cliente e vice-versa, para que haja a troca de mensagens. O que possibilita que multiplos clientes possam solicitar recursos a um servidor como por exemplo aplicações *web*, aplicações móveis nativas e, como será feito no caso deste trabalho, aplicações móveis híbridas.

2.3. APLICAÇÕES MÓVEIS HÍBRIDAS

Aplicações móveis híbridas, são aplicações capazes de transformar o que seria o desenvolvimento de um *web* site, em uma aplicação móvel com capacidade de ser executada em múltiplas plataformas, tais como Android e iOS, (Coelho, 2015). Pela a sua capacidade de ser executada em diferentes plataformas através do desenvolvimento de uma única aplicação, o desenvolvimento híbrido tem um custo operacional mais barato e uma maior facilidade de manutenção do código em relação ao desenvolvimento de aplicações móveis nativas, sendo uma ótima opção para o desenvolvimento de aplicações que não demandam muita interação com recursos nativos das plataformas, (Madureira, 2017). Entretanto, para aplicações que dependem de uma maior performance o desenvolvimento nativo é uma melhor opção, embora tenha um custo mais elevado devido a necessidade de se utilizar uma linguagem específica para cada plataforma (Ramires, 2017). Devido às facilidades do desenvolvimento híbrido muitas ferramentas têm ganho popularidade entre desenvolvedores, entre elas pode-se destacar React Native, Ionic e Flutter.

2.4. FLUTTER

Flutter, que segundo o seu site oficial, é um *framework open source* da linguagem Dart para a construção de aplicativos bonitos, compilados nativamente e multiplataforma a partir de uma única base de código. A arquitetura em camadas do Flutter dá a você controle sobre cada pixel na tela e seus poderosos recursos de composição permitem que você sobreponha e anime gráficos, vídeo, texto e controles sem limitação. O Flutter inclui um conjunto completo de widgets que entregam experiências pixel-perfeitas, tanto para a construção de aplicações IOS quanto para aplicações Android.

2.5. ELIXIR

Elixir é considerada uma linguagem funcional, por aplicar o paradigma funcional. Esse tipo de paradigma é fortemente ligado ao conceito de funções matemáticas e adicionando ao código características como:

- Funções Puras, que são funções que não tem o seu estado alterado, garantindo que sempre o mesmo resultado se as entradas não forem alteradas, ou seja, elas possuem uma natureza determinística.
- Funções de primeira ordem: uma função de primeira ordem é uma função que pode ser passada como entrada de uma outra função, ou até mesmo, uma função que é retorno de outra.
- Imutabilidade: ela garante que o valor de um objeto não será alterado depois da sua criação, na o lingual o sinal “=” não é um sinal de atribuição e sim de *matching.*

Além disso, a linguagem é compilada para ser executada sobre a máquina virtual da linguagem Erlang, uma linguagem poderosa muito utilizada para lidar com as redes de telefonia, adicionando a linguagem Elixir a capacidade de ser tolerante a falhas, facilmente escalável, a tornando uma linguagem preparada para a construção de sistemas robustos.

2.6. POSTGRESQL

Uma ferramenta poderosa e open-source, o PostgreSQL é um banco de dados relacional robusto que permite usar linguagem SQL para manipulação das tabelas e de fácil conexão com qualquer *API*, a escolha dessa tecnologia se deu por sua alta confiabilidade e segurança.

3.ANÁLISE E ESPECIFICAÇÕES DO SISTEMA

Antes do início do desenvolvimento, para que se tenha uma melhor eficiência e otimização do tempo é importante que seja realizada a etapa de análise e especificações do sistema. Ela consiste na estruturação do projeto que resulta na representação do

sistema gerando a compreensão do mesmo, antes que seja iniciada a fase de desenvolvimento. A UML (*Unified Modeling Language*), é uma linguagem que padroniza a criação das estruturas de projetos através da elaboração de diversos diagramas que são divididos em três grupos: diagramas comportamentais, diagramas estruturais e diagramas de interação. Neste trabalho será utilizado o diagrama de caso de uso, para se obter uma visualização dos requisitos do sistema.

3.1. DIAGRAMA DE CASO DE USO

O diagrama de caso de uso é um diagrama comportamental e é um dos primeiros diagramas a serem elaborados na criação de um projeto, pois tem a função descrever de forma gráfica quais funcionalidades do software atenderão a quais usuários, sem preocupar em como elas serão implementadas (VASQUEZ, 2016). Desse modo, será elaborado um diagrama de caso de uso, para que se tenha uma visão das funcionalidades do sistemas e de que modo elas interagem.

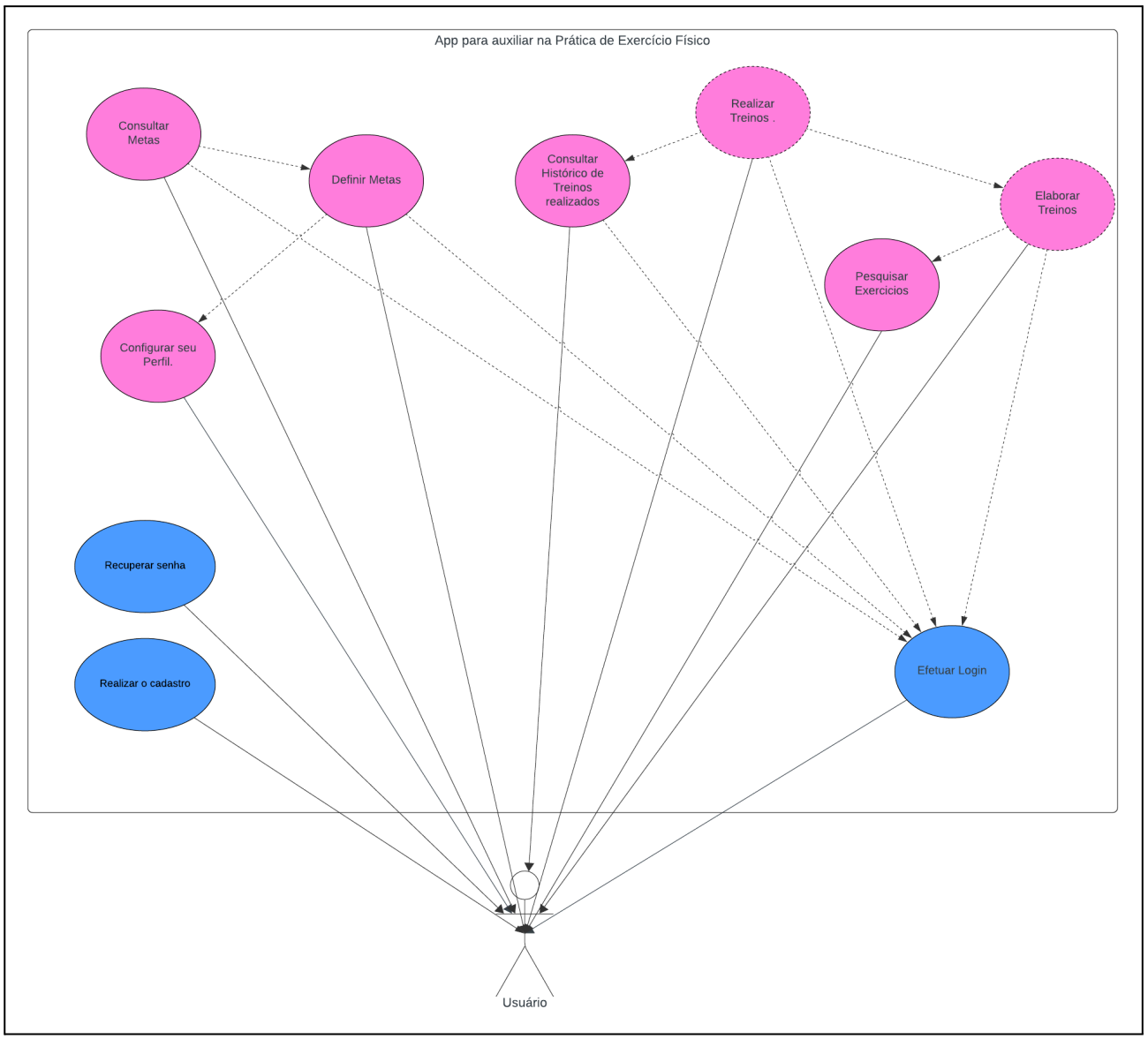


Figura 1: Diagrama de Caso de Uso Geral

3.2.MAPA MENTAL

Implementação de um mapa mental para ilustrar a estrutura das entidades e como elas irão se relacionar para o melhor funcionamento do sistema.

