# Especificação de Sistema
# DoATask

**Grupo No. 7**

David Carvalho Nº27973
Diogo Marques Nº.27972
Gabriel Fortes Nº27976
Gonçalo Vidal Nº27984
Diogo Caldas Nº27951
Vasco Gomes Nº27955

**Licenciatura em Engenharia Sistemas Informáticos**

**2ºano**

Barcelos |, 2024

# Índice de tabelas

# Índice de Figuras

# Índice

# 1. Introdução

**DoATask** é uma plataforma desenvolvida na qual facilita e promove a realização de voluntariado dentro das comunidades.

Através da **DoATask** os utilizadores podem criar ou se juntarem-se a comunidades, sejam elas focadas em apoio a instituições de solidariedade social, limpeza de espaços públicos, ou qualquer outra iniciativa de caráter voluntário. Cada comunidade funciona como um espaço virtual onde é possível criar e realizar tarefas voluntarias. Para valorizar a participação e tornar o processo mais motivador, cada tarefa concluída concede ao voluntário pontos e moedas virtuais que podem ser trocadas na loja da Comunidade, oferecendo recompensas e reforçando o ciclo de colaboração.

## 1.1. Objetivo do projeto

O **DoATask** tem como objetivo facilitar o acesso ao voluntariado e incentivar as pessoas a envolverem-se no mesmo. A plataforma foi desenvolvida para simplificar todas as etapas do processo desde encontrar uma tarefa até à sua realização, permitindo que qualquer pessoa, possa contribuir para a sua comunidade. Para incentivar o envolvimento foi desenvolvido um sistema de recompensa com pontos e moedas que podem ser trocados por itens na loja, criando um ambiente que valoriza cada ação solidária e encoraja uma participação das pessoas.

## 1.2. Problemas a resolver

Atualmente, uma das maiores dificuldades para quem quer fazer voluntariado é a falta de um ponto central onde possam ser consultadas, de forma clara e organizada, as várias oportunidades disponíveis. As tarefas voluntárias estão frequentemente espalhadas por diferentes meios, redes sociais, sites de associações ou grupos informais, o que obriga os interessados a procurar ativamente em múltiplas fontes, muitas vezes sem saber por onde começar. Esta dispersão dificulta a adesão, reduz a visibilidade de muitas iniciativas e acaba por afastar potenciais voluntários que, apesar da vontade de ajudar, não encontram facilmente onde ou como fazê-lo. Além disso, a ausência de um sistema comum que valorize as contribuições de cada pessoa contribui para uma menor motivação e continuidade nas ações de voluntariado.

Por outro lado, também não existem ferramentas para quem quer criar uma comunidade de voluntariado ou criar as suas próprias tarefas de voluntariado. Muitos grupos informais ou pessoas com ideias para ajudar acabam por não avançar por falta de uma plataforma que ajude a centralizar as atividades de voluntariado. Esta ausência de suporte tecnológico compromete a mobilização de novas iniciativas e limita o crescimento de redes de solidariedade locais.

## 1.1. Solução proposta

O **DoATask** resolve estes problemas ao oferecer uma plataforma que centraliza todas as oportunidades de voluntariado e simplifica a criação de novas comunidades e tarefas. Os utilizadores encontram, num só lugar, um catálogo organizado de ações disponíveis, filtrável por comunidade. Quem pretende criar iniciativas tem acesso a ferramentas intuitivas para abrir comunidades. Um sistema de notificações que informa sobre os vários estados das atividades de voluntario. Além disso, o **DoATask** valoriza cada contribuição através da atribuição de pontos e moedas que podem ser trocadas na loja da comunidade valorizando o esforço de cada voluntário e incentivando a participação contínua. Desta forma, o **DoATask** oferece uma solução completa e acessível tanto para quem quer ajudar como para quem organiza iniciativas solidárias, fomentando a criação e o fortalecimento de redes de voluntariado locais.

# 2. Processos de negócio

## 2.1. Processo da tarefa de voluntariado

### Etapas

- **Criar uma tarefa:** O utilizador cria uma tarefa na plataforma, especificando os detalhes da mesma, titulo, descrição, localização, dificuldade, comunidade na qual a tarefa será criada e opcionalmente imagens.
- **Eliminar ou não a tarefa:** Caso o utilizador quiser eliminar a tarefa criada pode elimina-la, mas somente se nenhum utilizador aceitou a mesma.
- **Aceitar a tarefa:** Se o utilizador não tiver outra tarefa em curso nessa comunidade, pode aceitar a tarefa.
- **Cancelar ou não a tarefa:** Caso o utilizador quiser cancelar a tarefa pode cancela-la.
- **Terminar a tarefa:** O utilizador termina a tarefa
- **Avaliar a tarefa:** O utilizador avalia a tarefa
- **Receber recompensa:** O utilizador recebe as recompensas em pontos e moedas.

### Descrição do processo

Um utilizador pode criar uma tarefa, a tarefa fica então visível para todos os membros da comunidade. Enquanto ninguém tiver aceite a tarefa, o criador pode apagá-la. No entanto, assim que for aceite por outro utilizador, já não pode ser eliminada. Quando a tarefa for terminada o utilizador avalia a mesma.

Os voluntários podem ver as tarefas disponíveis e, se não tiverem nenhuma em andamento nessa comunidade, podem aceitar uma. Se mudarem de ideia antes de a realizar, podem cancelar a aceitação. Depois de concluírem a tarefa, marcam-na como terminada e têm a opção de a avaliar. Ao finalizar, recebem automaticamente pontos e moedas como recompensa.

## 2.2. Processo da comunidade

### Etapas

- **Criar uma comunidade:** Um utilizador cria uma comunidade especificando o nome e a localidade da mesma, sendo que o utilizador têm que pertencer a localidade que especificar.
- **Entrar na comunidade:** Um utilizador entra numa comunidade caso possuía morada na localidade a que a comunidade pertence.

### Descrição do processo

O utilizador pode criar uma nova comunidade ao indicar o nome e a localidade da mesma. Para isso, é necessário que o utilizador pertença à localidade que está a registar. Outros utilizadores podem entrar nessa comunidade desde que também tenham morada na mesma localidade.

## 2.3. Processo da loja

### Etapas

- **Criar item:** Utilizador cria um item para ser disponibilizado na loja da sua comunidade
- **Esconder ou Mostrar item:** Utilizador pode esconder o item caso o mesmo esteja disponível na loja ou pode mostrar o item caso o mesmo não esteja disponível na loja.
- **Comprar o item:** Utilizador compra o item caso possuía moedas suficientes

### Descrição do processo

Um utilizador pode criar um item para a loja da sua comunidade, depois de criado, o item pode ser colocado visível ou invisível na loja, o utilizador pode escondê-lo a qualquer momento, ou voltar a mostrá-lo quando quiser que fique novamente disponível. Qualquer membro da comunidade pode comprar o item, desde que tenha moedas suficientes.

