**UNIVERSIDADE SÃO JUDAS TADEU**

**TI & COMPUTAÇÃO**

**BEATRIZ BIAGIONI DOS SANTOS RA: 821123729**

**BRUNO VENÂNCIO DE SOUZA E SILVA RA: 821135934**

**EDUARDA FERREIRA GALENO RA: 821135196**

**ERICK DE SOUSA CAVALCANTE RA: 821142827**

**GUSTAVO DE LANA ROCHA RA: 82118015**

**HENRICK MELO VITAL RA: 821224905**

**YGOR BRANCAGLIONE CUCHI RA: 821141099**

**PROJETO UNIDADE CURRICULAR A3**

**GESTÃO E QUALIDADE DE SOFTWARE**

**MODELO, METODOS E TECNICAS DE ENGENHARIA DE SOFTWARE**

[**1 INTRODUÇÃO 3**](#_ohk81b5ire5t)

[**2 PADRÕES PARA O PROJETO 3**](#_eptudoo33mhf)

[**2.1 LINGUAGENS 3**](#_grehnldg0slj)

[**2.2 BANCO DE DADOS 4**](#_jvfmwaxbmb8b)

[**2.3 FERRAMENTA DO PROJETO 4**](#_gm00m49n25ef)

[**2.3.1 GIT 4**](#_7d5w7w61yohv)

[**2.3.2 ANDROID STUDIO 4**](#_2ybejdhdrl20)

[**2.3.3 SPRING BOOT 4**](#_1gynb7viflu0)

[**2.3.4 JETPACK COMPOSE 5**](#_fdoih39og62c)

[**3 BOAS PRÁTICAS 5**](#_l36pekiy2umi)

[**3.1 CONVENÇÕES DE LINGUAGENS 5**](#_uqwlj04014t9)

[**3.2 ORGANIZAÇÃO DOS ARQUIVOS 5**](#_qzal43mykk76)

[**3.3 API MÍNIMA 5**](#_kv2o6ji7hr58)

[**3.4 AMBIENTE DE EXECUÇÃO 6**](#_kycey2xvo8br)

[**3.5 BANCO DE DADOS 6**](#_1bd8am8prw4x)

[**3.6 PROCESSO DE DESENVOLVIMENTO ÁGIL 6**](#_lwsc3v32je59)

[**4 REQUISITOS DE SOFTWARE 7**](#_wyesjdfn287h)

[**5 ARQUITETURA DE SOFTWARE 10**](#_q9mqo06rm0o5)

[**5.1 ARQUITETURA DO APLICATIVO MÓVEL 10**](#_3qrcndpbbdcf)

[**5.1.1 MODEL 10**](#_5nhs638i6e54)

[**5.1.2 VIEW 10**](#_gg6902286jym)

[**5.1.3 VIEWMODEL 10**](#_rnvi65fmmfpe)

[**5.2 ARQUITETURA DO BACK-END 11**](#_9nv7lqu5d7yz)

[**5.3 ARQUITETURA DO BANCO DE DADOS 12**](#_tig5vdmg9jz1)

[**6 MODELAGEM 12**](#_j6s1ahwlkvh5)

[**6.1 VISÃO GERAL DO APLICATIVO 13**](#_sjievk4xhcx8)

[**6.2 CASO DE USO 13**](#_6j92rcdafadq)

[**6.3 DIAGRAMA DE ENTIDADE-RELACIONAMENTO 14**](#_85nypurr564b)

[**7 ENGENHARIA DE SEGURANÇA 15**](#_jig9ug1r5p7x)

[**7.1 VALIDAÇÃO DE DADOS DE ENTRADA 15**](#_lolx5wtjaz3)

[**7.2 ARMAZENAMENTO DE DADOS SENSÍVEIS 15**](#_gzgczc67vvit)

[**7.3 LIMITE DE TENTATIVAS DE ACESSO E RECUPERAÇÃO DE CONTA 15**](#_lzjbmgnrcayg)

[**7.4 CONFIRMAÇÃO DE CADASTRO E AVISO DE ALTERAÇÕES 15**](#_b30s0x75ci5)

[**8 PLANOS DE TESTE 16**](#_xa07s8mwpj9r)

[**8.1 TELA LOGIN 16**](#_xplqdmkw2qfh)

[**8.2 CADASTRO DE USUÁRIO 16**](#_rtg99bin9mm3)

[**8.3 RECUPERAÇÃO DE SENHA 17**](#_seu3v5iwiqgv)

[**8.4 TELA INICIAL 17**](#_csawpixeomxp)

[**8.5 COLETA DE DADOS NUTRICIONAIS 17**](#_o6tgkm1mw1h3)

[**9 CUSTO DO DESENVOLVIMENTO DE SOFTWARE 18**](#_lce9oox1st26)

[**10 MONETIZAÇÃO DA APLICAÇÃO 18**](#_q2hv2jp8sjb7)

[**11 REFERÊNCIAS 19**](#_ph1ra7cdphme)

# 

# **1 INTRODUÇÃO**

Este documento tem como objetivo documentar o projeto de avaliação A3 “Your Health” para as UCs de Gestão e Qualidade de Software e Modelos, métodos e técnicas da engenharia de software, tratando os requisitos de software, a arquitetura de software utilizada, plano de testes, usabilidade e monetização do projeto.