Figura 2: Mapa Mental das Entidades do Sistema

3.3. DIAGRAMA DE SEQUÊNCIA

Para elucidar o funcionamento de cada caso de uso, serão elaborados diagramas de sequência, este é um diagrama comportamento de iteração e tem o objetivo de descrever como acontece a interação do usuário com o sistema.

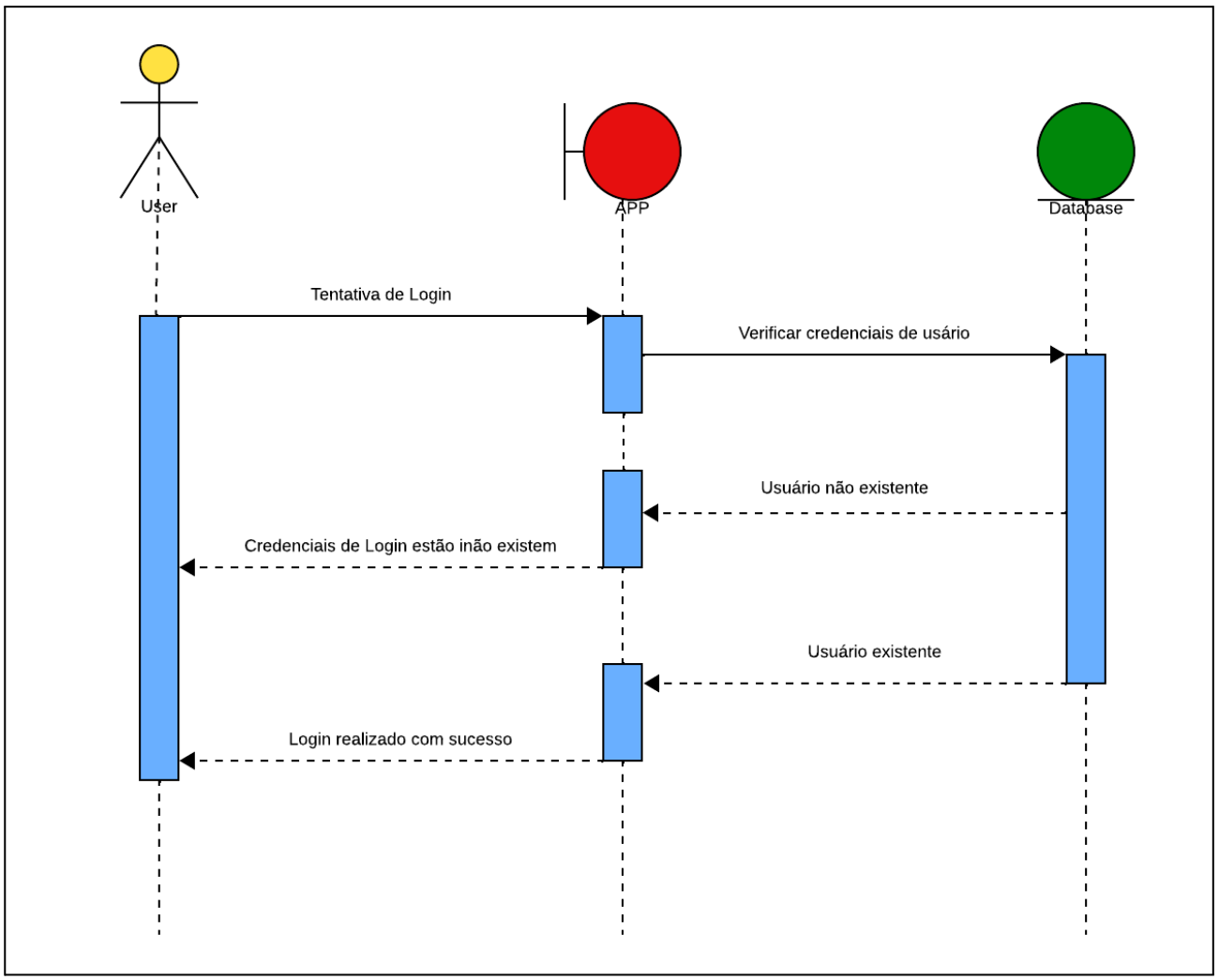


Figura 3: Diagrama de Sequência - Login

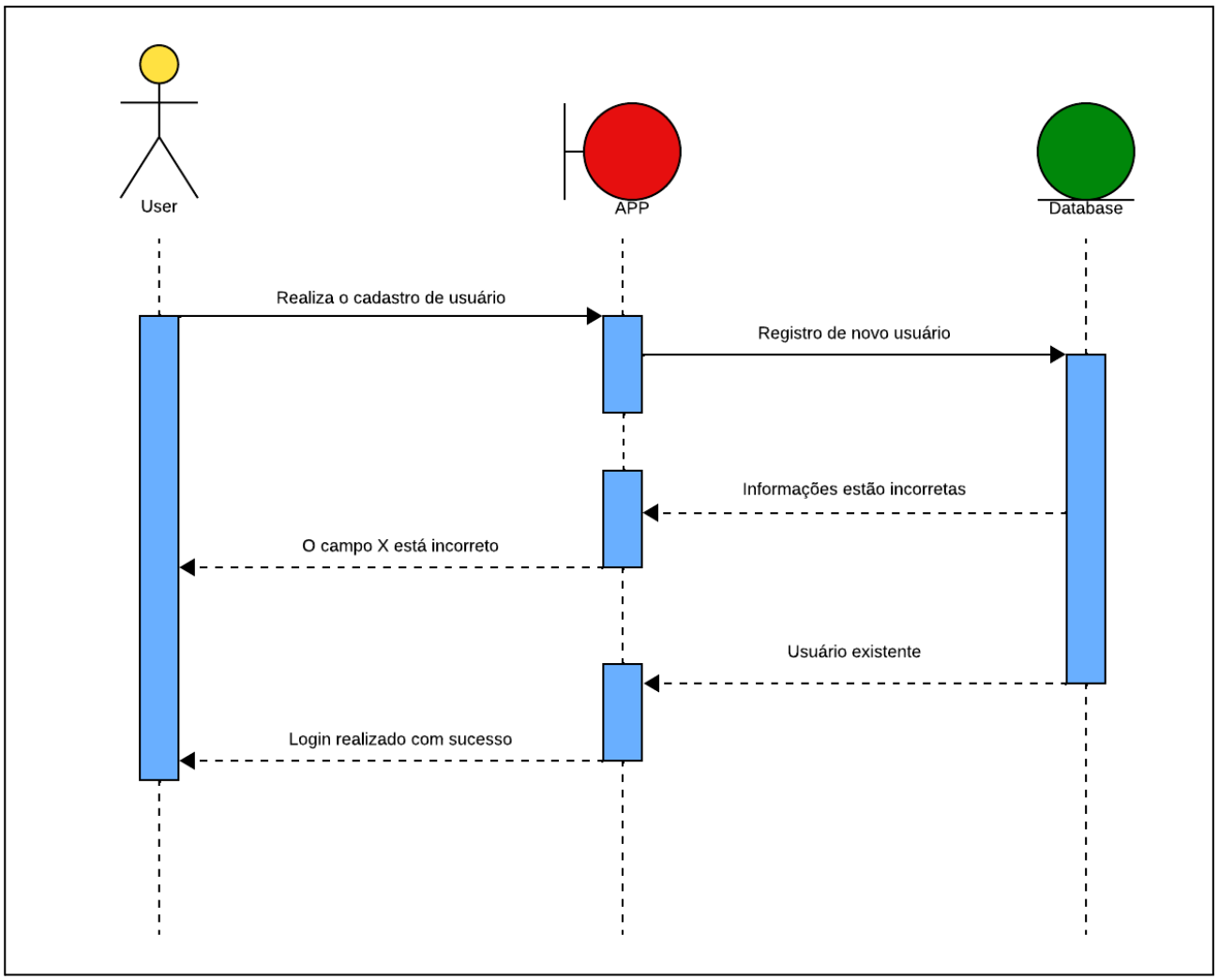


Figura 4: Diagrama de Sequência - Cadastro de Novo Usuário

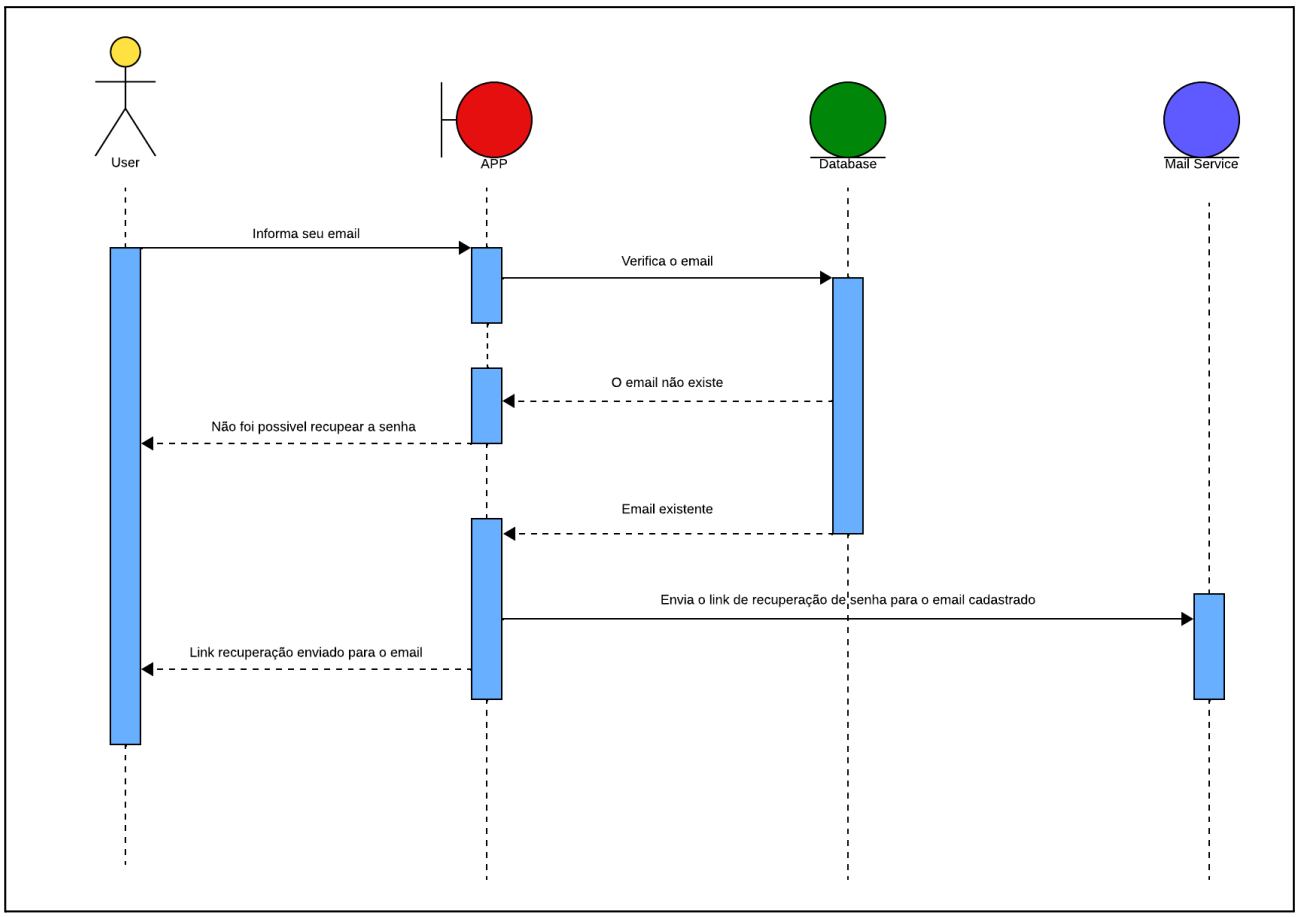


Figura 5: Diagrama de Sequência - Recuperar Senha

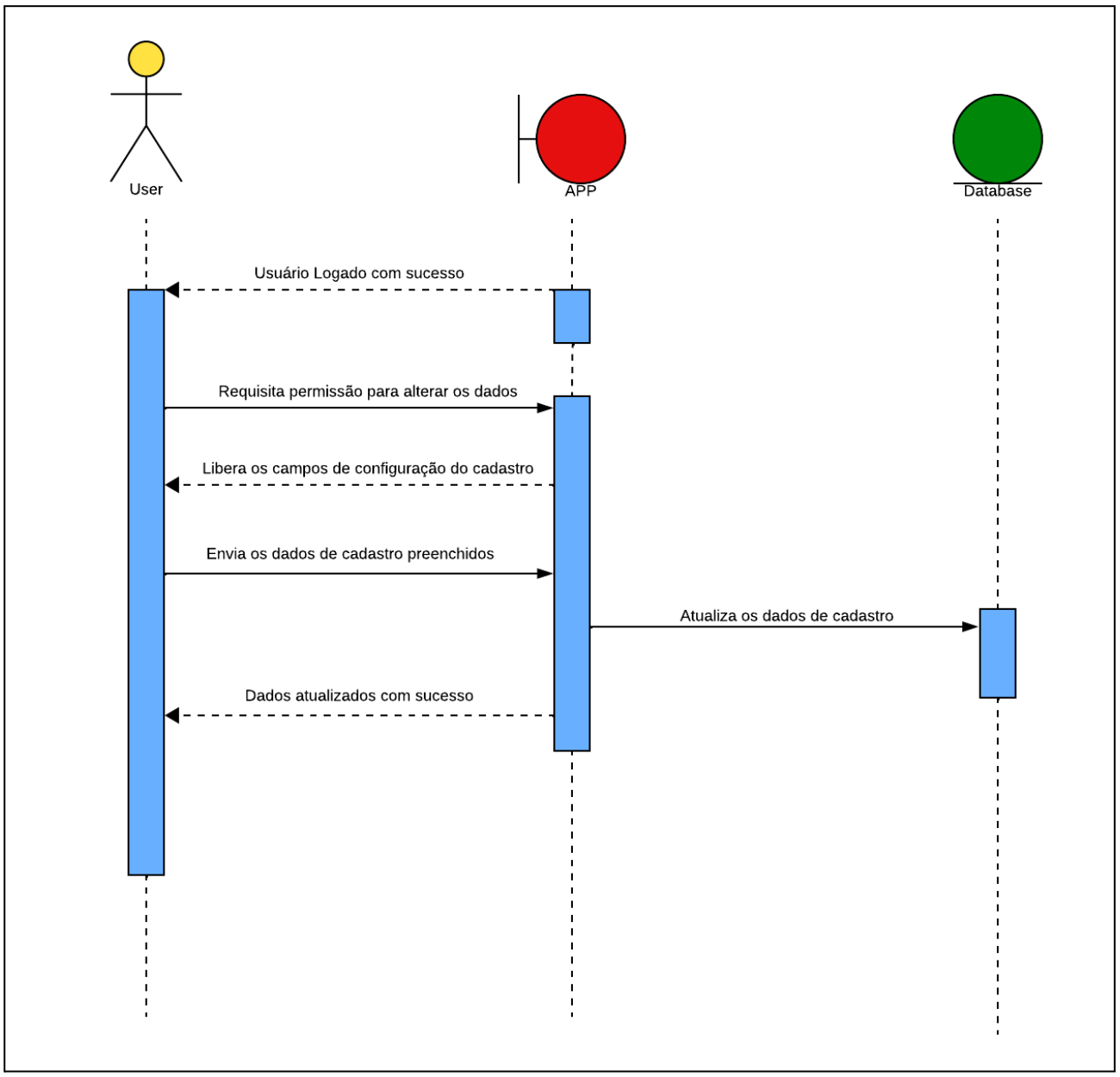