# 3. Requisitos funcionais

Os requisitos funcionais descrevem o comportamento esperado da aplicação e as funcionalidades que devem ser implementadas para satisfazer as necessidades dos utilizadores.

| Código | Regra |
|---|---|
| **RF 01** | O sistema deve permitir que um utilizador crie tarefas dentro de uma comunidade à qual pertence. |
| **RF 02** | O sistema deve permitir que o criador elimine uma tarefa, desde que ainda não tenha sido aceite por nenhum voluntário. |
| **RF 03** | O sistema deve permitir que um utilizador aceite uma tarefa, desde que não tenha outra tarefa em andamento na mesma comunidade. |
| **RF 04** | O sistema deve permitir que o utilizador cancele uma tarefa aceite, antes da sua conclusão. |
| **RF 05** | O sistema deve permitir que o utilizador marque a tarefa como concluída. |
| **RF 06** | O sistema deve permitir que o utilizador avalie a tarefa após a sua conclusão. |
| **RF 07** | O sistema deve atribuir pontos e moedas automaticamente ao utilizador após concluir uma tarefa. |
| **RF 08** | O sistema deve permitir que um utilizador crie uma comunidade, definindo o nome e a localidade da mesma. |
| **RF 09** | O sistema deve verificar se o utilizador pertence à localidade indicada antes de permitir a criação da comunidade. |
| **RF 10** | O sistema deve permitir que um utilizador entre numa comunidade, caso a sua morada corresponda à localidade da mesma. |
| **RF 11** | O sistema deve listar as comunidades disponíveis de acordo com a morada do utilizador. |
| **RF 12** | O sistema deve permitir que o utilizador crie um item na loja da sua comunidade. |
| **RF 13** | O sistema deve permitir que o utilizador esconda ou mostre um item na loja. |
| **RF 14** | O sistema deve permitir que o utilizador compre um item, desde que tenha moedas suficientes e pertença a comunidade da loja que o item pertence. |
| **RF 15** | O sistema deve atualizar automaticamente o saldo de moedas do utilizador após a compra de um item. |

| | |
|---|---|
| **RF 16** | O sistema deve contabilizar os pontos e moedas recebidos por cada utilizador após completar tarefas. |
| **RF 17** | O sistema deve permitir ao utilizador visualizar o seu saldo de pontos e moedas. |

*Tabela 1- Requisitos Funcionais*

# 4. Requisitos não funcionais

Os requisitos não funcionais descrevem as características de qualidade que a aplicação deve cumprir, sem estarem diretamente relacionadas com funcionalidades específicas.

| Código | Regra |
|---|---|
| **RNF 01** | A interface do sistema deve ser simples, intuitiva e acessível para todos utilizadores. |
| **RNF 02** | O sistema deve disponibilizar mensagens de erro claras e orientadoras sempre que ocorrer uma falha. |
| **RNF 03** | O sistema deve validar os dados introduzidos pelo utilizador para prevenir injeções de código malicioso. |
| **RNF 04** | O sistema deve garantir que apenas utilizadores autenticados possam criar, aceitar ou gerir tarefas e comunidades. |
| **RNF 05** | O sistema deve ter um mecanismo de recuperação de conta. |
| **RNF 06** | O sistema deve possuir autenticação. |

*Tabela 2 - Requisitos não funcionais*

# 5. Project Backlog

O Project Backlog representa as funcionalidades, requisitos e tarefas que devem ser implementadas na aplicação e tem como objetivo organizar e descrever, de forma clara, as ações que o sistema deve permitir, baseando-se nas necessidades dos utilizadores.

| ID | Como Utilizador | Pretendo |
|---|---|---|
| **1** | Utilizador não autenticado | Registar |
| **2** | Utilizador não autenticado | Login |
| **3** | Utilizador autenticado | Atualizar os dados da minha conta |
| **4** | Utilizador autenticado | Criar uma comunidade |
| **5** | Utilizador autenticado | Entrar numa comunidade |
| **6** | Utilizador autenticado | Criar uma tarefa |
| **7** | Utilizador autenticado e criador da tarefa | Eliminar uma tarefa |
| **8** | Utilizador autenticado | Aceitar tarefa |
| **9** | Utilizador autenticado | Cancelar Tarefa |
| **10** | Utilizador autenticado | Terminar Tarefa |
| **11** | Utilizador autenticado e criador da tarefa | Avaliar tarefa |
| **12** | Utilizador autenticado e dono da comunidade | Criar um item para a loja |
| **13** | Utilizador autenticado e dono da comunidade | Esconder ou mostrar item na loja |
| **14** | Utilizador autenticado | Comprar um item na loja |
| **15** | Utilizador autenticado | Consultar o meu saldo de moedas e pontos |

*Tabela 3 - Project Backlog*

# 6. Arquitetura do sistema

## 6.1. Frontend

O **frontend** foi construído com Next.js, uma framework baseada em React, a frontend comunica-se com a backend chamando os endpoitnts para obter e manipular os dados.

## 6.2. Backend

A backend é constituída por vários serviços que em conjunto garantem a funcionalidade do sistema:

- **NestJs**: Tem como função processar os pedidos recebidos do frontend, tratar da lógica da aplicação, fazer a gestão dos dados e comunicar-se com a base de dados
- **Supabase Auth**: Responsável pela criação de contas, login e gestão das sessões dos utilizadores.
- **Supabase Storage:** Responsável por guardar ficheiros. Estes ficheiros são armazenados e acedidos através de URLs gerados automaticamente.
- **Prisma:** Utilizado em conjunto com o NestJS, para gerir todas as interações com a base de dados.

## 6.3. Base de Dados PostgreSQL

A aplicação utiliza uma base de dados PostgreSQL, fornecida pelo serviço Supabase, para armazenar todas as informações da aplicação. O PostgreSQL é um sistema de gestão de base de dados relacional, que permite realizar operações complexas de forma eficiente, garantindo a integridade dos dados e o bom desempenho da aplicação.

## 6.4. Base de Dados PostgreSQL

1. **Interação do utilizador:** O utilizador acede á aplicação, carregando paginas e componentes contruídos com NextJS.

2. **Pedido REST ao Backend:** Quando o utilizador por exemplo cria uma tarefa, Nextjs envia uma requisição HTTP para o endpoint correspondente no NestJS, incluindo o token JWT em cookie para autenticação.

3. **Autenticação:** O Nestjs recebe o pedido, valida o JWT junto ao serviço de autenticação do Supabase, garantindo que o utilizador está autenticado e autorizado para executar aquela ação.

4. **Acesso à base de dados:** Ainda no NestJs, o serviço correspondente utiliza o Prisma para interagir com a base de dados PostgreSQL, executando operações de leitura ou escrita.

5. **Armazenamento de Ficheiros:** O NestJS recebe o ficheiro, caso o mesmo exista, e utiliza o serviço de Storage do Supabase para guarda-lo, se o pedido da frontend for para receber o serviço gera e retorna um URL.

6. **Resposta ao frontend:** O resultado do pedido é devolvido pelo prisma ou pelo serviço Supabase, dependendo do pedido do utilizador, o NestJS formata uma resposta JSON adequada e envia-a de volta á NextJS.

# 7. Diagramas casos de uso

Neste capítulo são apresentados os diagramas de casos de uso da aplicação DoATask, que representam as principais interações entre os utilizadores e o sistema.

## 7.1. Interações do Sistema

O diagrama de casos de uso "Interações do Sistema" representa as principais funcionalidades disponíveis na aplicação DoATask, divididas consoante o tipo de utilizador. Este diagrama mostra de forma clara quais as ações que um utilizador não autenticado e autenticado pode realizar. Este diagrama ajuda a visualizar os diferentes níveis de acesso dentro da aplicação e a organizar as interações consoante o estado de autenticação do utilizador.

