UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
ENGENHARIA DE SOFTWARE

ARTHUR MOURA BERNARDO
PIETRO NIERO ROQUE

TASKMATE

Documento de Arquitetura do Software TaskMate

GOIÂNIA
2023

1. INTRODUÇÃO

1.1. FINALIDADE

Este documento tem como objetivo descrever a arquitetura de software do sistema TaskMate, fornecendo uma visão geral das principais características, componentes e interações.

1.2. ESCOPO

O escopo deste documento abrange a arquitetura de software do sistema TaskMate, um aplicativo de organização de *tasks,* que permite aos usuários criar, editar e excluir tarefas com diferentes atributos e opções de personalização.

2. CONTEXTO DA ARQUITETURA

2.1. FUNCIONALIDADE E RESTRIÇÕES ARQUITETURAIS

O sistema TaskMate deve fornecer as funcionalidades descritas no **documento de requisito (DRS)**, como a criação, edição e exclusão de tarefas, personalização de notificações e suporte a tarefas isoladas e recorrentes. Além disso, a arquitetura deve levar em consideração as restrições de desempenho, segurança e usabilidade.

2.2. ATRIBUTOS DE QUALIDADE PRIORITÁRIOS

Conforme as funcionalidades e restrições definidas no tópico anterior (2.1), foram selecionados os seguintes atributos de qualidade a serem priorizados:

- **Confiabilidade:** Garantir que as notificações sejam enviadas corretamente nas datas associadas às tarefas.
- **Usabilidade:** Proporcionar uma interface amigável e intuitiva para os usuários.
- **Desempenho:** Assegurar tempos de resposta rápidos ao criar, editar e excluir tarefas.
- **Segurança:** Proteger os dados do usuário e garantir a privacidade das informações.
- **Disponibilidade:** Garantir a disponibilidade do sistema de forma constante, mesmo que sem conexão com a internet.

A escolha desses atributos e as funcionalidades e restrições arquitetônicas definidas direcionam o software TaskMate para um estilo arquitetural em **Camadas**.

A abordagem em camadas é adequada para separar as responsabilidades de forma modular e bem organizada, facilitando o desenvolvimento, manutenção e escalabilidade do software. Além disso, permite que cada camada seja modificada independentemente, garantindo a flexibilidade e adaptabilidade do sistema.

O padrão de arquitetura Cliente-Servidor foi levantado como um modelo arquitetural para ser usado em conjunto com o padrão em Camadas, mas foi

rejeitado pois não atenderia o atributo de qualidade *Disponibilidade*, que foi priorizado. Esse atributo enfatiza a importância da disponibilidade off-line do sistema, algo que não é garantido no padrão Cliente-Servidor. Além disso, a capacidade do cliente de acessar recursos do servidor neste modelo arquitetural não seria de ajuda, visto que o documento de requisitos especifica um armazenamento local.

3. REPRESENTAÇÃO ARQUITETURAL.

De acordo com o definido pelos tópicos anteriores, a arquitetura do software será uma arquitetura em Camadas, e vai priorizar Confiabilidade, Usabilidade, Desempenho, Segurança e Disponibilidade como atributos de qualidade.

Para representar as decisões arquiteturais definidas, serão utilizados pontos de vista derivados do modelo de visão arquitetural 4+1, que tem o objetivo de descrever o funcionamento de software a partir de múltiplas visões concorrentes. Serão considerados os seguintes pontos de vista:

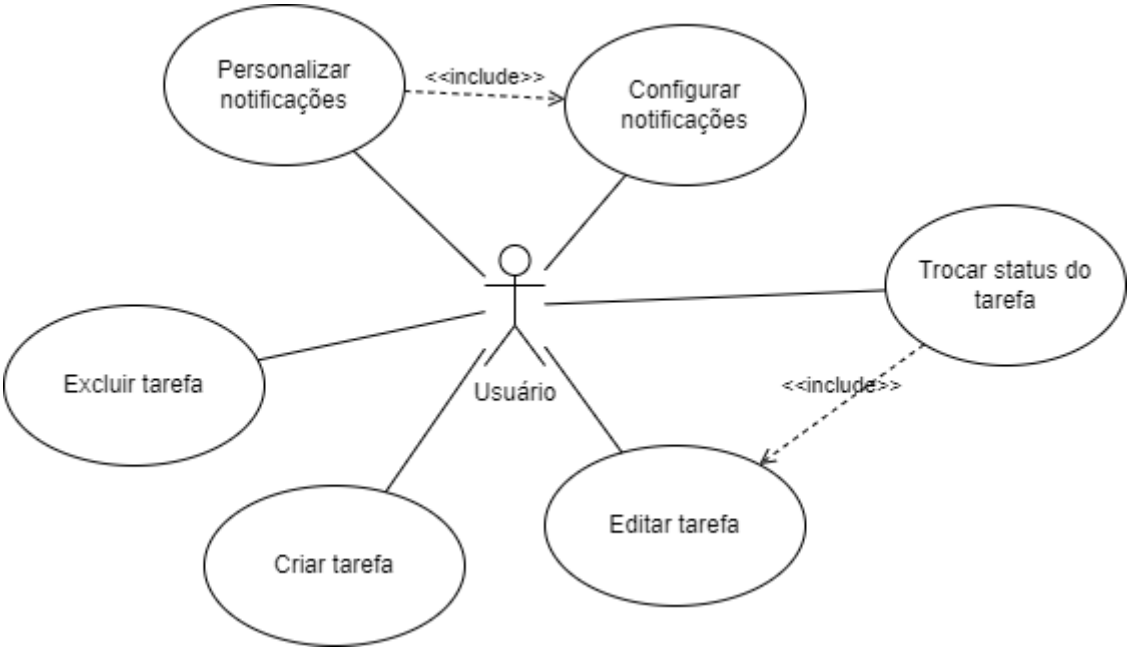
- Ponto de vista dos casos de uso
- Ponto de vista do projetista
- Ponto de vista do desenvolvedor
- Ponto de vista do implantador

4. PONTO DE VISTA DOS CASOS DE USO

4.1. DESCRIÇÃO

A visão de casos de uso é gerada na etapa de análise de requisitos, e é utilizada para fornecer uma base para o planejamento da arquitetura e de todos os outros artefatos que serão criados durante o ciclo de vida do projeto. A visão ilustra os casos de uso e os cenários que englobam o comportamento, as classes e riscos técnicos significativos relativos à arquitetura. A visão de casos de uso não é imutável, ela é refinado e considerada inicialmente em cada iteração do ciclo de vida.

4.2. VISÃO DE CASOS DE USO



O diagrama de casos de uso acima foi criado com base em uma análise dos requisitos funcionais presentes no Documento de Requisitos de Software. Na representação gráfica é possível perceber que o usuário é o ator principal de todos os casos de uso. Além disso, é importante destacar a relação *include* entre **personalizar notificações** e **configurar notificações** e entre **editar lembrete** e **trocar status do lembrete**. Essa relação ocorre quando para que quando um caso de uso for executado, outro caso deverá também ser executado, sempre.

5. PONTO DE VISTA DO PROJETISTA

5.1. VISÃO GERAL

O ponto de vista do projetista é direcionado a equipe de desenvolvimento do sistema, em especial aos projetistas e desenvolvedores, e tem como objetivo definir as principais partes que o compõem, como os componentes e camadas, e quais as suas responsabilidades.

Essa visão foi escolhida por ser uma visão primordial, que permite uma compreensão mais completa do software e de todo o seu ecossistema.