A ideia do aplicativo consiste em monitorar a alimentação de nossos usuários, disponibilizando informações detalhadas sobre os alimentos consumidos com base na tabela brasileira de composição de alimentos, gerar dados e infográficos com base nas informações salvas a fim de conscientizar sobre alimentação saudável, promovendo o bem-estar de jovens adultos até a terceira idade.

Com os dados de consumo salvos no dispositivo móvel do usuário, o mesmo pode facilmente monitorar seu consumo de calorias e nutrientes, facilitando inclusive o atendimento de profissionais da nutrição que podem usar os dados para melhor atender seus pacientes, com a autorização do usuário.

# **2 PADRÕES PARA O PROJETO**

O aplicativo e serviços derivados como os microsserviços e o banco de dados seguem alguns padrões de desenvolvimento. Os padrões arquitetônicos estão descritos na seção “5 Arquitetura de Software”, esta seção destina-se apenas aos padrões que envolvem o desenvolvimento e manutenção do software.

## **2.1 LINGUAGENS**

A aplicação móvel utiliza a linguagem de programação Kotlin, esse é um requisito importante devido a dependência com o kit de ferramentas JetPack Compose, a linguagem é interoperável com a linguagem Java, podendo inclusive utilizar suas bibliotecas padrão. Os microsserviços do back-end utilizam a linguagem Java através do framework Spring Boot, apesar de ser possível reutilizar a linguagem Kotlin, prefere-se priorizar Java para melhor distinguir os domínios da aplicação, como as linguagens interoperar entre si, componentes podem ser facilmente reutilizáveis. Por fim, a comunicação do banco de dados deve ser feita em SQL. Para maiores informações consultar a seção “3 Boas Práticas” deste documento.

## **2.2 BANCO DE DADOS**

O projeto vai utilizar dois tipos diferentes de bancos de dados relacionais, para a aplicação móvel, o SQLite será utilizado, pois fará o armazenamento local de apenas os dados do usuário ativo na aplicação. Nos microsserviços, o banco de dados será o PostGreSQL, devido ao maior volume de dados e aos acessos simultâneos entre os microsserviços.

## **2.3 FERRAMENTA DO PROJETO**

O projeto utiliza diversas ferramentas para o desenvolvimento da aplicação e dos microsserviços, abaixo uma breve descrição dessas ferramentas:

### **2.3.1 GIT**

O sistema de gerenciamento de versões do projeto é feito pelo software Git, o repositório central é localizado em <https://github.com/d4rkwav3/projetoa3.2023.1/> , e cada desenvolvedor deve criar seu próprio ramo (branch) ao realizar qualquer tipo de alteração no projeto. Apenas após as alterações terem sido testadas, revistas e aprovadas pela equipe, o ramo poderá ser fundido ao ramo main (principal).

### **2.3.2 ANDROID STUDIO**

Para o desenvolvimento da aplicação móvel, o desenvolvedor utiliza a IDE (integrated development environment) Android Studio, essa IDE possui diversas facilidades que agilizam o desenvolvimento da aplicação, entre elas o download e verificação automática de dependências, automatização de testes, padrões de projeto, checagem de qualidade de código, cálculo de cobertura dos testes, ambiente de emulação do sistema Android, pré-visualização de interfaces gráficas, entre outras.

### **2.3.3 SPRING BOOT**

Os microsserviços de suporte ao aplicativo móvel devem ser desenvolvidos utilizando a framework Spring Boot, utilizando as dependências “Spring Web” para as requisições HTTP e HTTPS e “Spring JPA” para transações com o banco de dados.

### **2.3.4 JETPACK COMPOSE**

Todo o desenvolvimento da interface gráfica de usuário para a aplicação móvel irá utilizar o kit de ferramentas JetPack Compose, de modo a acelerar e simplificar o desenvolvimento da aplicação.

# **3 BOAS PRÁTICAS**

A seguir uma série de boas práticas que devem ser utilizadas extensivamente e sem exceções durante todo o ciclo de desenvolvimento da aplicação.