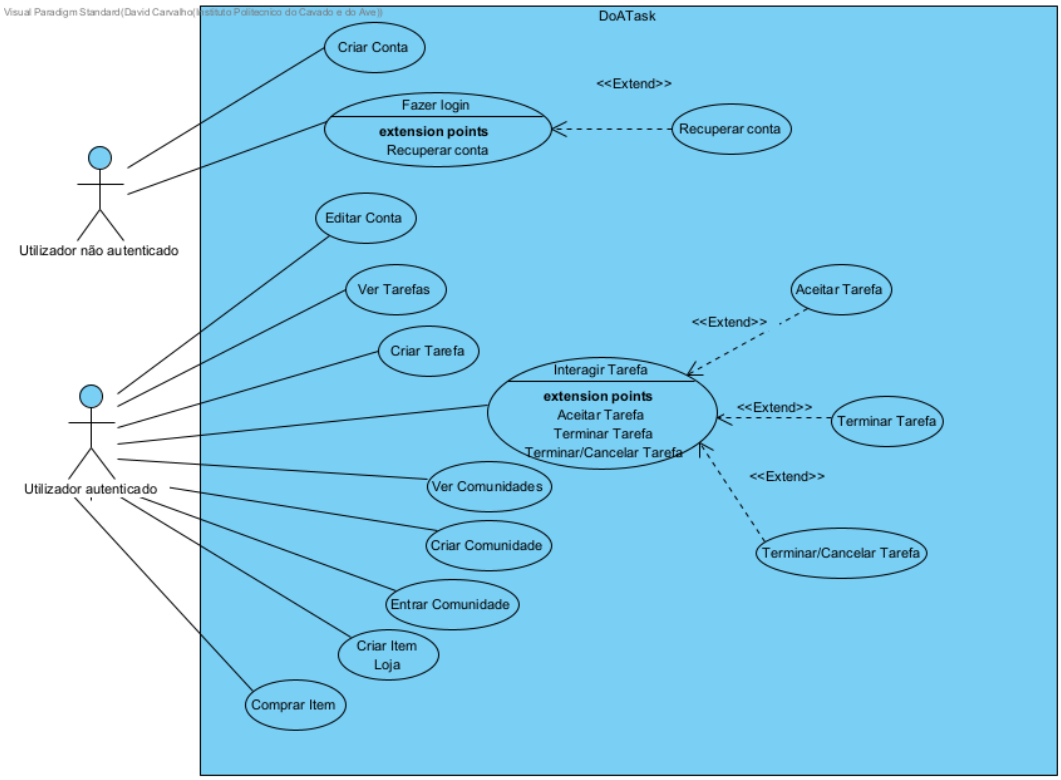



*Figura 1- Diagrama Caso de Uso Interações do Sistema*

## 7.2. Utilizador não autenticado

Um **utilizador não autenticado** tem acesso limitado à aplicação e apenas pode realizar ações básicas relacionadas com a entrada no sistema.

- **Criar Conta:** Permite ao utilizador registar-se na plataforma.
- **Fazer Login:** Permite iniciar sessão na conta existente.
- **Recuperar Conta:** Caso o utilizador tenha esquecido as credenciais, pode aceder à funcionalidade de recuperação de conta.

## 7.3. Utilizador autenticado

Depois de iniciar sessão, o utilizador ganha acesso a um conjunto mais alargado de funcionalidades. Estas incluem:

- **Editar Conta**: Alterar os dados do seu perfil.
- **Ver Tarefas**: Ver as tarefas disponíveis por comunidade.
- **Criar Tarefa**: Criar uma nova tarefa voluntária numa comunidade.
- **Interagir com Tarefa**: Envolve diferentes ações:
- **Aceitar Tarefa:** Aceitar tarefa voluntaria
- **Terminar Tarefa:** Terminar tarefa voluntaria
- **Cancelar Tarefa**: Cancelar tarefa voluntaria
- **Ver Comunidades**: Ver as comunidades disponíveis.
- **Criar Comunidade**: Criar uma nova comunidade, caso o utilizador pertença à mesma localidade.
- **Entrar Comunidade**: Juntar-se a uma comunidade da localidade do utilizador.
- **Criar Item**: Adicionar um item à loja da comunidade do utilizador.
- **Comprar Item**: Comprar itens na loja de uma comunidade.

# 8. Diagramas BPMN

Neste capítulo são apresentados os diagramas BPMN que servem para representar os processos de negócio da aplicação DoATask. Estes diagramas permitem descrever, de forma clara e padronizada, o fluxo de atividades realizadas pelos utilizadores dentro do sistema.

## 8.1. Diagrama BPMN da loja

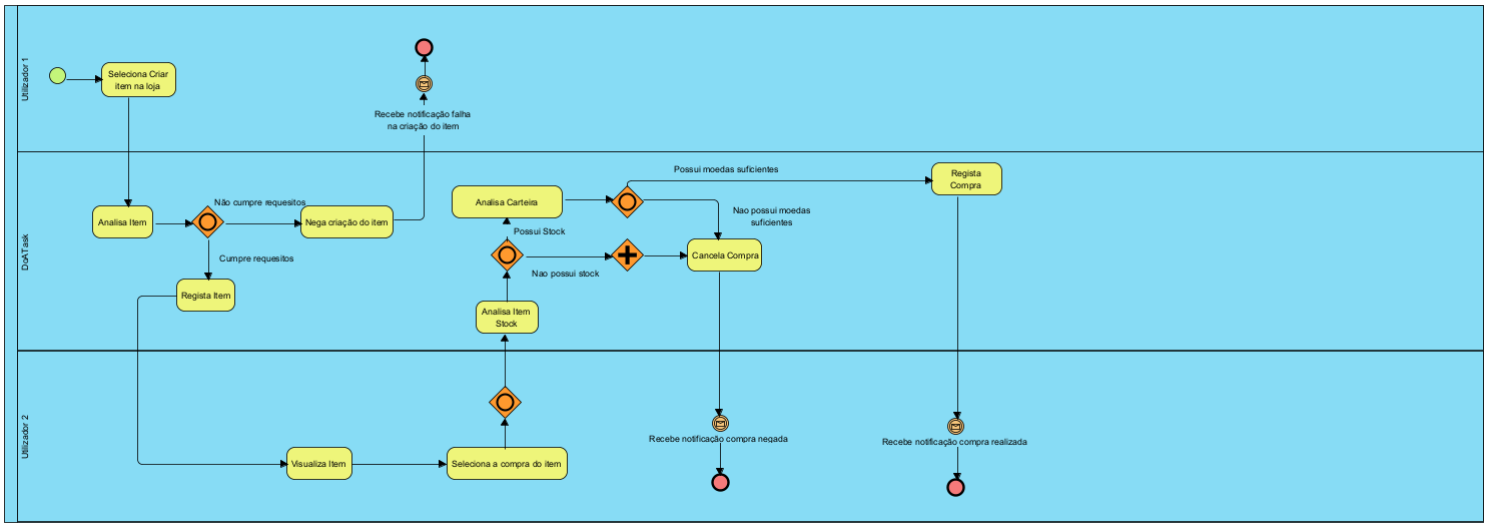



*Figura 2 - Diagrama BPMN da Loja*

### Participantes do processo

- **Utilizador 1:** Responsável por iniciar o processo ao criar um item na loja
- **DoATask:** Responsável por registar item e compra e por validar pedidos.
- **Utilizador 2:** Responsável por selecionar o item para comprar.

### Descrição do processo

O processo inicia-se quando o **Utilizador 1** seleciona a opção de criar um item na loja. O sistema analisa o item submetido para verificar se este cumpre os requisitos. Se o item não cumprir os critérios definidos, o sistema recusa a criação do item e envia uma notificação de falha ao utilizador. Caso os requisitos estejam corretos, o item é registado com sucesso na loja.

Posteriormente, o **Utilizador 2** pode visualizar os itens disponíveis na loja e selecionar aquele que pretende adquirir. Após essa seleção, o sistema analisa se o item possui stock suficiente e verifica a carteira do utilizador para garantir que existem moedas suficientes para a compra.

Se o utilizador tiver saldo suficiente e o item estiver em stock, a compra é registada com sucesso e o utilizador recebe uma notificação a confirmar a realização da compra. Caso o utilizador não tenha moedas suficientes, o sistema cancela a transação e envia uma notificação informando que a compra foi negada. da mesma forma, se não houver stock disponível, a compra é igualmente cancelada.

