UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

INF01120 - TÉCNICAS DE CONSTRUÇÃO DE PROGRAMAS

TRABALHO PRÁTICO 2 - DESIGNER

DIAGRAMAS DE CLASSE E DE SEQUÊNCIA DE SISTEMA DE AVALIAÇÃO

Bruno Silva Trindade - *00240505*

Fábio Azevedo Gomes - *00287696*

Paulo Cesar Valcarenghi Junior - *00243695*

Rafael Baldasso Audibert - *00287695*

MAIO, 2019

1. INTRODUÇÃO:

A implementação está dividida em 4 pacotes, definidos como:

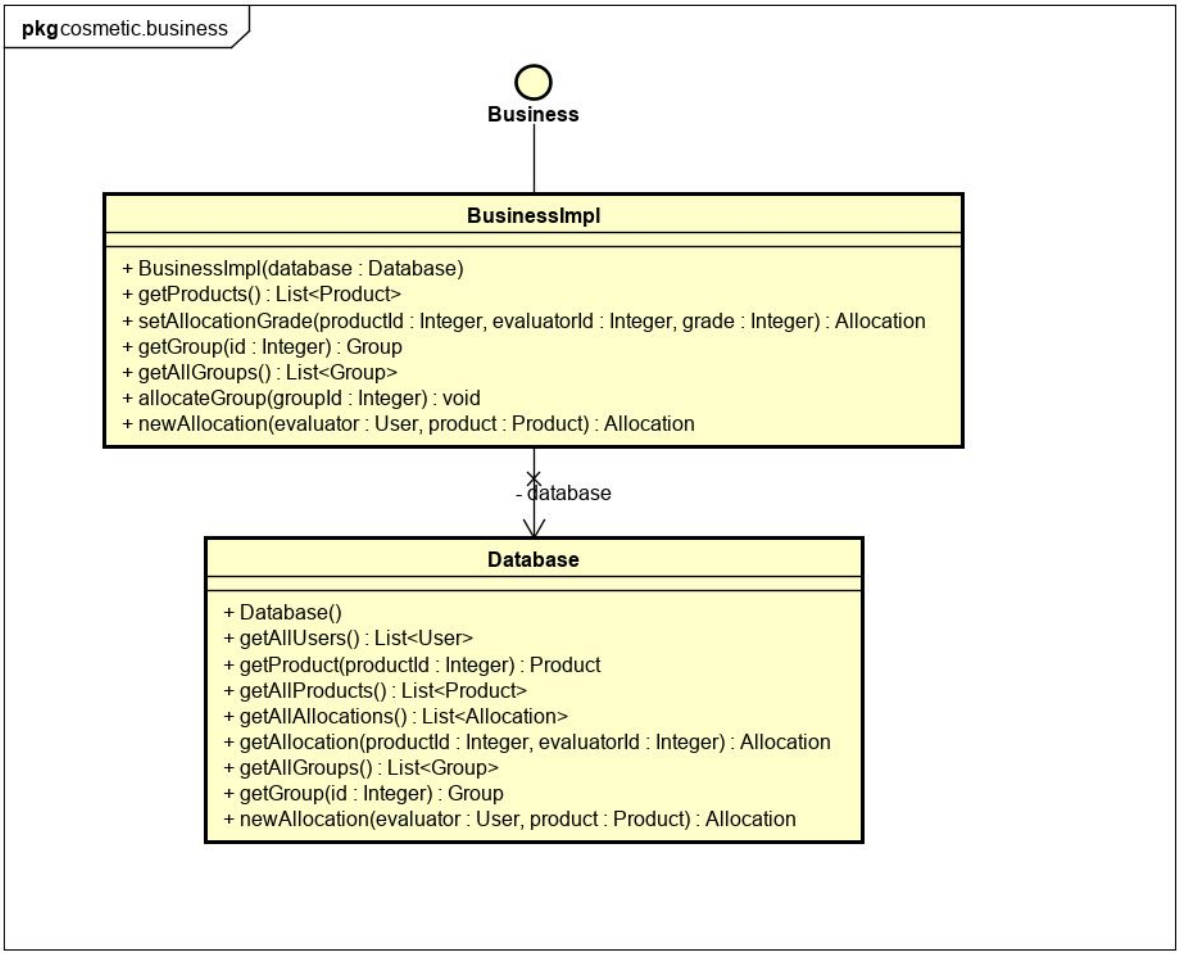
- `cosmetic.business` (responsável pelas regras de negócio)
- `cosmetic.command` (responsável por executar o que é pedido pela UI)
- `cosmetic.database` (responsável pelo armazenamento dos dados)
- `cosmetic.ui` (responsável pela UI)

A descrição do trabalho está então separada por cada um dos pacotes, com um diagrama de classes para cada um deles.