Figura 6: Diagrama de Sequência - Configurar Perfil de Usuário

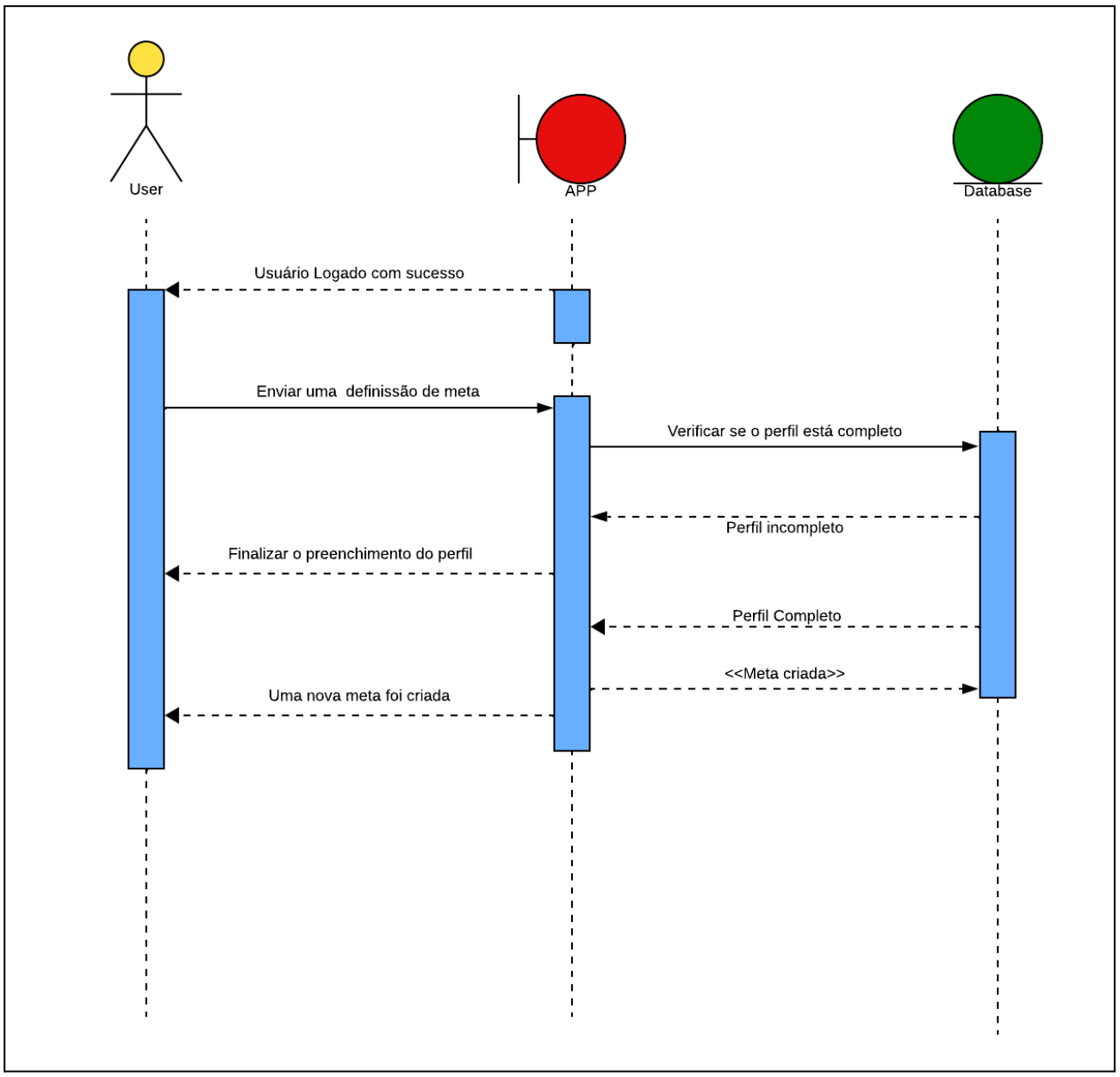


Figura 7: Diagrama de Sequência - Definir Metas

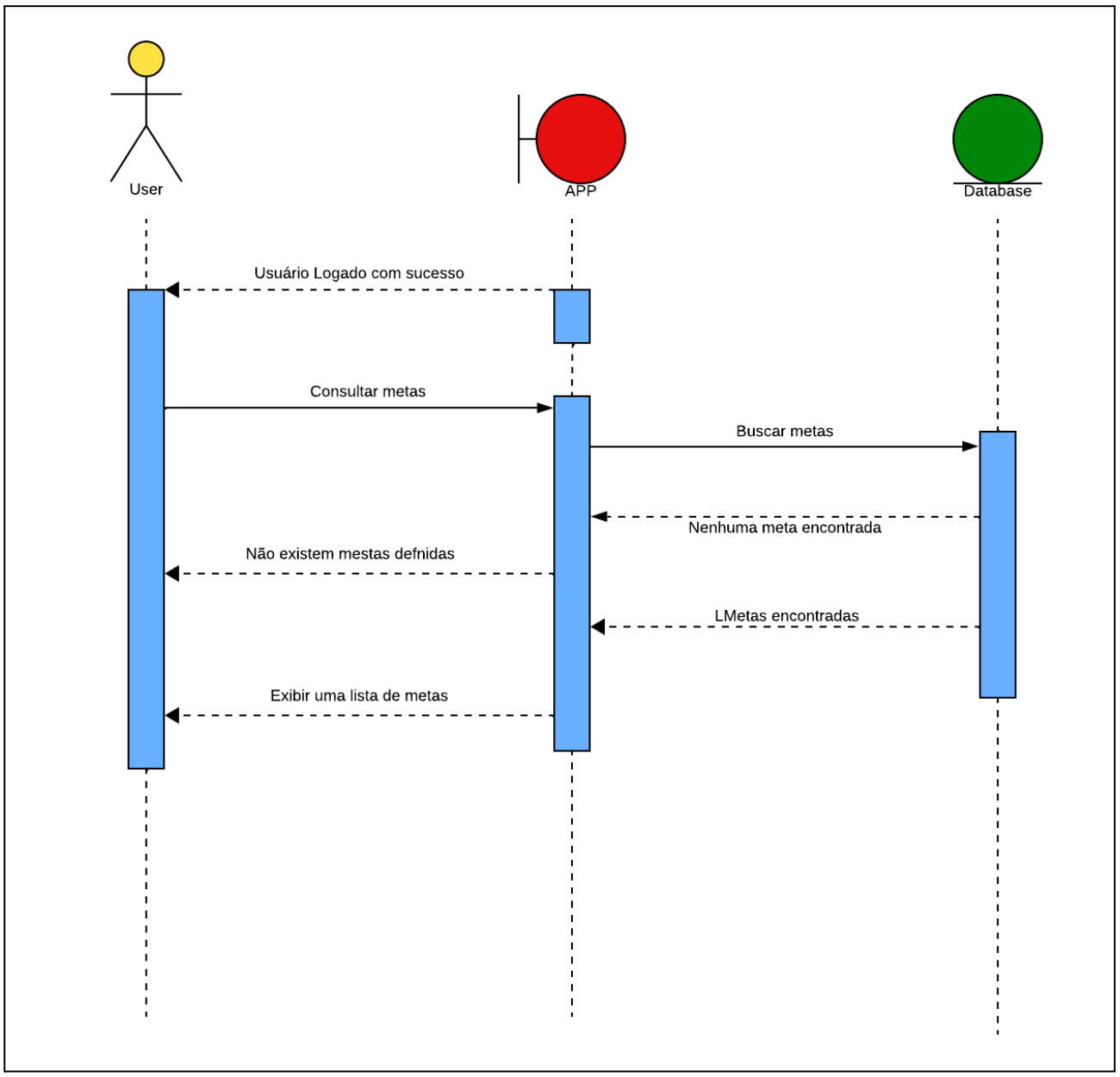


Figura 8: Diagrama de Sequência - Consultar Metas

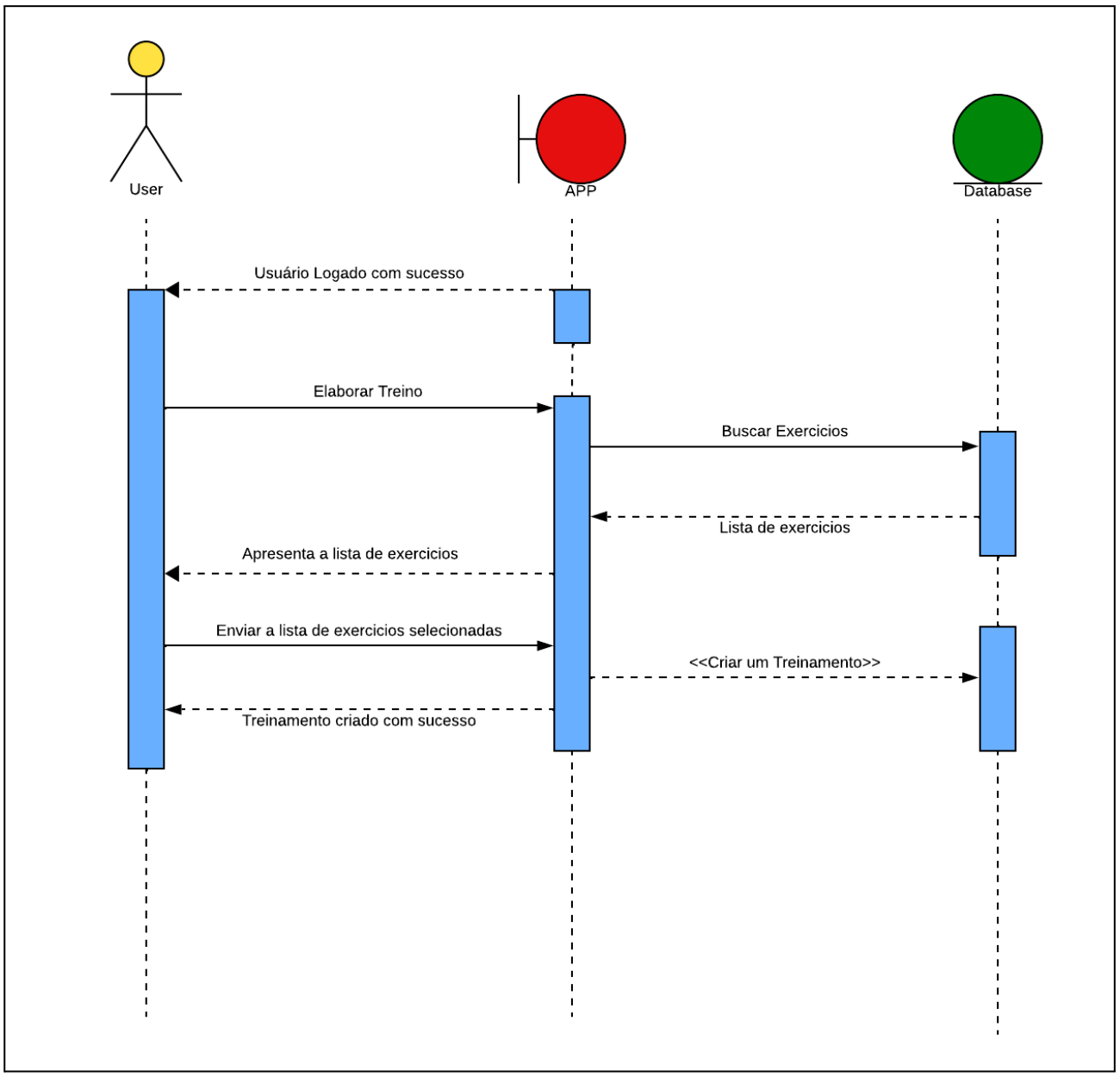