## 8.2. Diagrama BPMN das tarefas

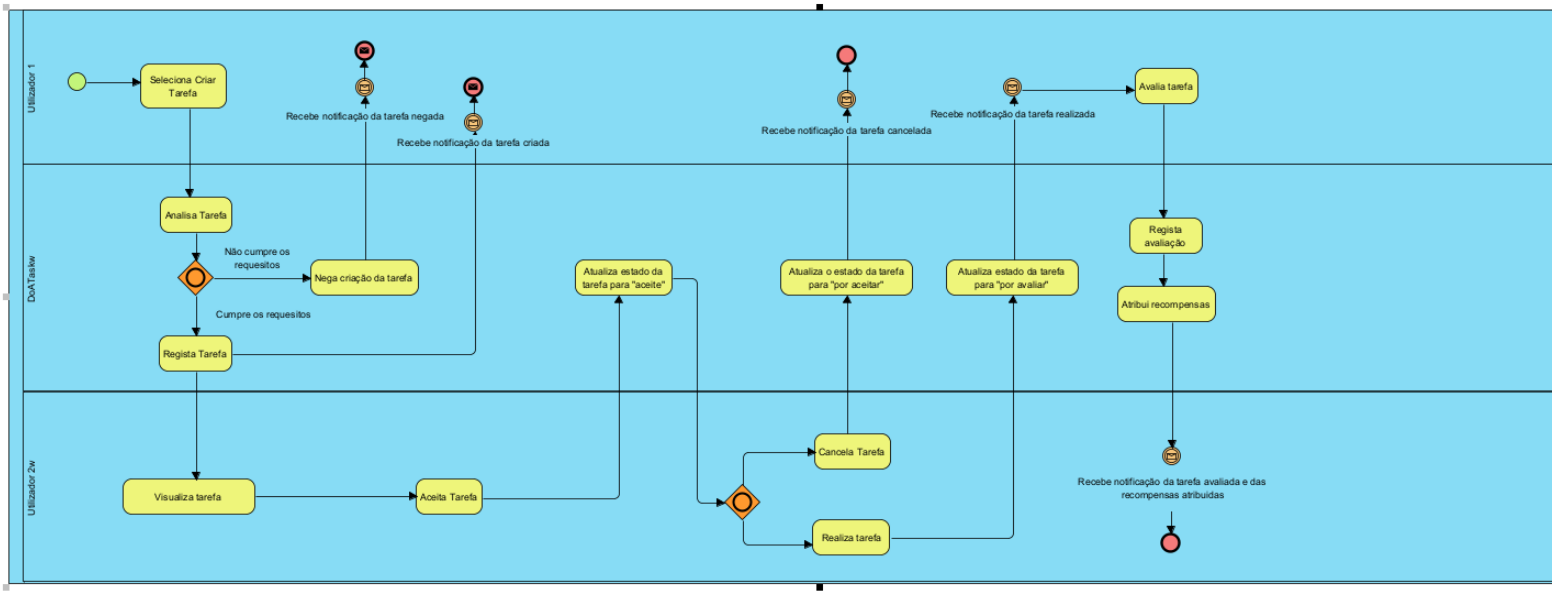



*Figura 3 - Diagrama BPMN das Tarefas*

### Participantes do processo

- **Utilizador 1:** Responsável por iniciar o processo ao selecionar criar tarefa
- **DoATask:** Responsável por registar tarefa, registar avaliação, atualizar o estado da tarefa, atribuir recompensas e validar pedidos.
- **Utilizador 2:** Responsável por visualizar, aceitar, cancelar e realizar tarefa.

### Descrição do processo

O processo inicia-se com o **Utilizador 1** a selecionar a opção de criar uma tarefa. A plataforma analisa os dados submetidos. Se a tarefa não cumprir os requisitos definidos, o sistema rejeita a criação e envia uma notificação ao utilizador, caso a tarefa esteja conforme, esta é registada com sucesso e o Utilizador 1 é notificado da criação.

Em seguida, o Utilizador 2 visualiza a tarefa disponíveis e aceita a tarefa, o sistema atualiza o estado da tarefa para "em andamento" e envia a respetiva notificação ao Utilizador 1.

Durante a execução da tarefa, o Utilizador 2 pode cancelar a tarefa (o estado é atualizado para "cancelada") ou concluir a tarefa (estado atualizado para "realizada"). Após a realização da tarefa, a plataforma notifica o utilizador 1 e o mesmo avalia a tarefa, o sistema regista a avaliação e atribui as recompensas ao Utilizador 2 conforme a avaliação. Para os utilizadores recebem notificações sobre a conclusão da tarefa e sobre as recompensas atribuídas.

## 8.3. Diagrama BPMN das comunidades

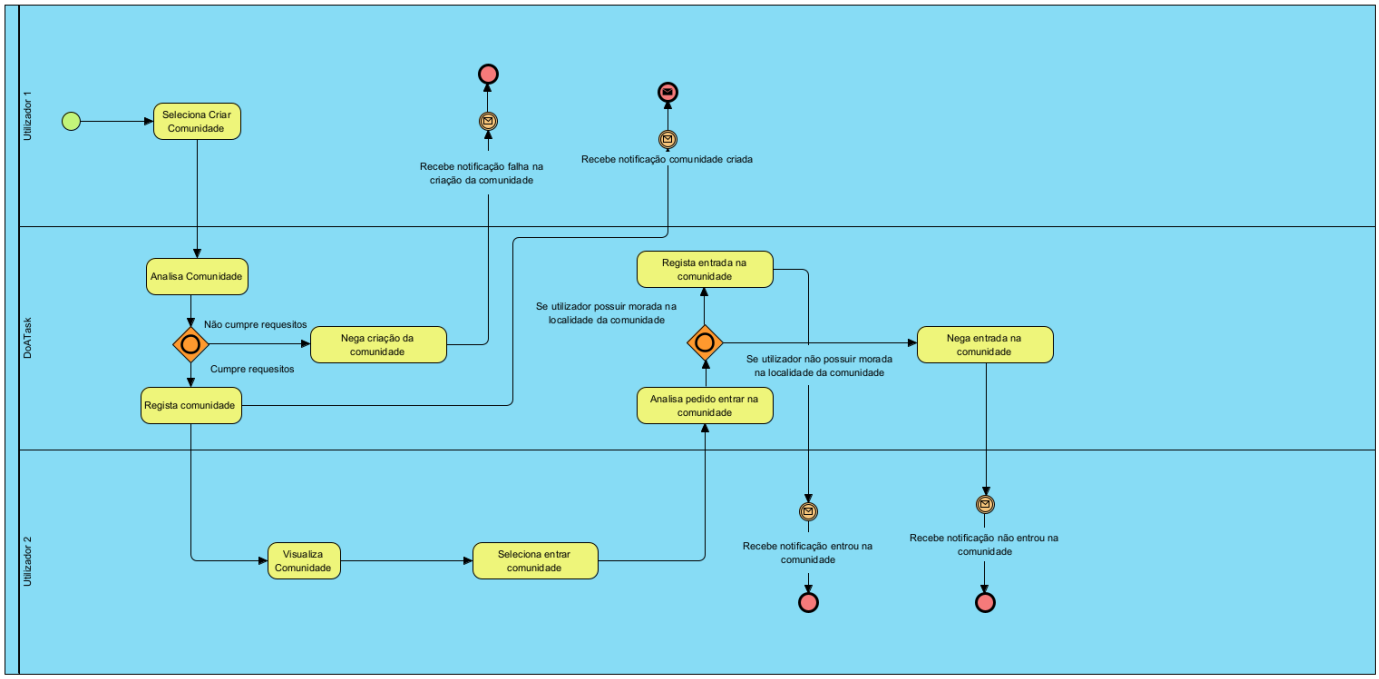



*Figura 4- Diagrama BPMN das Comunidades*

### Participantes do processo

- **Utilizador 1:** Responsável por iniciar o processo ao selecionar criar comunidade
- **DoATask:** Responsável por registar comunidade, registar entrada na comunidade e validar pedidos.
- **Utilizador 2:** Responsável por visualizar, e entrar na comunidade.

### Descrição do processo

O processo inicia-se com o **Utilizador 1** a selecionar a opção de criar uma comunidade. A plataforma analisa os dados submetidos e verifica se a proposta cumpre os requisitos, caso não cumpra, o sistema rejeita a criação da comunidade e envia uma notificação ao Utilizador 1 informando da falha. Se os requisitos forem cumpridos, a comunidade é registada com sucesso e o Utilizador 1 é notificado da sua criação.