Os diagramas de sequência são exibidos ao final do relatório, com uma descrição demonstrando como ocorrem as 3 funcionalidades exemplificadas na definição do trabalho, percorrendo desde a chamada das funções de *Command* até chegarmos nos dados no `Database`.

2. DIAGRAMA DE CLASSES

2.1 cosmetic.business



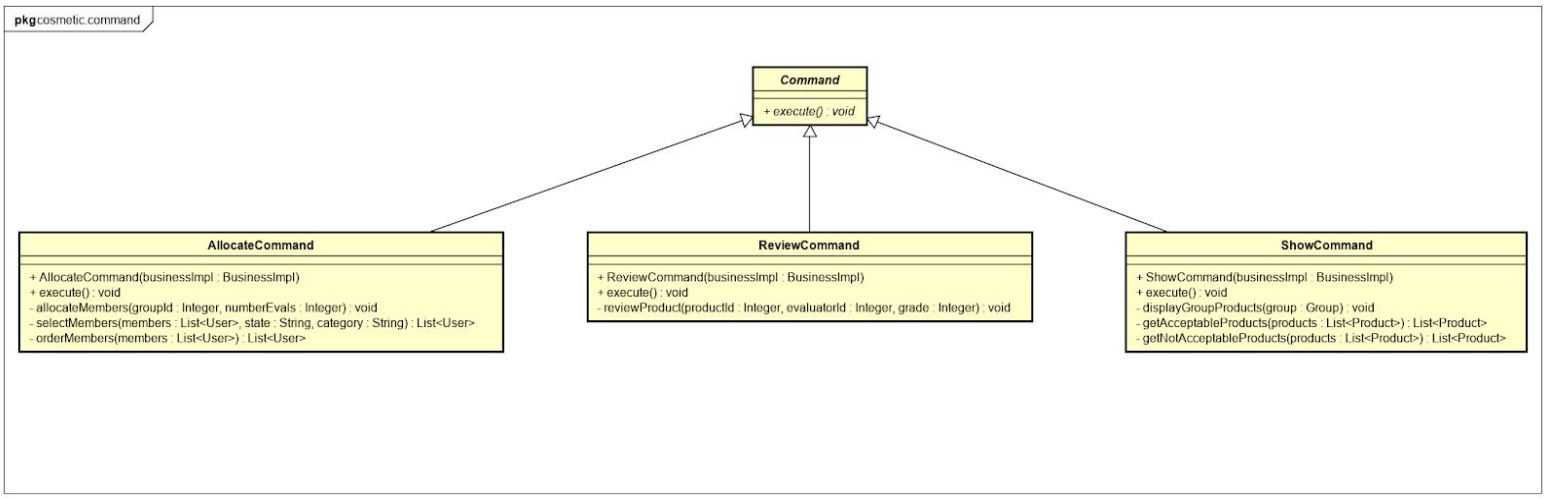
Temos uma interface `Business` (caso seja necessário implementar diferentes versões do business, porém por enquanto apenas achamos necessário ter uma só) com sua implementação `BusinessImpl`, que é a responsável por se conectar ao `Database`. O `Database` é representado nesse diagrama apenas por questões organizacionais, mas ele se encontra descrito em `cosmetic.database`.

Uma explicação rápida para os métodos da classe `BusinessImpl` se encontra abaixo:

- `+ BusinessImpl(database):` Construtor padrão, configurando o database

- *+ getProducts()*: Apenas chama a função *getAllProducts()* do *Database*;
- *+ setAllocationGrade(productId, evaluatorId, grade)*: Chama a função *getAlocation(productId, evaluatorId)* do *Database* para conseguir a *allocation*, e após chama *allocation.setGrade(grade)*.
- *+ getGroup(id)*: Apenas chama a função *getGroup(id)* do *Database*;
- *+ getAllGroups()*: Apenas chama a função *getAllGroups()* do *Database*;
- *+ allocateGroup(groupId)*: Chama a função *getGroup(groupId)* do *Database* para conseguir o *group* necessário, e após chamamos *group.allocate()* para podermos marcar esse grupo como já alocado totalmente.
- *+ newAllocation(evaluator, product)*: Apenas chama a função *newAllocation(evaluator, product)* do *Database*;

2.2 cosmetic.command



Temos uma classe abstrata `Command` que é a classe generalizada para os três comandos que nós temos, que são `AllocateCommand`, `ReviewCommand` e `ShowCommand`. Essa classe abstrata apenas possui o método, também abstrato, execute() que é o chamado pela UI do programa. As classes especializadas são instanciadas passando como parâmetro uma instância (a mesma para todos os commands) de `BusinessImpl`, para poder ter acesso aos serviços da aplicação (e um wrapper para o `Database`).