Figura 9: Diagrama de Sequência - Preparar Treinamento

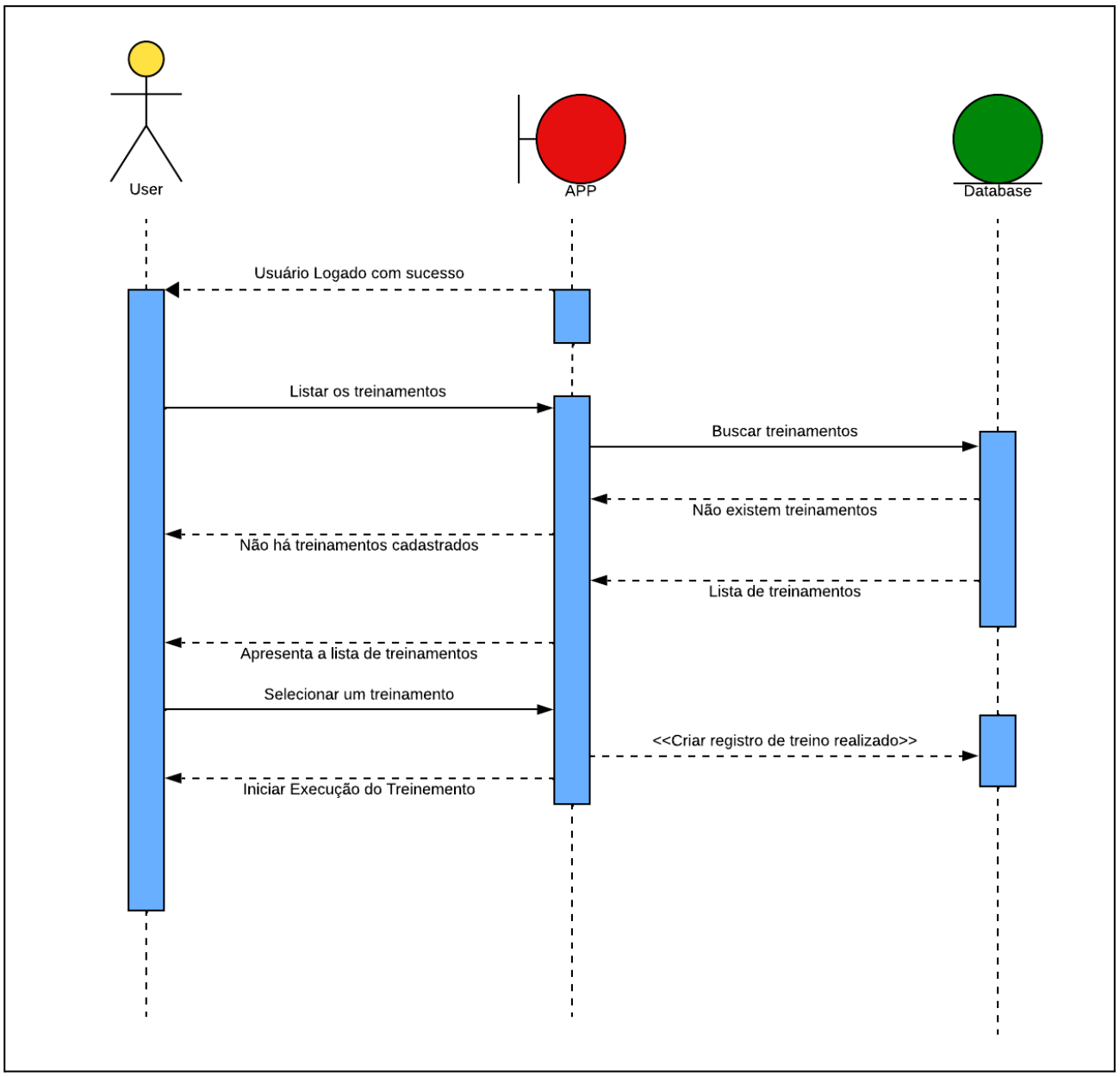


Figura 10: Diagrama de Sequência - Realizar um Treinamento

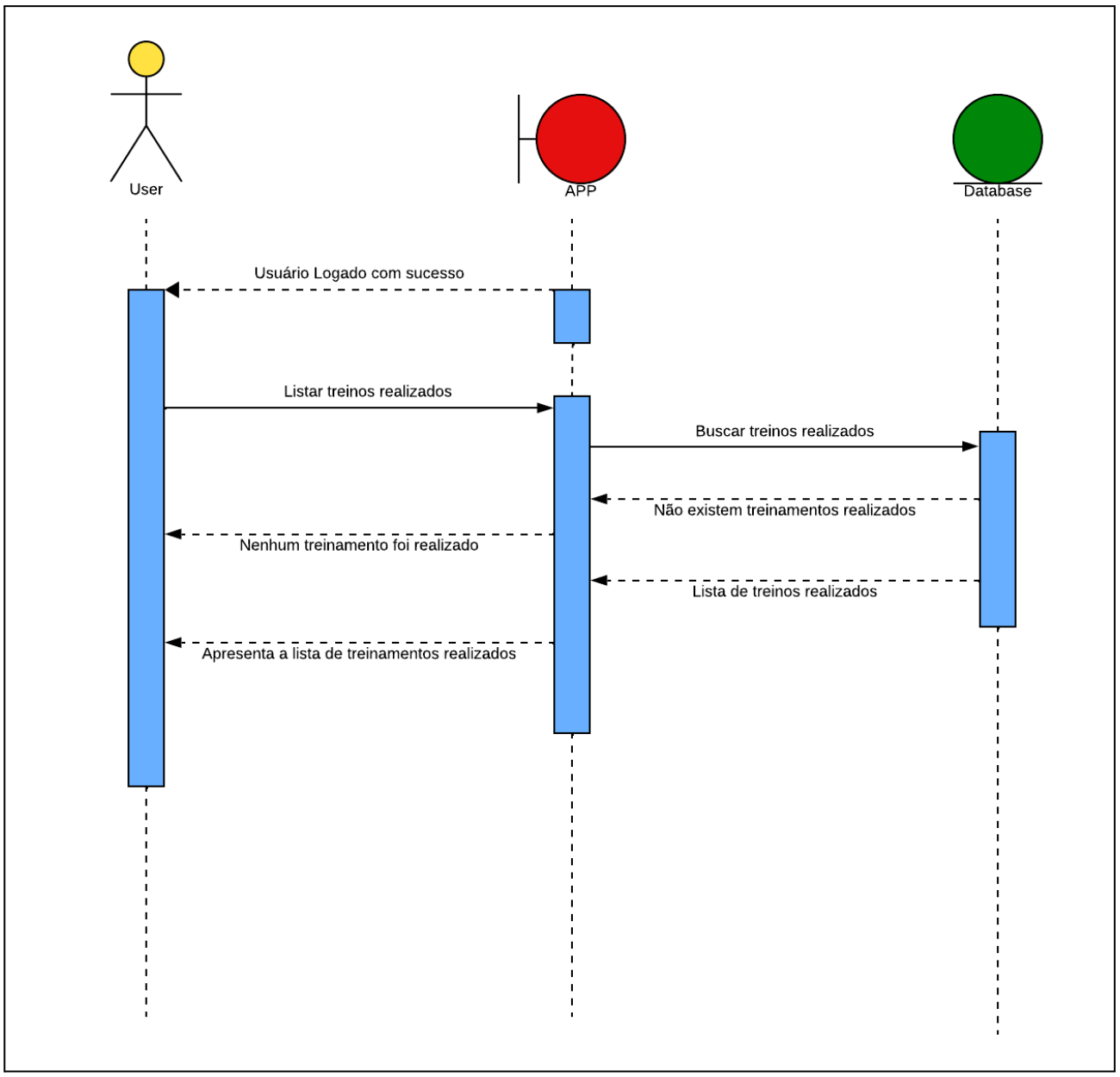


Figura 11: Diagrama de Sequência - Consultar Histórico de Treinamentos Realizados