## **3.1 CONVENÇÕES DE LINGUAGENS**

As conversões de programação das linguagem Kotlin (para a aplicativo móvel) e Java (para os serviços em back-end) são o guia para o projeto da aplicação, tais convensões podem ser acessadas nos links <https://kotlinlang.org/docs/coding-conventions.html> e <https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>. É importante exaltar algumas dessas convenções nesse documento. Apesar da interoperabilidade entre kotlin e java, todos os arquivos do pacote do projeto móvel devem utilizar a sintaxe kotlin, ao se utilizar arquivos do back-end, eles devem ser reestruturados às convenções kotlin, o mesmo se aplica na reutilização de arquivos kotlin nos microsserviços em java.

## **3.2 ORGANIZAÇÃO DOS ARQUIVOS**

Os arquivos do projeto devem residir em diretórios que abriguem arquivos de funções semelhantes, por exemplo, os arquivos de classe de UI devem estar na pasta/pacote “UI”, as classes de modelo na pasta “Model”. Toda classe de função deve ter uma pasta própria, arquivos que desempenham funções semelhantes porém agem em domínios diferentes devem ainda ser separados em sub diretórios específicos.

## **3.3 API MÍNIMA**

O uso da API mínima do android deve obrigatoriamente cobrir um mínimo de 90% dos dispositivos móveis, a versão Java não deve ser inferior a 11.

## **3.4 AMBIENTE DE EXECUÇÃO**

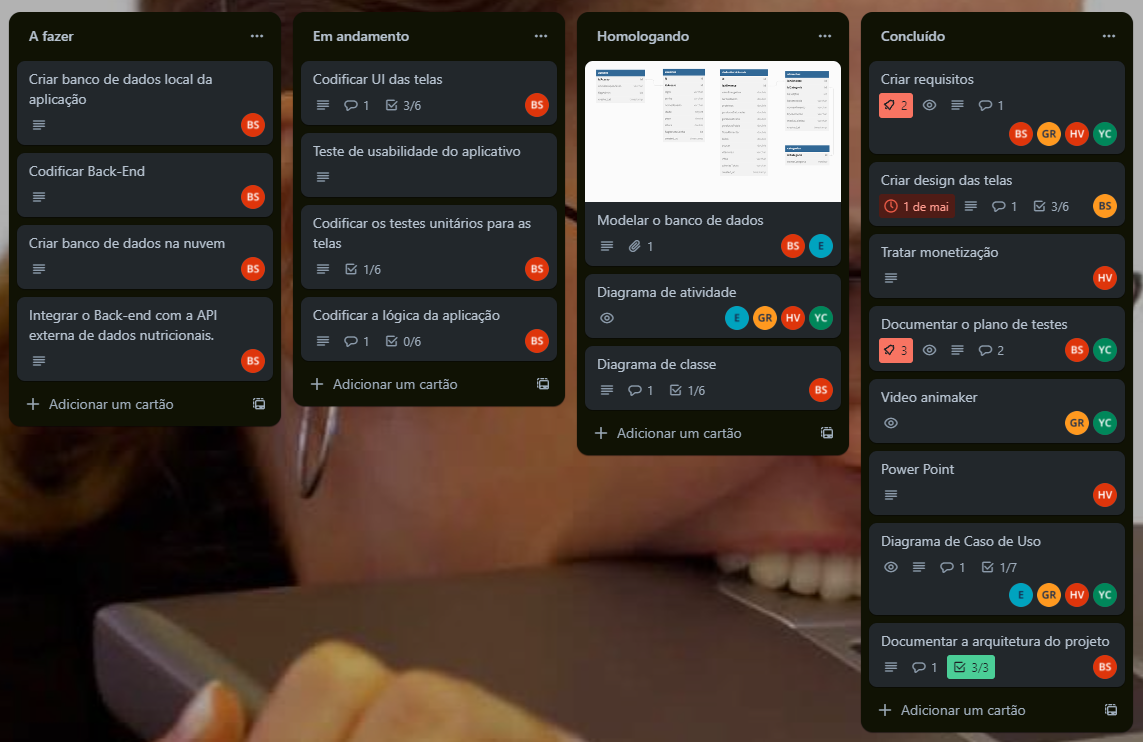
O ambiente de execução dos microsserviços é o GNU/Linux Debian, em sua versão LTS (Long Term Support) mais recente. O aplicativo móvel é executado na versão 7 ou superior do sistema operacional Android (com base no item 3.3).

## **3.5 BANCO DE DADOS**

Apesar de usar bancos de dados diferentes, o SQLite no aplicativo móvel e o PostGreSQL nos microsserviços, ambos serão acessados a partir da ORM Room no aplicativo móvel e pela Spring JPA nos microsserviços, o que garante a abstração na manipulação dos dados e uso de melhores práticas de segurança.