Uma explicação mais detalhada para cada um desses comandos pode ser encontrada na seção 3, juntamente com os diagramas de sequência.

Uma explicação rápida para os métodos da classe `AllocateCommand` se encontra abaixo:

- `+ AllocateCommand(businessImpl):` Construtor padrão, inicializa recebendo `businessImpl` para comunicação com a `Database`
- `+ execute()`: Realiza as operações de obtenção dos dados do usuário, e dá início ao processo de alocação
- `- allocateMembers(groupId, numberEvals)`: Utiliza as informações obtidas para realizar uma iteração sobre os produtos e membros do grupo informado.

- `- selectMembers(members)`: Utilizando os critérios estabelecidos sobre a elegibilidade de membros do comitê remove da lista de membros aqueles que não estão aptos
- `- orderMembers(members)`: Ordena uma lista de membros de um comitê de acordo com a quantidade de alocações que cada membro possui. Em caso de empate o `Id` do membro é utilizado.

Uma explicação rápida para os métodos da classe `ReviewCommand` se encontra abaixo:

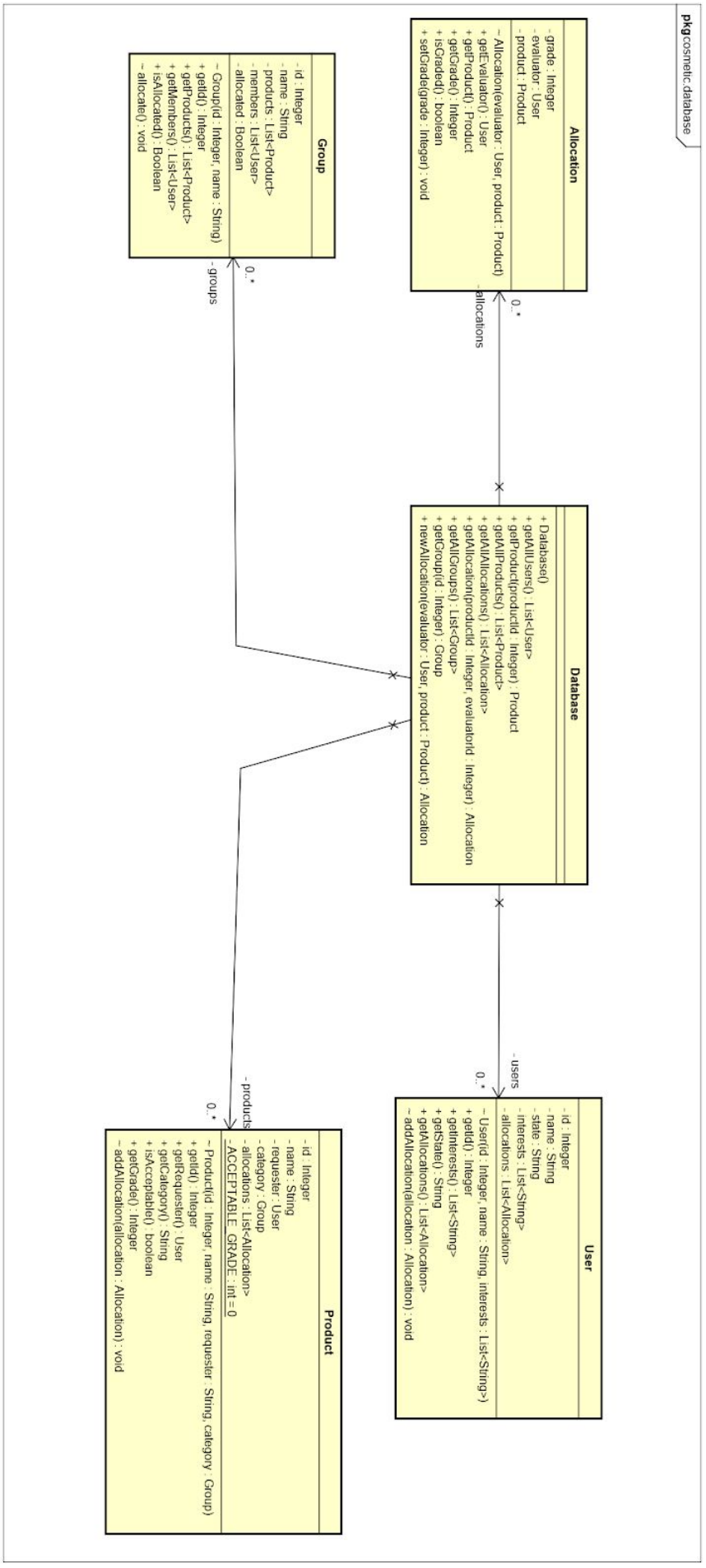
- `+ ReviewCommand(businessImpl):` Construtor padrão, inicializa recebendo `businessImpl` para comunicação com a `Database`
- `+ execute()`: Resgata os produtos e membros para que o usuário possa dar as notas.
- `- reviewProduct(productId, evaluatorId, grade)`: Permite ao usuário dar, como membro do comitê, uma nota a um produto