5.2. VISÃO DAS CAMADAS

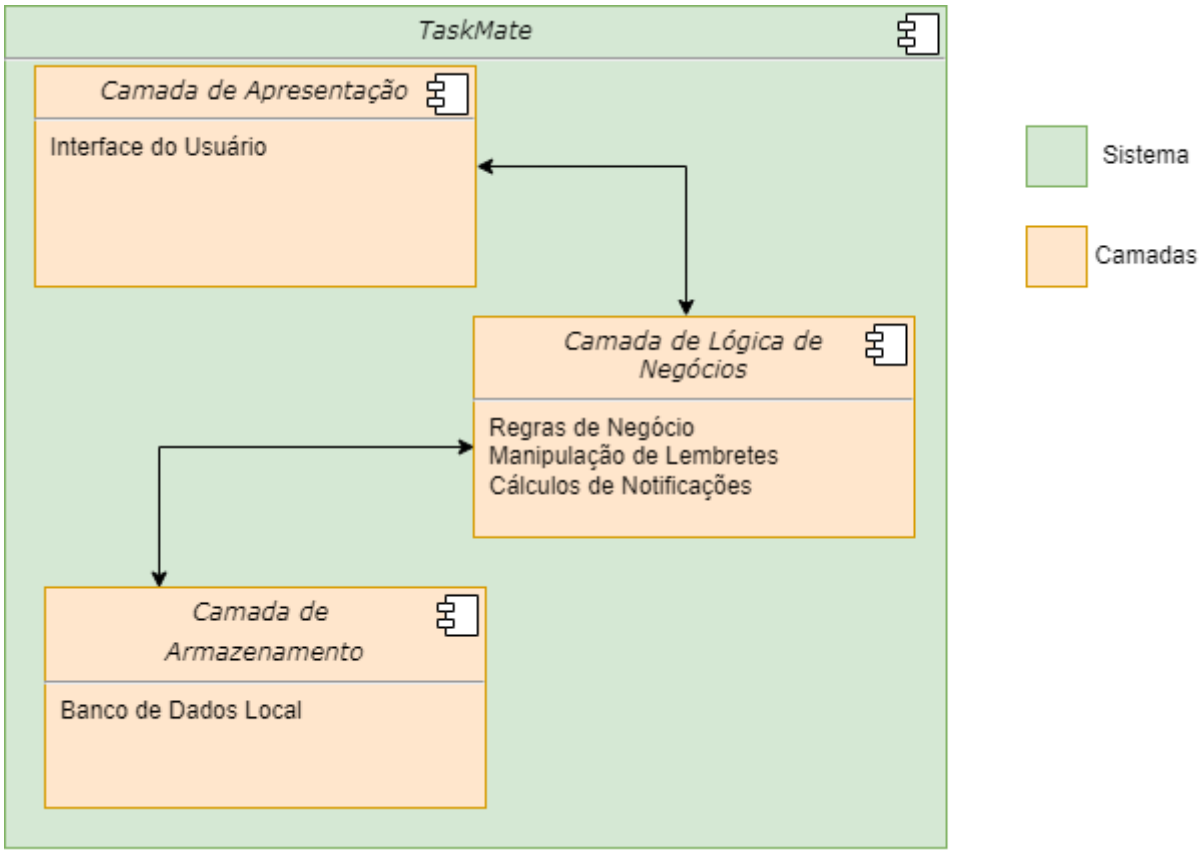
No modelo arquitetural proposto (Padrão em Camadas), o principal componente do sistema são suas camadas. O TaskMate será dividido em 3 (três) camadas, cada uma independente e com suas próprias responsabilidades.

A **Camada de Apresentação** é responsável pela interface do usuário e interação com o usuário. É nessa camada que o usuário pode criar, editar e excluir lembretes, além de personalizar suas notificações.

A Camada de Lógica de Negócios é responsável por lidar com a lógica do sistema, como a manipulação de lembretes, cálculos de datas e horários de notificações. Nesta camada são implementadas as regras de negócio do TaskMate.

A camada de Armazenamento Local é responsável por armazenar todos os dados do sistema, incluindo lembretes, configurações e notificações personalizadas. Essa camada é a responsável pela persistência de dados do sistema.

O Diagrama de componentes abaixo representa visualmente as camadas projetadas para o sistema, suas responsabilidades e como se comunicam.



Com o diagrama acima, é possível perceber que a comunicação entre a camada de *Apresentação* e a camada de Armazenamento Local é intermediada pela camada de *Lógica* de *Negócios*. A camada de apresentação solicita operações à camada de lógica de negócios, que, por sua vez, realiza a manipulação adequada dos dados na camada de armazenamento local.

Essa abordagem ajuda a garantir a separação de responsabilidades e a manter a modularidade e a flexibilidade da arquitetura, permitindo que cada camada seja desenvolvida, testada e mantida de forma independente.

6. PONTO DE VISTA DO DESENVOLVEDOR

6.1. VISÃO GERAL

O ponto de vista do desenvolvedor analista e ilustra o software da perspectiva de um programador, considerando a organização dos módulos, estruturação das classes e comunicação dos componentes, para que todos reflitam a lógica de negócios.