Após a criação, o Utilizador 2 pode visualizar as comunidades disponíveis e, se assim desejar, selecionar a opção de entrar numa comunidade. A plataforma analisa o pedido, validando se o Utilizador 2 possui morada na localidade correspondente à comunidade. Se não possuir, a entrada é recusada e o utilizador recebe uma notificação indicando que não foi possível entrar na comunidade. Caso o critério seja cumprido, o sistema regista a entrada do Utilizador 2 na comunidade e envia uma notificação a confirmar o sucesso da operação.

# 9. Diagrama ER

O **Diagrama Entidade-Relacionamento** representa, de forma estruturada, a organização lógica dos dados da aplicação. Este diagrama permite identificar as entidades principais do sistema, os seus atributos e os relacionamentos existentes entre elas. Através deste modelo, é possível garantir a coerência e integridade dos dados, servindo de base para a implementação da base de dados. O diagrama foi construído tendo em conta os requisitos funcionais da aplicação, assegurando que todas as interações e dependências entre utilizadores, comunidades, tarefas, lojas, compras e notificações estejam corretamente modeladas.

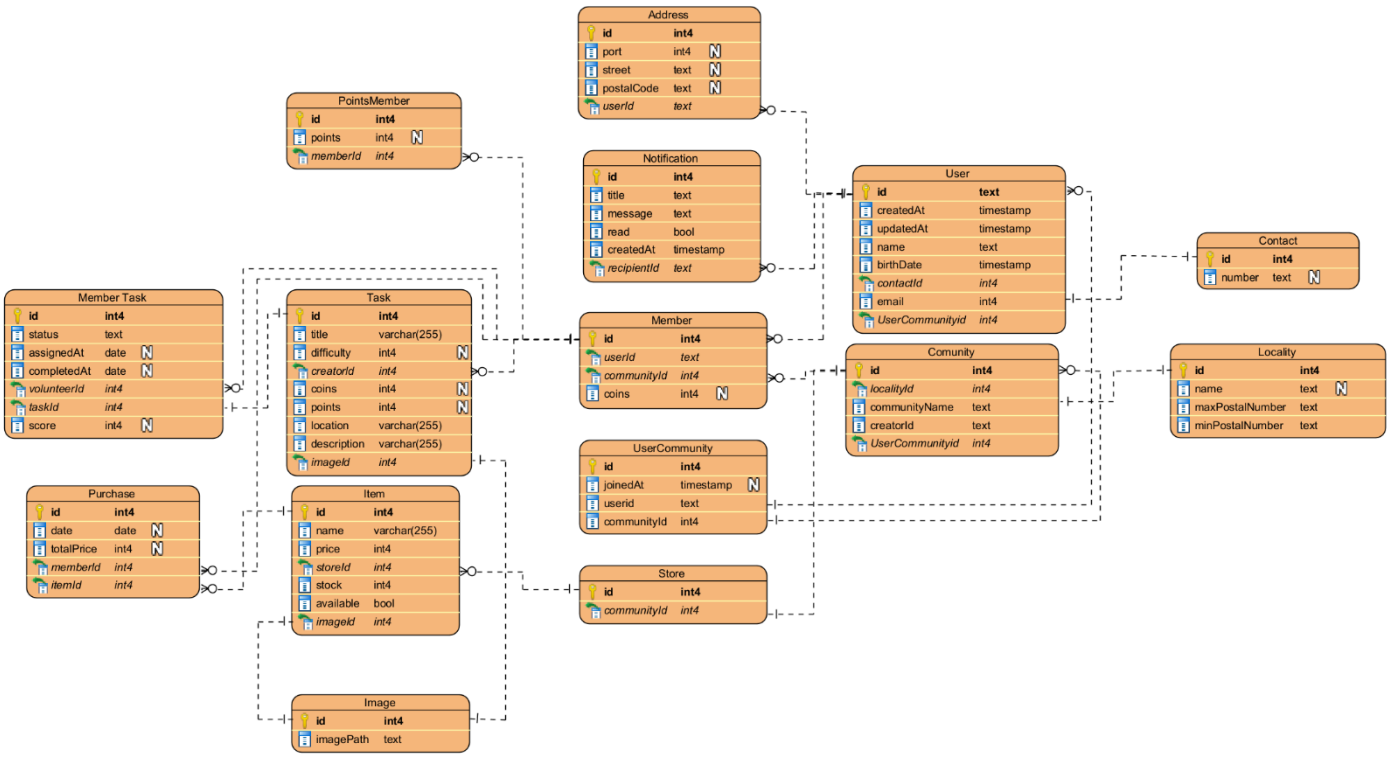



*Figura 5- Diagrama ER*

## 9.1. Utilizador e Membro

A entidade User armazena os dados pessoais do utilizador, incluindo o nome, data de nascimento, email, data de criação e data de atualização da conta. Está associada à tabela **Address**, permitindo que cada utilizador possua múltiplos endereços. Possui também uma ligação à tabela **Contact**, onde é armazenado o seu número de contacto, e à tabela **Notification**, que regista todas as notificações enviadas ao utilizador. Adicionalmente, a tabela **UserCommunity** está associada ao User e é criada sempre que um utilizador entra numa comunidade, permitindo rastrear essa associação.

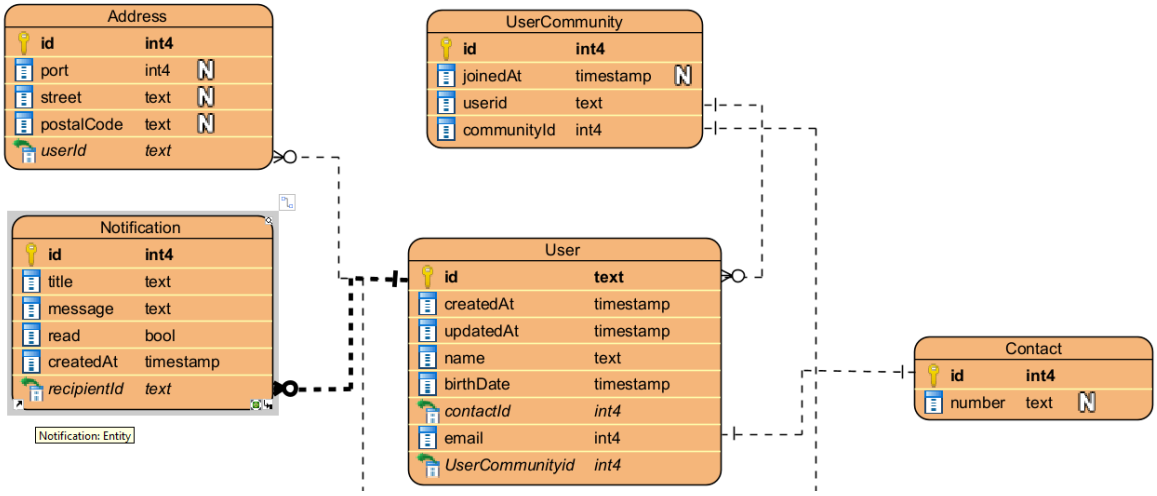



*Figura 6- Diagrama ER Utilizador*

Cada membro está associado a um utilizador e a uma comunidade específica, possuindo um saldo de moedas próprio. Como um utilizador pode integrar várias comunidades, é criado um novo registo na tabela **Member** sempre que este adere a uma comunidade diferente. Desta forma, tanto o saldo de moedas como os pontos são geridos de forma independente em cada comunidade, não estando concentrados num único registo de utilizador. Adicionalmente, a tabela **PointsMember** encontra-se ligada à tabela **Member**, permitindo armazenar separadamente os pontos acumulados por cada membro.