Uma explicação rápida para os métodos da classe `ShowCommand` se encontra abaixo:

- `+ ShowCommand(businessImpl):` Construtor padrão, inicializa recebendo `businessImpl` para comunicação com a `Database`
- `+ execute()`: Obtém do usuário o `Id` do `group` desejado e confere se o mesmo está alocado e possui todas as notas dadas pelos membros do comitê
- `- displayGroupProducts(group)`: Mostra na tela todos os produtos que estão relacionados ao `group` especificado
- `- getAcceptableProducts(products)`: Seleciona todos os produtos que tiverem uma nota aceitável (maior ou igual a 0), sendo um `product` aceitável se `product.isAcceptable()` retornar `true`, e os retorna

- `- getNotAcceptableProducts(products)`: Seleciona todos os produtos que não tiverem uma nota aceitável (menor que 0), e os retorna

2.3 cosmetic.database



Temos uma classe base *Database* que é responsável por armazenar todas as instâncias geradas das classes *Allocation*, *Group*, *User* e *Product*. A única dessas classes a qual são adicionados mais elementos é *Allocation*, portanto ela é a única que possuirá um método de instanciação público da mesma, através da *Database*, portanto todos os construtores desse pacote possuem método de visibilidade packaged.

Uma explicação rápida para os métodos da classe *Database* se encontra abaixo:

- *+ Database():* Inicialização da *Database*, com as instâncias pré-definidas
- *+ getAllUsers()*: Retorna todos os *Users* cadastrados
- *+ getProduct(productId)*: Retorna o *Product* correspondente àquele *Id*
- *+ getAllProducts()*: Retorna todos os *Products* cadastrados
- *+ getAllAllocations()*: Retorna todas as *Allocations* cadastradas
- *+ getAllocation(productId, evaluatorId)*: Dado um *Id* de *Product* e de *User* retorna a *Allocation* correspondente, caso a mesma exista
- *+ getAllGroups()*: Retorna todos os *Groups* cadastrados
- *+ getGroup(id)*: Retorna o *Group* correspondente àquele *Id*
- *+ newAllocation(evaluator, product)*: Cria uma nova *Allocation*, relacionando aquele *User* com aquele *Product*

Uma explicação rápida para os métodos da classe *Allocation* se encontra abaixo:

- *~ Allocation(evaluator, product):* Método packaged para que não possamos instanciar isso de outros lugares (restringir somente para a classe *Database*).
- *+ getEvaluator()*: Retorna o *User* que faz parte desta *Allocation*

- `+ getProduct():` Retorna o `Product` que faz parte desta `Allocation`
- `+ getGrade():` Retorna a `grade` que o usuário deu para este `Product`, caso ela já tenha sido dada
- `+ isGraded():` Retorna `true`, caso o valor presente no atributo `grade` seja diferente de `null`.
- `+ setGrade(grade):` Irá alterar o valor da propriedade grade para esse valor, apenas se `isGraded()` retornar `false`, isto é, apenas atribui o valor para o atribute `grade` se ele for `null`, ou seja se nunca tiver sido atribuído valor a ele antes, caso contrário, lança exceção dizendo que esse valor já havia sido preenchido antes e não poderia mais estar sendo preenchido agora.