## **3.6 PROCESSO DE DESENVOLVIMENTO ÁGIL**

O sistema será desenvolvido utilizando o método de desenvolvimento ágil SCRUM, dividindo as tarefas da equipe em Sprints, focando nos objetivos de curto, médio e longo prazo para alcançar os objetivos da maneira mais rápida e eficiente possível.  
O quadro de tarefas da equipe usará o conceito Kanban, da ferramenta Trello.



# **4 REQUISITOS DE SOFTWARE**

Todas as funcionalidades iniciais da aplicação estão descritas a seguir:

| **FUNÇÃO** | Cadastro de usuários |
| --- | --- |
| **REQUISITOS FUNCIONAIS** | O usuário deve inserir seus dados pessoais como altura, peso, idade, e-mail e senha para o cadastro.  O sistema permitirá que usuários sejam cadastrados, classificando-os como: usuário comum ou Premium.  O sistema permitirá que os usuários façam alterações em seus cadastros.  O cadastro do usuário comum deve conter: nome, e-mail, senha, idade, peso e altura.  O cadastro do usuário premium deve conter: nome, e-mail, senha, idade, peso, altura e cartão de crédito. |
| **REQUISITOS NÃO FUNCIONAIS** | Segurança: Os dados inseridos pelo usuário devem ser válidos e seguir os requisitos mínimos de segurança, como quantidade mínima de caracteres e utilização de caracteres especiais na senha.  Confiabilidade: Às informações inseridas como endereço de e-mail deve ser de um formato válido.  Usabilidade: Instruções e dicas são exibidas na tela de modo a auxiliar a inserção dos dados. |

| **FUNÇÃO** | Login |
| --- | --- |
| **REQUISITOS FUNCIONAIS** | Um usuário deve poder fazer login no sistema usando seu email e senha.  O sistema deve permitir que usuário realizem novos cadastros.  O sistema deve permitir que o usuário utilize o recurso de recuperação de senha **sendo encaminhado para a tela responsável**. |
| **REQUISITOS NÃO FUNCIONAIS** | Segurança: A conexão com o servidor deve ser criptografada.  Usabilidade: Apenas o necessário para o usuário acessar o aplicativo deve estar disposto na tela, demais funções de recuperação deve redirecionar o usuário a tela correspondente. |

| **FUNÇÃO** | Recuperação de Senha |
| --- | --- |
| **REQUISITOS FUNCIONAIS** | O sistema deve auxiliar o usuário a executar o passo a passo necessário para a recuperação de senha.  O sistema deve enviar um email de recuperação de senha para o email cadastrado pelo usuário, **isso se** o mesmo possuir um cadastro ativo.  O sistema deve redefinir a senha do usuário a partir do email enviado. |
| **REQUISITOS NÃO FUNCIONAIS** | Usabilidade: A tela fornece as instruções necessárias para o processo de recuperação de senha.  Segurança: O processo de recuperação cria um código de uso único com tempo limite enviado ao e-mail cadastrado, caso exista. |

| **FUNÇÃO** | Tela Inicial (dashboard) |
| --- | --- |
| **REQUISITOS FUNCIONAIS** | O sistema deve apresentar um dashboard que exibe os dados de consumo referentes ao usuário logado.  O sistema deve possuir recursos na tela inicial que possibilitem o redirecionamento para todas as demais telas do sistema. |
| **REQUISITOS NÃO FUNCIONAIS** | Usabilidade: As informações são organizadas em formato de cartão para melhor visualização.  Desempenho: O aplicativo não deve travar enquanto carrega as informações, que devem ser processadas em segundo plano e exibidas quando concluídas. |

| **FUNÇÃO** | Coleta de dados nutricionais |
| --- | --- |
| **REQUISITOS FUNCIONAIS** | O sistema vai receber os dados dos alimentos ingeridos pelo usuário através de uma tela de pesquisa de alimentos.  O sistema vai aceitar alterações dos dados dos alimentos inseridos, por exemplo, quantidade ingerida, e recalcular os dados nutricionais automaticamente. |
| **REQUISITOS NÃO FUNCIONAIS** | Usabilidade: A pesquisa de termos deve ser dinâmica a acontecer à medida que o usuário digita uma palavra, exibindo resultados preliminares.  Desempenho: O aplicativo não deve travar enquanto carrega as informações, que devem ser processadas em segundo plano e exibidas quando concluídas. |

# **5 ARQUITETURA DE SOFTWARE**

Esta seção descreve a arquitetura de software utilizada no aplicativo “Your Health”, sendo dividido em três seções, a primeira, descreve a arquitetura utilizado para o desenvolvimento do aplicativo móvel, a segunda, a arquitetura utilizada nos serviços de back-end de suporte ao aplicativo, e a terceira seção descreve brevemente a arquitetura de banco de dados da aplicação e o serviço de back-end. Ao final as referências são descritas na quarta seção.