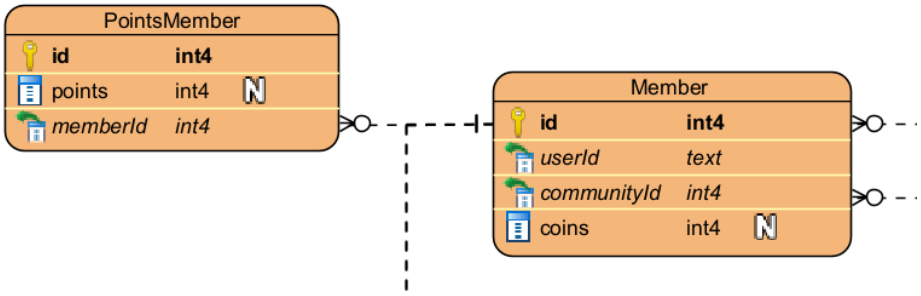



*Figura 7- Diagrama ER Membro*

## 9.2. Comunidade

A entidade **Community** contém o nome da comunidade e o identificador do seu criador. Está também associada à tabela **Locality**, que define a localidade da comunidade através de um nome e de um intervalo de códigos postais, permitindo estabelecer os seus limites geográficos.

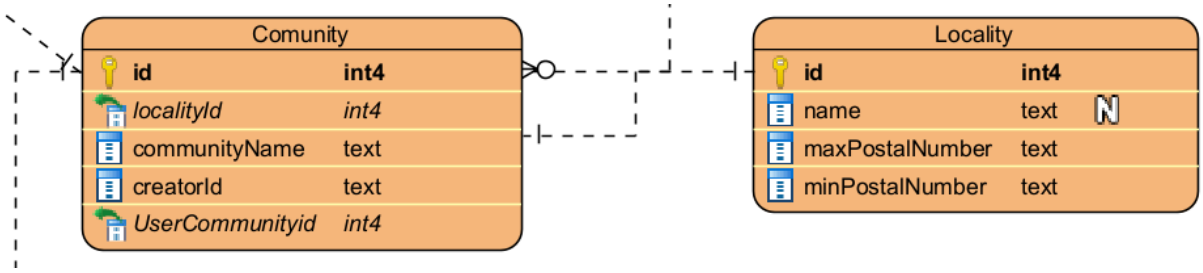



*Figura 8 - Diagrama ER Comunidade*

## 9.3. Tarefa

A entidade **Task** representa uma tarefa e contém informações como o título, grau de dificuldade, recompensas atribuídas (em moedas e pontos), localização e descrição. Está associada à entidade **Member**, que identifica o membro responsável pela criação da tarefa, e à entidade **Image**, que armazena o caminho da imagem associada à tarefa. Além disso, a **Task** está ligada à tabela **MemberTask**, a qual é criada apenas quando um membro aceita realizar uma tarefa. Esta última tabela regista o estado da execução da tarefa por parte do voluntário, permitindo acompanhar o seu progresso e desempenho.

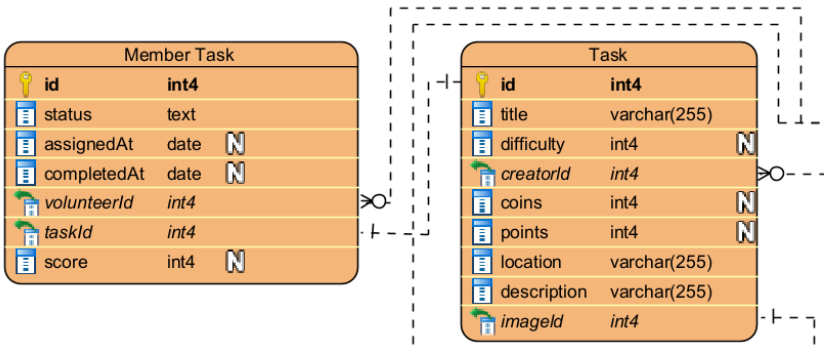
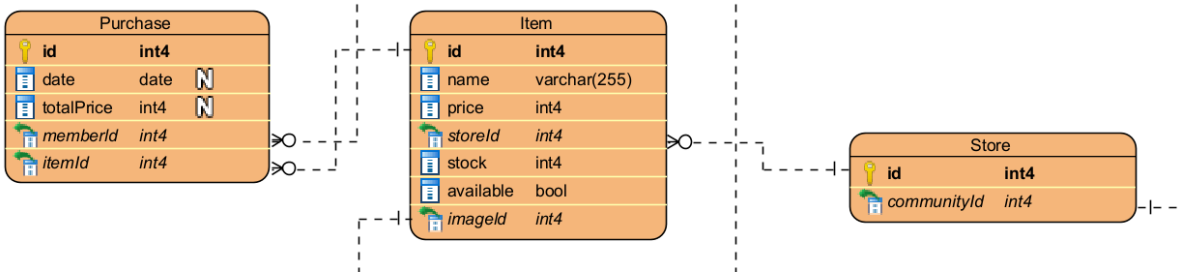



*Figura 9 - Diagrama ER Tarefa*

## 9.4. Loja

A entidade **Store** está associada a uma comunidade, sendo que cada comunidade possui apenas uma loja. A tabela Item está ligada à **Store**, permitindo que uma loja tenha vários itens associados. A entidade **Item** armazena as informações de cada produto, incluindo o nome, preço, stock e um indicador de disponibilidade. Por fim, a tabela **Purchase** representa as compras realizadas pelos membros e está ligada à tabela Item, identificando qual item foi adquirido. Cada registo de compra inclui a data da transação, o preço da compra e a referência ao membro que a efetuou.



# 10. Diagrama de classes

O diagrama de classes apresentado nesta secção mostra a estrutura base do sistema, focando-se nas classes, nos seus atributos e nas relações entre elas, como heranças, associações e cardinalidades.

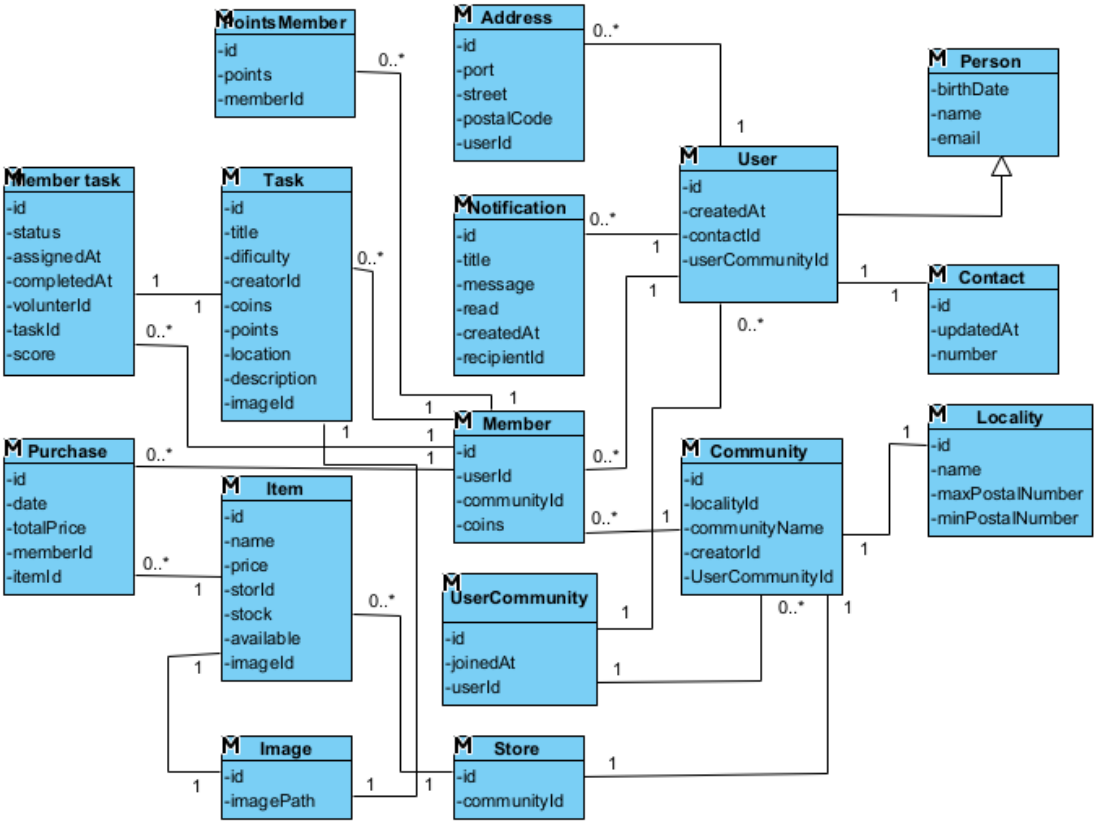



*Figura 10 - Diagrama de Classes*

## 10.1. Gestão do utilizador

O pacote **User Managment** gere as entidades relacionadas com o utilizador.