3.4. DIAGRAMA DE ENTIDADE RELACIONAMENTO

A fim de, criar uma representação da estrutura lógica do banco de dados. Descrevendo suas entidades, que podem ser entidades fortes e fracas, os atributos dessas entidades e como elas se relacionam, foram desenvolvidos dois diagramas de

entidade-relacionamento. Eles têm a função de projetar e documentar a organização dos dados, auxiliando na compreensão da forma como as entidades se conectam e facilitando a criação do banco de dados da aplicação.

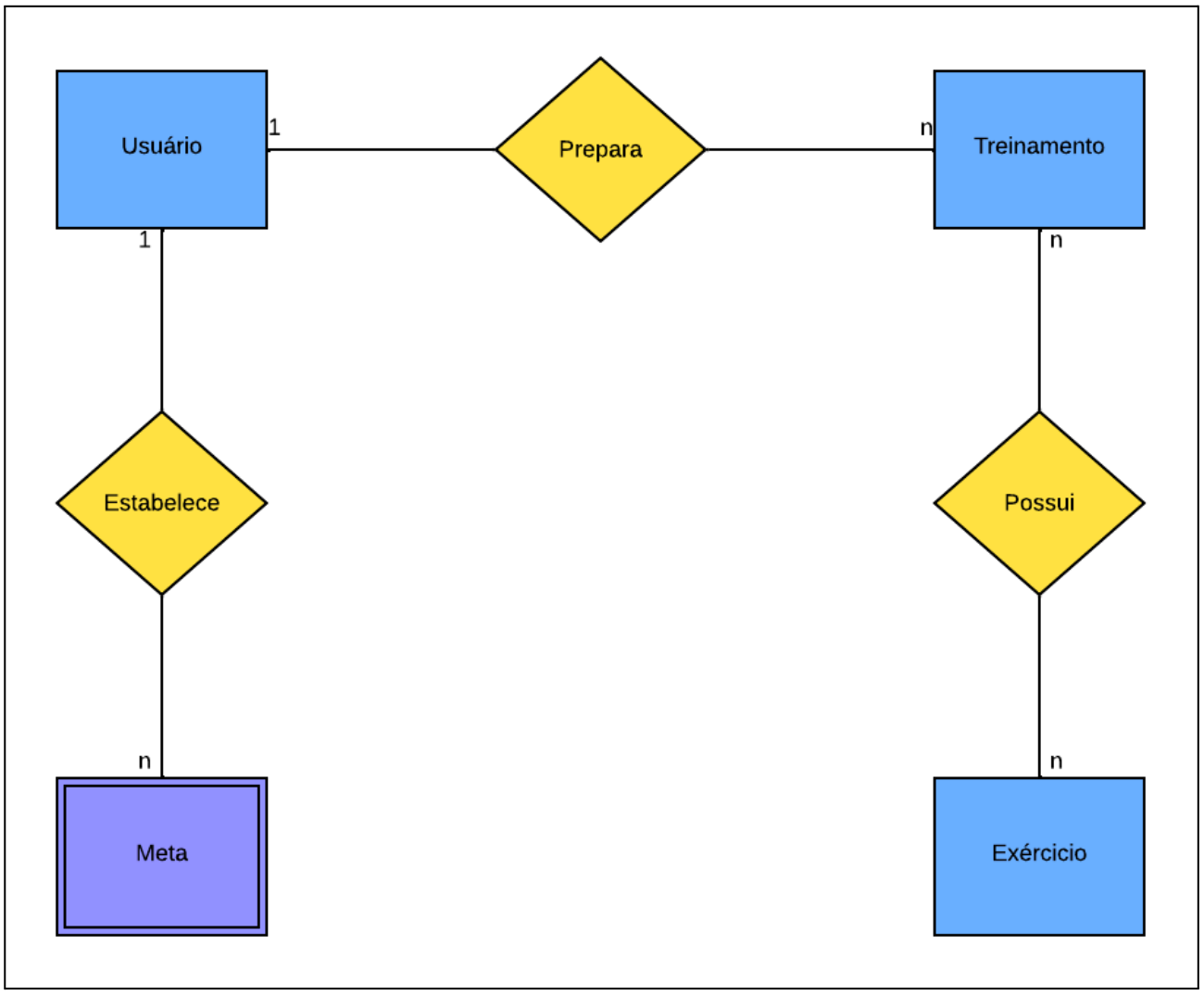


Figura 13 - Diagrama entidade-relacionamento sem atributos

Este primeiro diagrama, é mais simples e tem o objetivo de definir as entidades, identificando quais possuem a natureza forte, ou seja, aquelas que não dependem de outrem para existirem, e aquelas de natureza fraca, que por sua vez, dependem de outrem para existir. No caso acima, podemos identificar que a entidade “Meta” é do tipo fraca e sua existência está vinculada a existência de um “Usuário”.

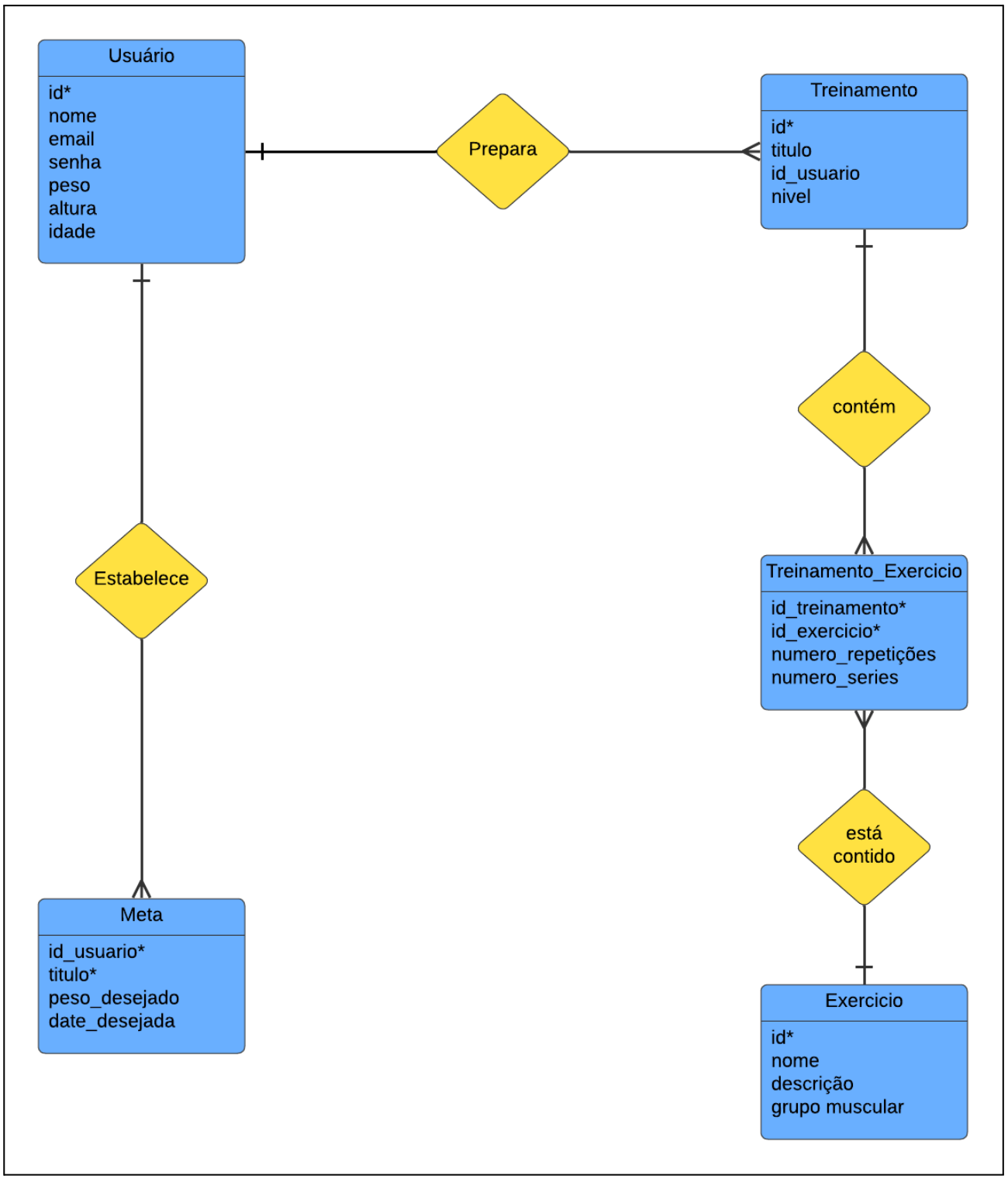


Figura 13 - Diagrama entidade-relacionamento com os atributos

Este segundo diagrama já expressa uma visão mais completa das entidades e como elas serão implementadas, aqui foram adicionados os atributos de cada entidade. Também é