## **5.1 ARQUITETURA DO APLICATIVO MÓVEL**

O projeto de arquitetura de software que será utilizado no aplicativo móvel “Your Health’ é o MVVM (Model View ViewModel), devido a utilização do toolkit (kit de ferramentas) JetPack Compose, que permite simplificar e acelerar o desenvolvimento de interface de usuário nativa no sistema Android, essa arquitetura é um requisito para a utilização dessas ferramentas. Apesar de bastante similar a arquitetura MVC (Model View Controller), o MVVM tem algumas peculiaridades que serão discutidas a seguir:

### **5.1.1 MODEL**

O Model (modelo, ou classes de modelo) refere-se às classes que determinam as regras de negócio da aplicação, classes de dados que se adequam às exigências do banco de dados ou qualquer classe para representar alguma informação estática ou dinâmica, como configurações do aplicativo, informações de banco de dados ou estado da interface de usu. São classes independentes utilizadas pelo ViewModel para enviar as informações à View adequada.

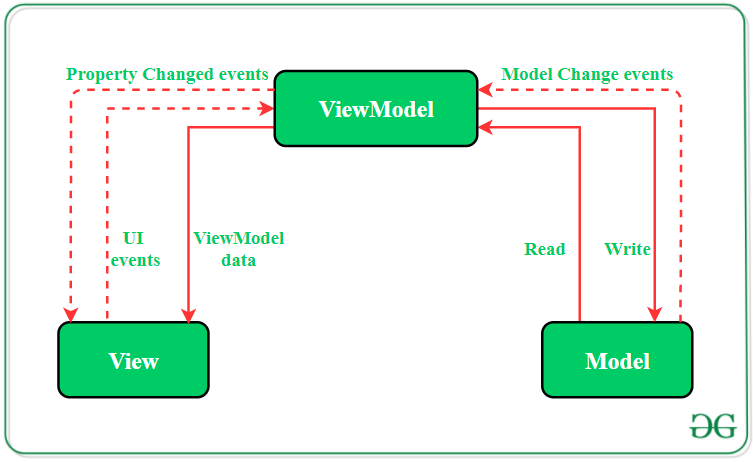
### **5.1.2 VIEW**

A View (visão, ou classes de visão) refere-se às classes responsáveis por desenhar a interface de usuário na tela do aplicativo, elas são usadas exclusivamente para descrever as regras para a construção dos elementos visuais, não contendo qualquer tipo de dados, estes serão trazidas do domínio de dados (as classes modelos) através da ViewModel.

### **5.1.3 VIEWMODEL**

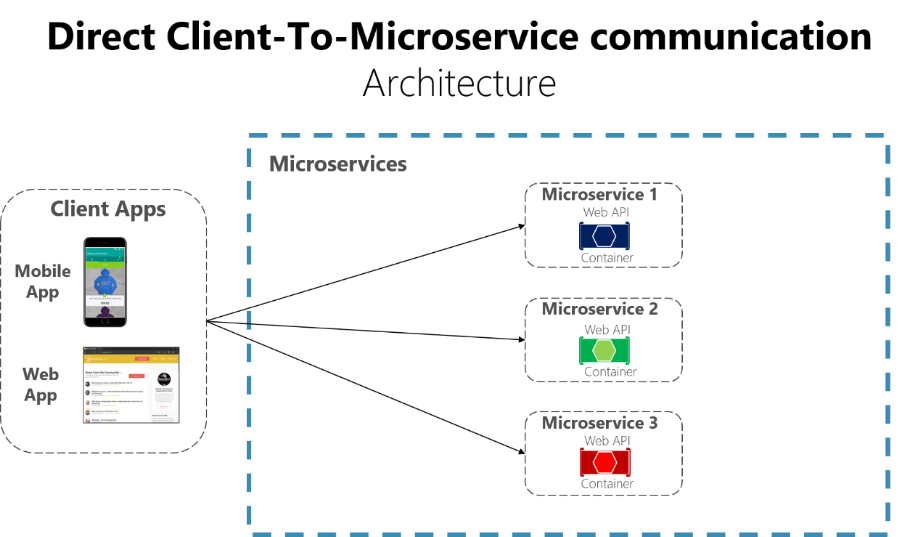
A ViewModel (modelo de visão, ou classes de modelo de visão) são as classes que utilizam e manipulam as classes de modelo para trazer as informações relevantes para as classes de visão exibidas na interface de usuário. Ela também é responsável por gerenciar o estado da interface de usuário e a utilização das regras de negócio para o funcionamento adequado da aplicação. Cada classe de visão deve possuir uma classe de modelo de visão correspondente.