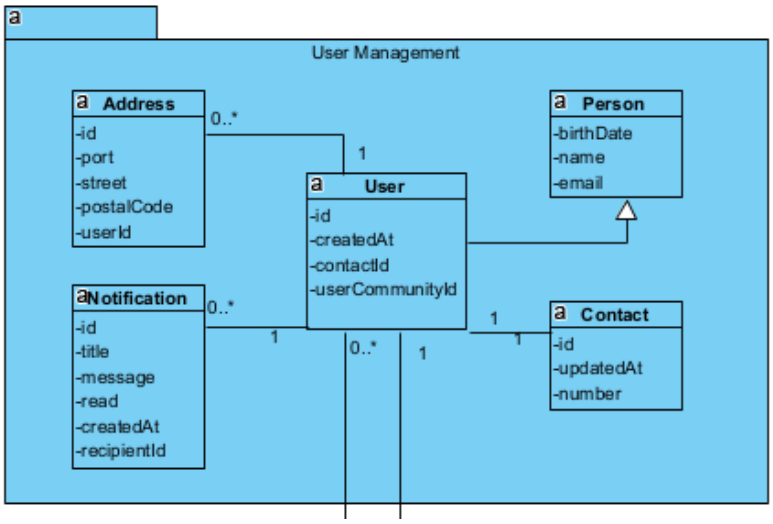



*Figura 11 - DIagrama Classes - User Management*

O pacote **UserManagement** inclui um conjunto de controladores responsáveis por implementar os métodos necessários à gestão das respetivas entidades. Estes controladores permitem realizar operações como a criação, atualização, consulta e eliminação de utilizadores, contactos, endereços e notificações, assegurando a correta manutenção dos dados associados ao utilizador.

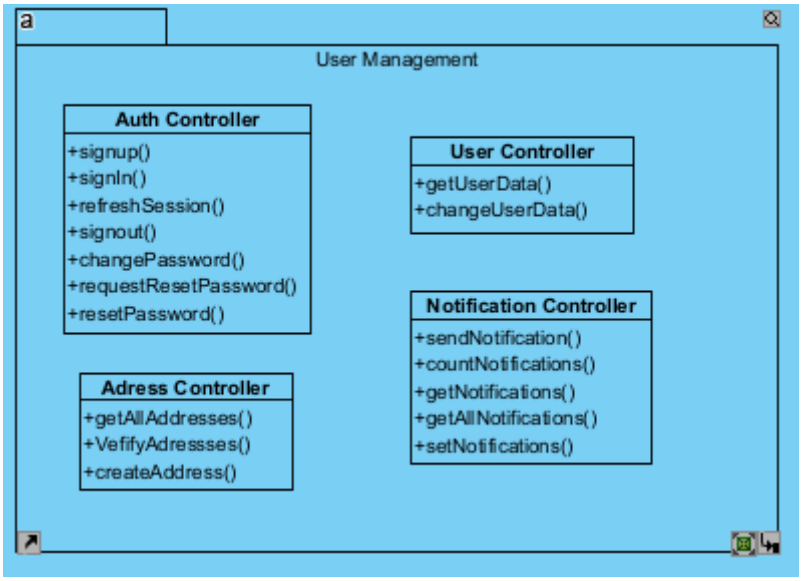



*Figura 12- DIagrama Classes User Management Controllers*

## 10.2. Gestão do membro

O pacote **Member Management** gera as entidades relacionadas com o membro .

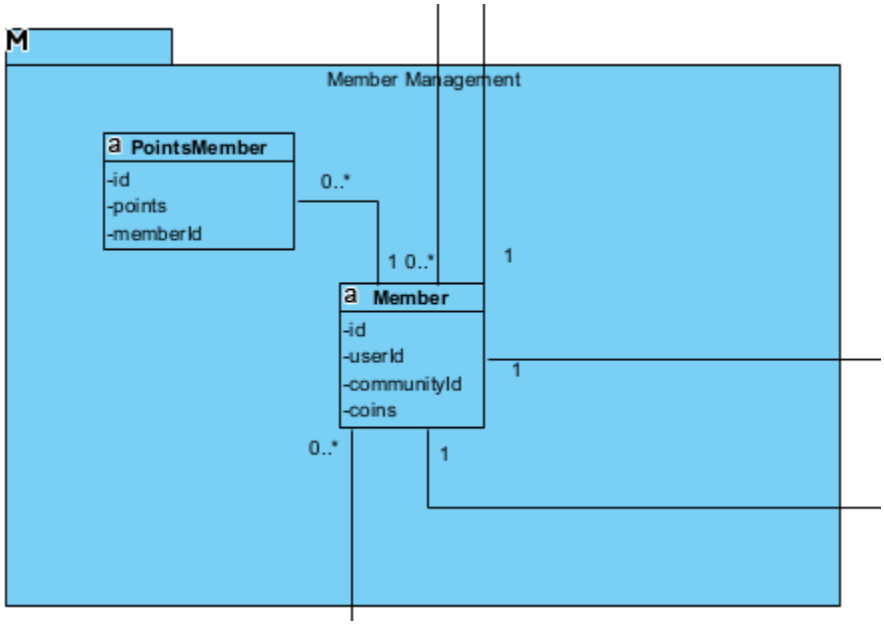



*Figura 13 - Diagrama Classes Member Management*

O pacote **Member Management** inclui um conjunto de controladores que implementam os métodos necessários para criar, atualizar e eliminar registos de membros. As operações realizadas neste pacote asseguram que cada utilizador possa ser corretamente associado a diferentes comunidades, com gestão individual do seu saldo de moedas, pontuação e restantes dados relevantes.

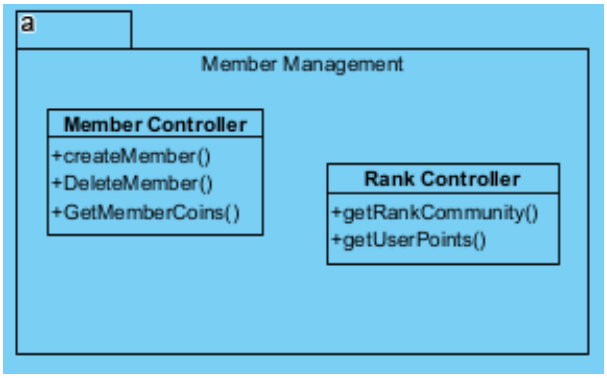



*Figura 14 - Diagrama Classes - Member Management Controllers*

## 10.3. Gestão da tarefa

O pacote **Task Management** gera as entidades relacionadas com a tarefa.

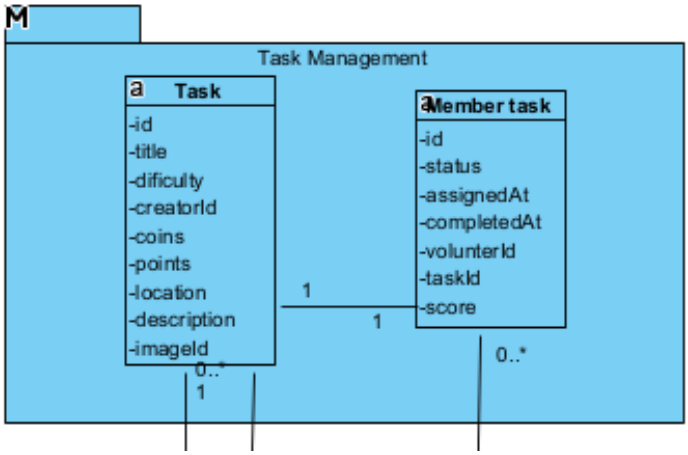



*Figura 15 - Diagrama Classes Task Management*

O pacote **Task Management** inclui um controlador que permite criar novas tarefas e alterar o seu estado ao longo do ciclo de vida (como atribuição, conclusão ou cancelamento). Este pacote garante que as tarefas sejam corretamente associadas aos membros, registando informações como dificuldade, recompensas, localização e estado de execução.