possível notar a adição de uma nova entidade denominada “Treinamento_Usuário”, essa entidade expressa a representação do relacionamento muitos para muitos do diagrama da figura 12, onde em um treinamento pode haver uma lista de exercícios e um exercício pode existir em diferentes treinamentos.

4. DESENVOLVIMENTO DO PROJETO

A elaboração do presente trabalho foi estruturada baseada na arquitetura *client-server,* de modo que as implementações, do tanto do *front-end* quanto do *back-end,* são independentes uma da outra podendo assim ter suas stacks modificadas sem causar nenhum efeito negativo ao funcionamento da aplicação, desde que essas respeitem o protocolo de comunicação estabelecido que neste projeto é o REST estabelecendo padrão JSON para a construção da comunicação dos sistemas.

Para a construção do *front-end* da aplicação foi utilizado o *framework* da linguagem Dart, Flutter, um *app mobile* responsável pela interação com o usuário e o *back-end* responsável pela persistência dos dados e a regra de negócio será uma *API-web*, desenvolvida utilizando o *framework* Phoenix da linguagem Elixir.

4.1. APLICAÇÃO EM FLUTTER

Este capítulo visa apresentar características e conceitos importantes do framework Flutter que foram utilizadas para a construção do *front-end* da aplicação.

4.1. COMPOSIÇÃO DE WIDGETS PARA CRIAÇÃO DE INTERFACES

Em Flutter todos os elementos são considerados Widgets, e a composição de simples widgets formam widgets mais complexos que por sua vez resultam nas interfaces da aplicação, para elucidar esse conceito será utilizado trecho do código do método *build* da classe *LoginScreen,* que tem a função de construir a tela de login da aplicação.

Note na imagem abaixo que a implementação do método *build* espera que o seu retorno seja um *widget*, desta maneira o *return* do método é um *Scaffold( ),* um widget que nesta implementação é composto apenas pelo atributo *“body”,* que recebe o *widget Center( ),* este por sua tem a responsabilidade de centralizar os elementos do widget que será composto através do atributo “*child:”,* que será o *Column( ),* este widget, diferente dos

anteriores, não retorna um *widget,* mas sim um lista de *widgets* através do atributo "*children:"* listas em *Flutter* são passadas entre "[ ]", para a composição da interface da tela de *Login,* o *children* desta *Column( )*, é composto pelos widgets: *Image.asset( ), SizedBox( ), TextField( ) e EnterorRegister( ),* que é um personalizado para implementar o botão de acesso da aplicação e as opções de registro e recuperação de senha.

```dart
Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Image.asset(
              height: 150,
              width: 150,
              'assets/images/logo.jpg',
            ),
            const SizedBox(height: 25,),
            const SizedBox(
              height: 50,
              width: 250,
              child: TextField(),
              ),
            const SizedBox(
              height: 75,
              width: 250,
              child: TextField(),
            ),
            const EnterOrRegister(),
          ],
        ),
      ),
    );
```

Figura 14: Código da implementação da tela de Login da aplicação.

Outra característica importante do Framework possível de se notar no código acima é a sua natureza declarativa o que facilita a construção de interfaces, principalmente para aqueles que têm dificuldade para trabalhar com estilização de telas A composição de todos esses widgets a partir do widget *Scaffold( )* resulta na interface de Login da aplicação que possui o aspecto apresentado na imagem a seguir.

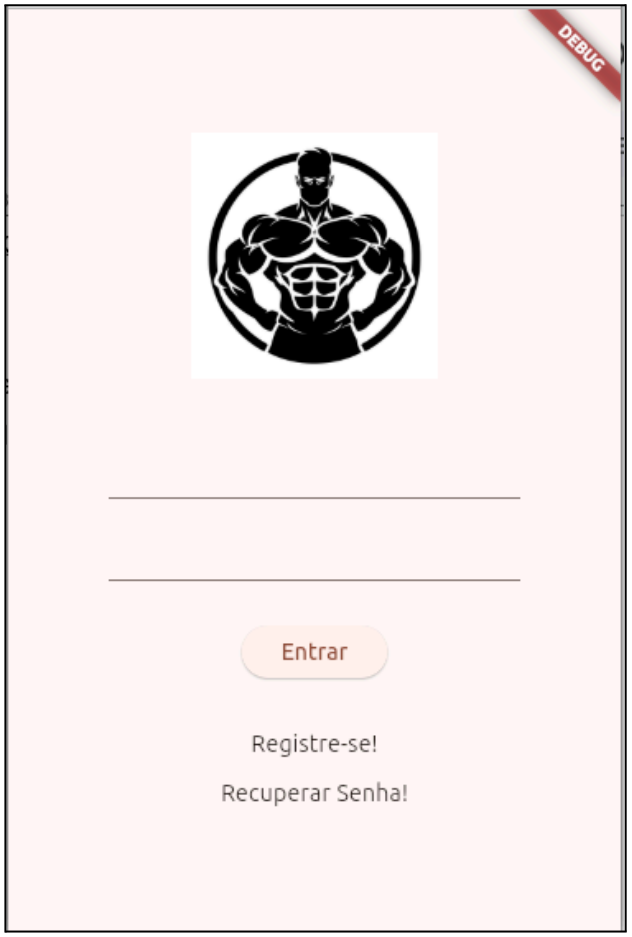


Figura 15: Tela de *Login* da aplicação desenvolvida.