Em suma, a arquitetura MVVM pode separada em dois domínios diferentes, o domínio da UI (user interface, ou interface de usuário) e o domínio de dados, as classes de visão pertencem ao domínio da UI, e as classes de modelo ao domínio de dados, as classes de modelo de visão são a interface entre os domínios, essas podem receber requisições das classes de visão para a atualização de componentes específicos, requisitar dados as classes de dados e gerenciar o estado geral do aplicativo. O diagrama abaixo mostra o fluxo de informações da arquitetura (MISHRA, 2022).



## **5.2 ARQUITETURA DO BACK-END**

A arquitetura dos serviços de back-end será baseada em microsserviços, utilizando o padrão de comunicação direta de cliente para microsserviço. Cada funcionalidade é separada em um microsserviço independente, como um microsserviço para login de usuários, outro para cadastro de usuários, um outro para pesquisa e assim por diante. A aplicação cliente é pré-configurada para solicitar informações e recursos aos microsserviços adequados. Abaixo um diagrama da arquitetura (“Padrão de gateway de API versus comunicação direta de cliente com microsserviço”, 2023):



## **5.3 ARQUITETURA DO BANCO DE DADOS**

A arquitetura de banco de dados será do modelo de dados relacional, um banco de dados será composto por diversas tabelas contendo informações altamente coesas, isto é, contendo apenas as informações relevantes a um domínio específico da aplicação, o cruzamento de informações em tabelas diferentes se faz através de chaves estrangeiras que referenciam informações em outras tabelas. Tanto a aplicação móvel como os microsserviços farão uso do banco de dados, com o diferencial de que a aplicação móvel terá uma cópia limitada no armazenamento do dispositivo, a fim de agilizar o processamento dos dados e reduzir a dependência de uma conexão constante com a internet.

# **6 MODELAGEM**

A seguir alguns diagramas para a iteração inicial do projeto.