6.2. VISÃO LÓGICA

6.2.1. Detalhamento das classes

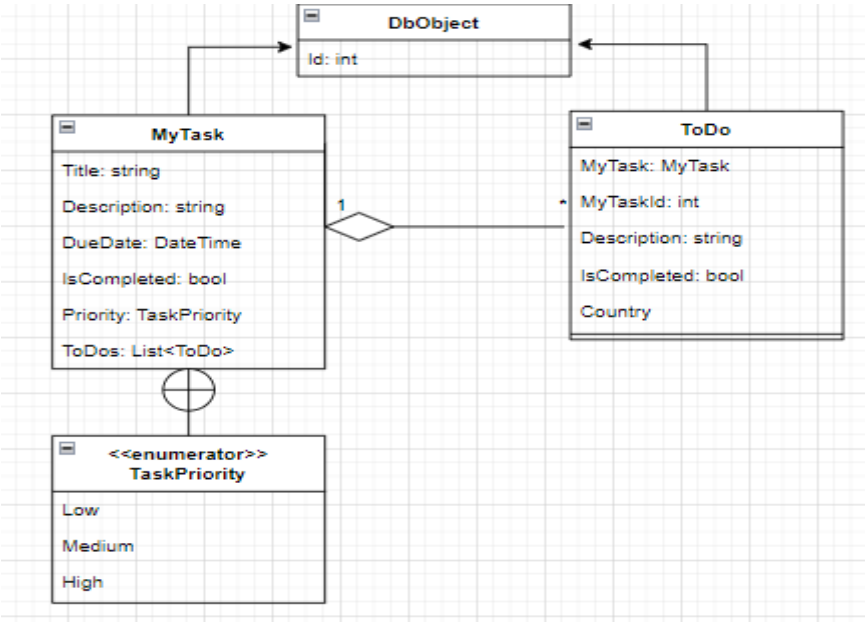
Para o sistema, foram desenvolvidas 3 (três) *classes* e 1 (um) *enumerador* para representação do modelo de negócio. As classes são:

Classe **MyTask**: Representa a tarefa em si e possui as propriedades: título, descrição, data de conclusão, booleano para definir conclusão, prioridade (enumerador) e uma lista de *To-Do's*. Dessa forma, pode-se dizer que a relação entre *MyTask* e *To-Do* é One-To-Many.

Classe **To-Do:** Representa uma etapa da tarefa. Um *to-do* deve estar sempre vinculado à uma tarefa, dessa forma, suas propriedades são: descrição, booleano para definir conclusão e uma única *MyTask* junto com um inteiro *MyTaskId*, que define a relação um para um entre essas duas entidades.

Classe **DbObject:** Possui apenas uma propriedade, um inteiro *Id.* Essa classe é herdada pelas duas classes supracitadas, ou seja, cada uma delas também possui esse inteiro ID. Essa classe é utilizada para fins de persistência e identificação única.

Enumerador **TaskPriority**: Possui 3 (três) valores possíveis, *low, medium* e *high*. Esse enumerador é implementado na classe *MyTask*, para definir a prioridade da tarefa.



6.2.2. Detalhamento da arquitetura

Como dito anteriormente, foi definido o modelo de *Camadas* para esse sistema. Com isso, a arquitetura definida foi a **MVVM**, que é a evolução do MVC. Com essa arquitetura, fica fácil a separação entre a lógica de domínio (**Model)** e a camada de apresentação **(View)**. Utiliza-se a **ViewModel** para realizar o vínculo entre o front-end e back-end.

Cada uma das classes citadas no tópico **6.2.1**, com exceção do *DbObject* que tem fins de modelo relacional, possuem uma **ViewModel** (ex: *MyTaskViewModel*). Nesse ViewModel, realiza-se o vínculo bidirecional entre a exibição e o modelo, podendo ter propriedades únicas para fins apenas visuais.

6.3. VISÃO DE SEGURANÇA

6.3.1. Detalhamento da segurança

Com a utilização do modelo em **Camadas MVVM**, é possível implementar e abordar formas de segurança desde o *Model*, até a *View*.

Com a implementação da *ViewModel*, o acesso ao *Model* fica restringido, o que evita manipulações e vazamentos de dados. Além disso, nessa camada, haverá métodos de validação, para evitar com que dados digitados pelo usuário sejam utilizados para *SQL Injection*.

Para ter acesso ao sistema, seu usuário deve ser autenticado com um e-mail válido.

7. PONTO DE VISTA DO IMPLANTADOR

7.1. VISÃO GERAL

O ponto de vista do implantador concentra-se em como o sistema será implantado em um ambiente de produção, detalhando quais recursos de hardware serão necessários, onde a aplicação será lançada, quais componentes serão distribuídos e em que tipo de máquina esse sistema será utilizado.

7.2. VISÃO FÍSICA

7.2.1. Detalhamento dos nós físicos

O sistema será distribuído pela plataforma Microsoft Store, para que ele possa ser acessado por qualquer máquina Windows e instalado livremente. A utilização dessa plataforma facilita o lançamento de novas versões e atualizações, onde o usuário atualiza automaticamente a aplicação assim que ela for aberta.

Como dito anteriormente, o armazenamento será no banco local do usuário. Utilizaremos também arquivos cache para guardar configurações e definições.