4.2. API-REST PHOENIX

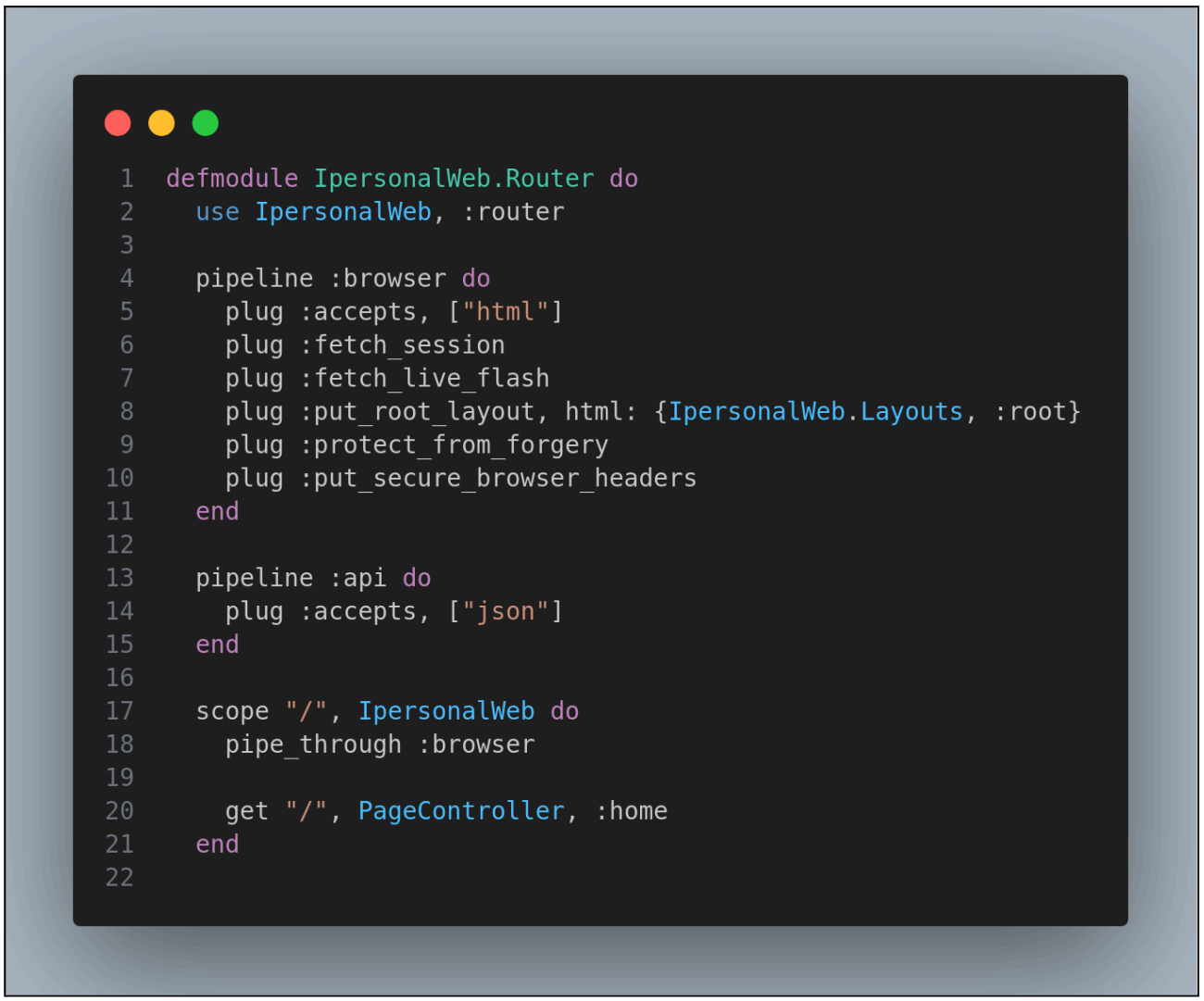
Este capítulo visa apresentar o desenvolvimento do Back-end da aplicação que consiste em um API-Rest desenvolvida com o Framework da linguagem Elixir, denominado Phoenix, um dos seus diferenciais é a utilização do paradigma funcional implementado pela Linguagem e a própria praticidade da linguagem que possibilita a criação de API's de forma limpa e prática.

4.3. SEPARAÇÃO ENTRE REGRA DE NEGÓCIO E COMUNICAÇÃO DA API

Algo que a estrutura do projeto oferece e já facilita o desenvolvimento é a separação da regra de negócio das funções de comunicação da API. Elas são divididas em dois modulos principais, no caso deste projeto, "*Ipersonal*" e "*IpersonalWeb*", o padrão é que um possua o nome da aplicação e ou o nome da aplicação seguido de *Web*. Eles são muito bem segmentadas deixando o código limpo, facilitando assim diferenciar quem tem a responsabilidade de realizar a regra de negócio (*ipersonal*) e quem é responśavel por receber e responder as requisições via API (*ipersonal_web*).

4.4. ESTABELECENDO ROTAS

O arquivo *router* é responsável por estabelecer de que maneira a API aceitará a comunicação, isso acontece com a criação de *pipelines,* que são definidas dentro de escopos é possível criar, escopos de acordo com a necessidade do recurso que está seno solicitado adicionando assim uma grande versatilidade a API. A seguir será analisado o arquivo *router* da aplicação em questão para exemplificar o que foi tratado acima.



```elixir
defmodule IpersonalWeb.Router do
  use IpersonalWeb, :router

  pipeline :browser do
    plug :accepts, ["html"]
    plug :fetch_session
    plug :fetch_live_flash
    plug :put_root_layout, html: {IpersonalWeb.Layouts, :root}
    plug :protect_from_forgery
    plug :put_secure_browser_headers
  end

  pipeline :api do
    plug :accepts, ["json"]
  end

  scope "/", IpersonalWeb do
    pipe_through :browser

    get "/", PageController, :home
  end

```

Figura 14: Arquivo router da aplicação.

Como o phoenix é um *framework web* completo, sendo capaz de gerar interfaces para usuários é possível notar que existe a construção de duas *pipelines,* uma delas denominada *browser,* que é composta por plugs, que tem a função de definir como aquela *pipeline* irá se comportar, por exemplo o *plug :accepts,* recebe o valor [“html”], isso significa que essa pipeline estabelece esse padrão de respostas. Já a *pipeline* api, também possui o *plug :accepts,* porém com o valor [“json”], o que significa que o padrão de comunicação permitido se dará no formato json.

A comunicação acontece através dos escopos que são criados, eles aplicam as pipelines sobre os cursos a fim de estabelecerem a comunicação na forma como essas serão implementadas através dos *Plugs* compostos na pipelines.

5. CONCLUSÃO

Com o desenvolvimento do presente trabalho espera demonstrar a praticidade do framework flutter para o desenvolvimento de aplicações de *User Interface,* que é dada graças a seus inúmeros métodos, que proporcionam soluções elegantes e práticas e sua natureza declarativa, que facilita e adiciona maior eficiência na construção de aplicações. Deste modo, acredito ter demonstrado que o Flutter se apresenta como uma ótima escolha, principalmente para aqueles que não possuem muita desenvoltura com a construção de aplicações *front-end.*

Ademais, a utilização do framework Phoenix na construção da API, teve o objetivo de apresentar a linguagem Elixir, que já consideravelmente utilizada fora do Brasil, a fim de contribuir com a popularização da linguagem em âmbito nacional. Apresentando também uma através do paradigma de programação funcional, uma alternativa prática e eficaz para a construção de aplicações web.

REFERÊNCIAS

Flutter. ***Framework*** **de desenvolvimento.** Disponível em <https://flutter.dev/>. Acesso em 20/04/2024.

Dart. **Linguagem de programação.** Disponível em <https://dart.dev/> Acesso em 20/04/2024.

Elixir. **Linguagem de programação.** Disponível em <https://elixir-lang.org/> Acesso em 20/04/2024.

PEREIRA, Luciana. **Obesidade: hábitos nutricionais, sedentarismo e resistência à insulina. Arquivos Brasileiros de Endocrinologia & Metabologia**, v. 47, n. 2, p. 111-127, 2003.

STEIN, Ricardo. **Physical Inactivity in Brazil and Sweden - Different Countries Similar Problem.** Arquivos Brasileiros de Cardiologia, v. 112, n. 2, p. 119-120, 2019.

BOOCH, Grady; JACOBSON, Ivar; RUMBAUCH, James. **UML – Guia do Usuário**. 2ª Edição. Tradução Fábio Freitas da Silva e Cristiana de Amorim Machado. Rio de Janeiro: Elsevier, 2005.

BOOTH, F. W.; KRUPA, D. **Sedentary death syndrome is what researchers now call America's second largest threat to public health.** Washington, DC, May 29, 2001.

WANNMACHER, Lenita. **Obesidade como fator de risco para morbidade e mortalidade: evidências sobre o manejo com medidas não medicamentosas.** OPAS/OMS - Representação Brasil. ISBN: 978-85-7967-108-1 Vol. 1, Nº 7, 2016, p. 1-10.

PISA, Marcel. **Saiba mais sobre a importância da atividade física na prevenção de doenças.** 2017.Disponível em: <https://claretiano.edu.br/batatais/noticias/88557/saiba-maissobre-a-importancia-da-atividade-fisica-na-prevencao-de-doencas>. Acesso em: 18 mar. 2019.

OSÓRIO, Victor Emanuel Perticarrari. **Middlewares Orientados a Mensagens.** 2023

Berners-Lee, T., Fielding, R. T., and Frystyk, H.. **Hypertext Transfer Protocol – HTTP/1.0. Request for Comments, 1945:1–60.** 1996.

Hura, G.. **Client-server computing architecture: an efficient paradigm for project management. In Proceedings for Operating Research and the Management Sciences, pages 146–152.** 1995.

SEGURA, Sergio et al. Metamorphic testing of RESTful web APIs. In: **Proceedings of the 40th International Conference on Software Engineering**. 2018. p. 882-882.

RAMIRES, Thammy. **Qual a diferença entre web app, app nativo e app híbrido?** FabApp. 25 jan. 2017 .Disponível em: Acessado em: 05 out, 2017

MADUREIRA, Daniel. **Aplicativo nativo, web App ou aplicativo híbrido?** .Net.8 mar. 2017. Disponível em: . Acesso em: 10 out, 2017.

Vazquez, Carlos; Simões, Guilherme. **Engenharia de Requisitos: Software Orientado ao Negócio**. 2016. p. 337.

LUCID. **Ferramenta criação de Diagramas UML**. Disponível em <https://lucid.app/>. Acesso em 10/07/2024