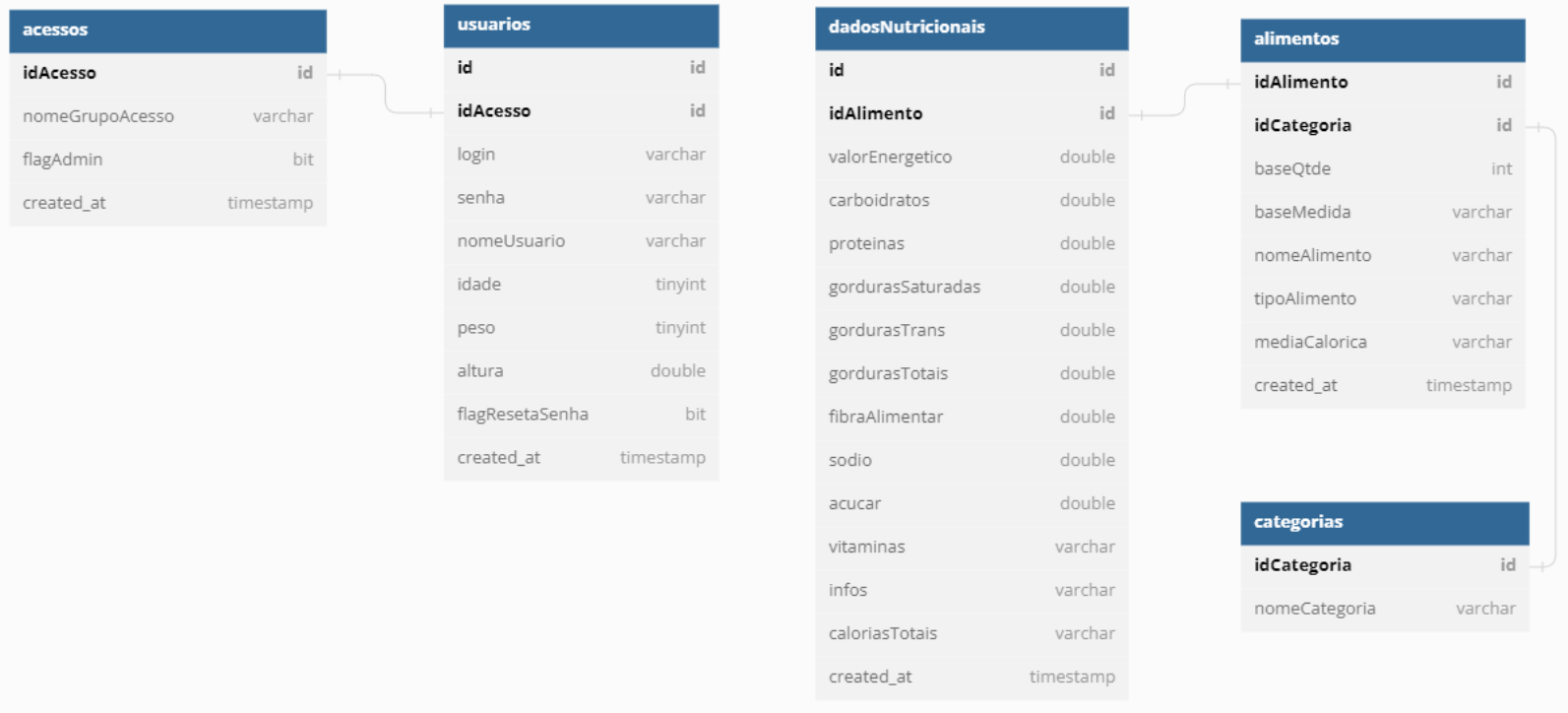
## **6.1 VISÃO GERAL DO APLICATIVO**

## 

## 

## **6.2 CASO DE USO**

## **6.3 DIAGRAMA DE ENTIDADE-RELACIONAMENTO**



# **7 ENGENHARIA DE SEGURANÇA**

Esta seção descreve os requisitos de segurança do aplicativo do projeto A3 “Your Health”, dividido por tópicos.

## **7.1 VALIDAÇÃO DE DADOS DE ENTRADA**

Todas as entradas de dados de fontes externas (consumo de APIs) ou de entrada de dados de pesquisa por parte do usuário precisam ser validadas, erros de formatação devem ser tratados automaticamente quando possível e caso contrário, o usuário deve ser informado sobre o erro de formatação.

## **7.2 ARMAZENAMENTO DE DADOS SENSÍVEIS**

A arquitetura de banco de dados foi desenvolvida para armazenar o mínimo de informações do usuário possível, porém a senha de acesso a conta do usuário ainda é necessária, o armazenamento da senha de acesso deve ser feito exclusivamente em formato de hash (uma string de tamanho fixo criada a partir de um algoritmo criptográfico) sha-256 ao invés de texto simples, também será utilizado o uso de salt (uma combinação randômica de letras, símbolos e números) na senha para maior confidencialidade dos dados.

## **7.3 LIMITE DE TENTATIVAS DE ACESSO E RECUPERAÇÃO DE CONTA**

O sistema deve limitar a quantidade de tentativa de acesso, contabilizando as tentativas sem sucesso e bloqueando novas tentativas após um limite pré-definido ser atingido, a recuperação do acesso deve gerar um código de uso único no formato UUID (identificador único universal), enviado para o e-mail cadastrado, ao acessar a url com o código, o mesmo deve ser automaticamente marcado como utilizado e o usuário deve atualizar a senha cadastrada. Não completar o processo de recuperação deve exigir a geração de um novo código de recuperação.

## **7.4 CONFIRMAÇÃO DE CADASTRO E AVISO DE ALTERAÇÕES**

Ao se cadastrar no aplicativo, o usuário só deve ter acesso completo às suas funcionalidades após confirmar o endereço de e-mail utilizado para criar o cadastro, alterações subsequentes de senha ou e-mail deve gerar uma nova confirmação a ser enviada no e-mail cadastrado, antes da alteração ser efetivada.

# **8 PLANOS DE TESTE**

Os planos de testes a seguir estão divididos com base nas telas de usuário da aplicação.

## **8.1 TELA LOGIN**

1. Verificar se o e-mail já consta em nossa base de dados.
2. A senha deve ser convertida em uma hash em formato SHA-256 antes de ser comparada à base de dados.
3. Botão de Login:
   1. Ao ativar o botão, a aplicação deve navegar a tela inicial em caso de sucesso.
   2. Em caso de erro, uma mensagem de erro deve ser apresentada.
4. Texto clicável de recuperação de senha.
   1. Clicar no texto deve abrir a tela de recuperação de senha.
5. Texto clicável de novo cadastro.
   1. Clicar no texto deve abrir a tela de cadastro de usuário.

## **8.2 CADASTRO DE USUÁRIO**

1. Campo de texto para o e-mail:
   1. A string digitada deve ter no mínimo 10 caracteres;
   2. O primeiro caractere da string não pode ser um dígito;
   3. O caractere “@” (Arroba) deve ser localizado na string;
   4. O caractere de . (Ponto) deve ser localizado após o caractere de @.
   5. A string não pode conter caracteres em branco.
   6. Deve ser único na base de dados.
2. Campo de texto para a senha:
   1. A string deve ser mascarada por algum caractere coringa.
   2. A string deve ter um comprimento mínimo de 8 caracteres.
   3. A string não pode conter caracteres em branco.
   4. A string deve conter no mínimo 3 números não sequenciais.
   5. A string deve conter ao menos 2 caracteres especiais.
   6. Uma hash em formato SHA-256 ou superior deve ser criada com base na string digitada.
3. Validar se o usuário é comum ou premium
4. Verificar se a segunda senha digitada é idêntica à primeira senha digitada.
5. Verificar se o campo “Idade” é um número inteiro não menor que 13 ou maior que 100.
6. Verificar se o campo “Altura” é um número de ponto flutuante de duas casas decimais, maior que 0,50 e menor que 2,75.
7. Verificar se o campo “Peso” é um número de ponto flutuante de duas casas decimais, maior que 40,00 e menor que 200,00.