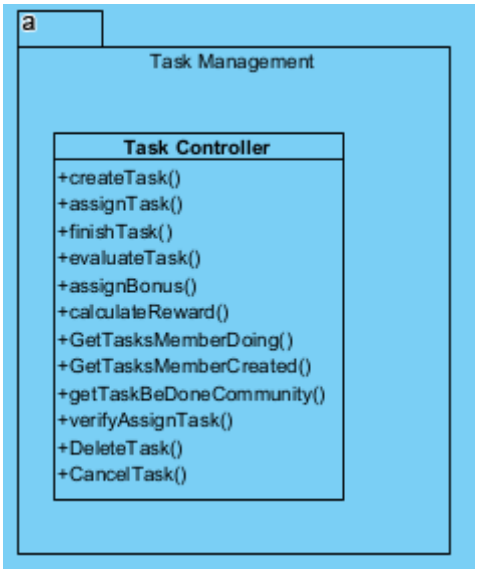



*Figura 16 - Diagrama Classes Task Management Controllers*

## 10.4. Gestão da Loja

O pacote **Store Management** gera as entidades relacionadas com a loja.

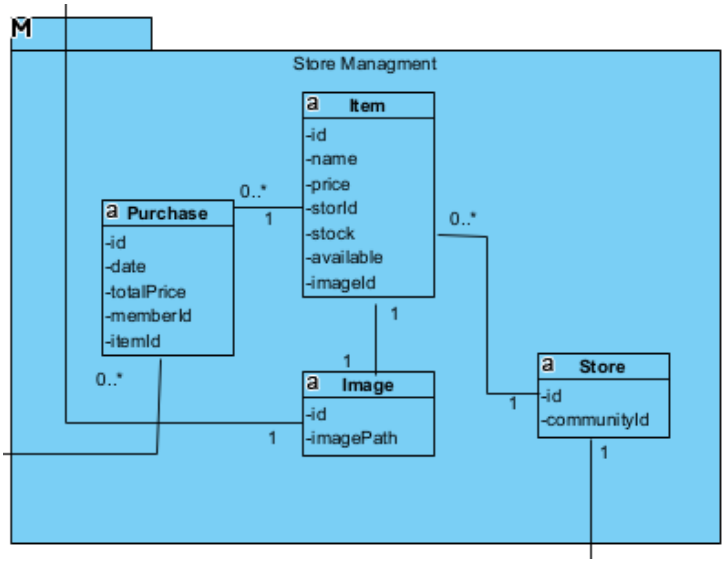



*Figura 17- DIagrama Classes Store Management*

O pacote **Store Management** inclui um de controlador que permite criar, atualizar e remover itens, bem como gerir o stock, os preços e a disponibilidade dos produtos. Além disso, este pacote trata do registo das compras efetuadas pelos membros, garantindo o correto desconto de moedas e a associação das compras aos respetivos utilizadores.

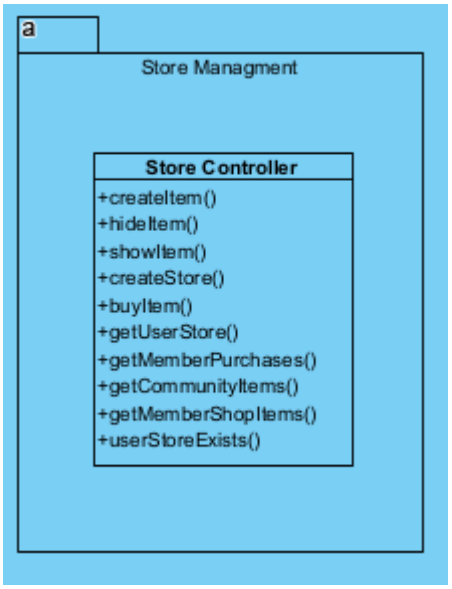



*Figura 18 - DIagrama Classes Store Management Controllers*

## 10.1. Gestão da Comunidade

O pacote **Community Management** gera as entidades relacionadas com a comunidade.

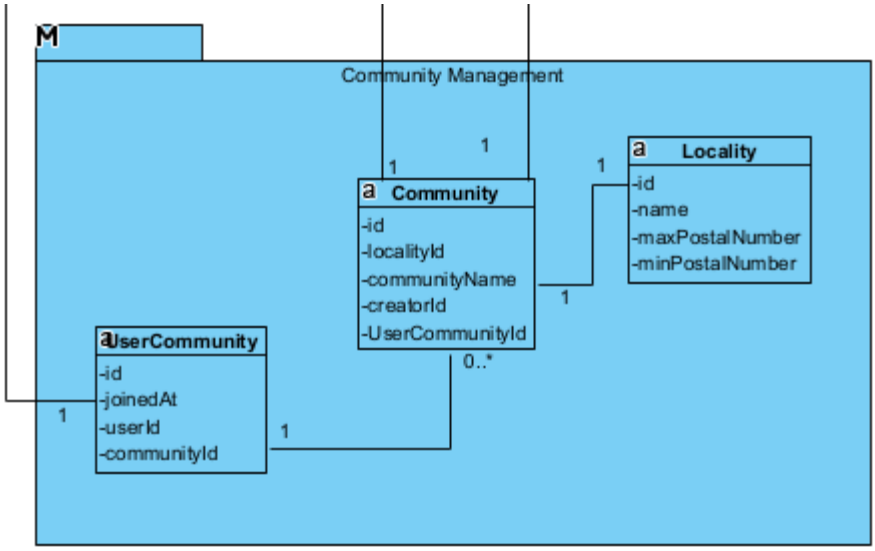
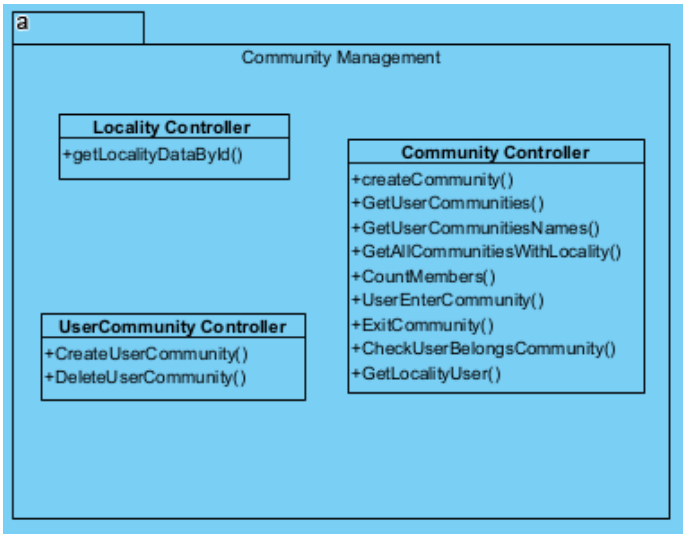



*Figura 19 - DIagrama Classes Community Management*

O pacote **Community Management** inclui um conjunto de controladores que permite criar novas comunidades, associar comunidades a uma localidade específica e gerir os utilizadores que pertencem a uma comunidade. Este pacote assegura que cada comunidade tenha limites geográficos bem definidos e que apenas os utilizadores elegíveis possam entrar ou interagir com a comunidade.



## 11. Conclusão

## 12. Bibliografia


- Material fornecido pela professora no moodle.