## **8.3 RECUPERAÇÃO DE SENHA**

1. Verificar se o e-mail inserido existe;
2. Verificar se a senha já foi utilizada;
3. Verificar se a senha atende os mesmo requisitos da senha da tela “cadastro de usuário" item B;
4. Verificar se o usuário consegue alterar sua senha somente com o código enviado ao e-mail cadastrado;
5. Verificar o tempo limite do código enviado para o e-mail;
6. Verificar o código enviado para o e-mail cadastrado é único.

## **8.4 TELA INICIAL**

1. Verificar se os dados que aparecem no dashboard são os mesmos dados do usuário que fez o login;
2. Verificar se essa tela está levando o usuário para demais telas;
3. Verificar se a tela está otimizada para trazer os dados sem travamentos.

## **8.5 COLETA DE DADOS NUTRICIONAIS**

1. Verificar se os dados estão corretamente de acordo com o usuário logado;
2. Verificar se o usuário pode inserir uma grande quantidade de itens sem causar lentidão no sistema.

# **9 CUSTO DO DESENVOLVIMENTO DE SOFTWARE**

Os custos iniciais para desenvolver um aplicativo móvel são:

* Salário do(s) programador(es)/desenvolvedor(es);
* Salário do designer;
* Custo mensal de infraestrutura;
* Custos para publicar o aplicativo nas lojas;

O “back-end” ficará nos servidores da Oracle Cloud, que tem um gasto de armazenamento em blocos de US$4,25/mês com disponibilidade de 100GB e 6000 IOPS, Saída de dados $0,0085 por GB/hora (a saída dos primeiros 10TB são gratuitos) e uma VM para fins gerais que custará $0,0980/hora (VM.STANDARD.E4.FLEX e 2 núcleos (4VCPU) e 32 GB RAM). O aplicativo será hospedado na loja da Google (Play Store) que há um custo de U$25, que será pago uma única vez. O gasto com um desenvolvedor Android é em média R$8.500,00. O gasto com o designer é em torno de R$4.000.

Portanto, os gastos iniciais são de aproximadamente R$14.000,00, compondo desenvolvimento, designer, infraestrutura e custos de publicação. Mas poderá aumentar conforme a demanda de dados e IOPS aumentem.

# **10 MONETIZAÇÃO DA APLICAÇÃO**

O objetivo principal é estar entre os 800 aplicativos de produtividade da Google Play, que faturam em torno de US$3.500,00 por mês.

O faturamento da aplicação será feito de três formas: Na primeira, o aplicativo terá anúncios pagos, com uma estimativa de que a cada 1000 visualizações pode se faturar entre 15 a 150 dólares. Na segunda, a aplicação terá planos premium com pagamento mensal, que darão acesso a funções exclusivas no aplicativo, como sincronização na nuvem, acesso a relatórios customizados e navegação sem anúncios. Na terceira, cupons de descontos para comprar alimentos e suplementos em mercados e marketplaces afiliados focados em vida saudável e nutrição, onde recebemos uma porcentagem da venda.

# **11 REFERÊNCIAS**

**Kit de ferramentas do Jetpack Compose para desenvolver a IU de apps: desenvolvedores Android**. Disponível em: <https://developer.android.com/jetpack/compose?hl=pt-br>. Acesso em: 15 maio. 2023.

MISHRA, R. **MVVM (model view ViewModel) architecture pattern in android**. Disponível em: <https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/>. Acesso em: 15 maio. 2023.

**Padrão de gateway de API versus comunicação direta de cliente com microsserviço**. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern>. Acesso em: 15 maio. 2023.

**Preços e Custos da Oracle Cloud**. Disponível em: <https://www.oracle.com/br/cloud/pricing/>. Acesso em: 26 jun. 2023.

**Salários do cargo de Android Developer**. Disponível em: <https://www.glassdoor.com.br/Sal%C3%A1rios/s%C3%A3o-paulo-android-developer-sal%C3%A1rio-SRCH\_IL.0,9\_IM1009\_KO10,27.htm>. Acesso em: 26 jun. 2023.

**Salários do cargo de Designer**. Disponível em: <https://www.glassdoor.com.br/Sal%C3%A1rios/s%C3%A3o-paulo-designer-sal%C3%A1rio-SRCH\_IL.0,9\_IM1009\_KO10,18.htm>. Acesso em: 26 jun. 2023.