Uma explicação rápida para os métodos da classe `Group` se encontra abaixo:

- `~ Group(id, name):` Método packaged para que não possamos instanciar isso de outros lugares (restringir somente para a classe `Database`).
- `+ getId()`: Retorna o `Id` do `Group`
- `+ getProducts():` Retorna a lista de `Products` que será avaliada por este `Group`
- `+ getMembers():` Retorna a lista de `Users` que fazem parte deste `Group`
- `+ isAllocated():` Retorna o valor contido no atributo `allocated`.
- `+ allocate():` Altera o status do atributo `allocated` para `true` caso `isAllocated()` retorne falso. Caso contrário, lança uma exceção já que não poderíamos estar alocando um grupo que já foi alocado antes.

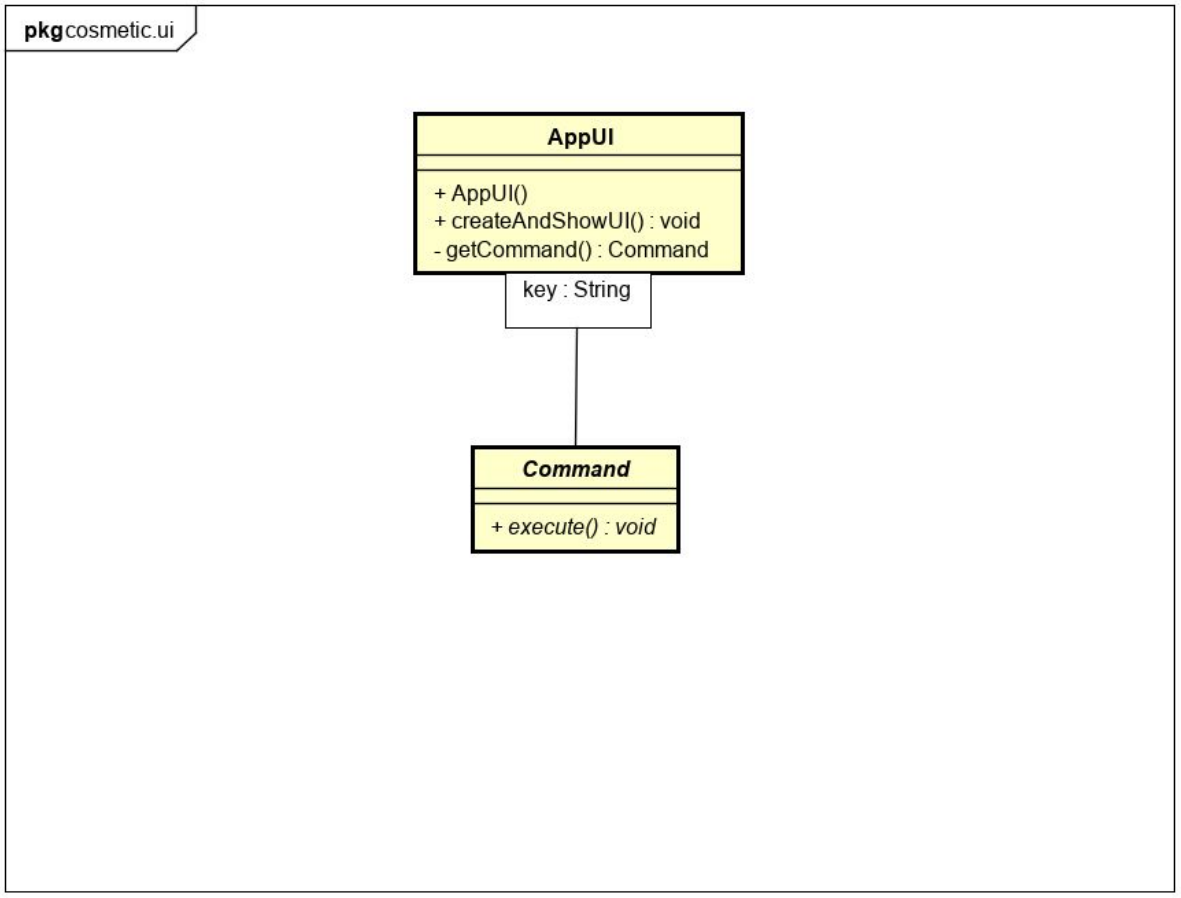
Uma explicação rápida para os métodos da classe `User` se encontra abaixo:

- *~ User(id, name, interests):* Método packaged para que não possamos instanciar isso de outros lugares (restringir somente para a classe *Database*).
- *+ getId()*: Retorna o *Id* do *User*
- *+ getInterests()*: Retorna a lista de tópicos pelos quais o *User* tem interesse
- *+ getState()*: Retorna o estado de residência do *User*
- *+ getAllocations()*: Retorna uma lista com todas as Allocations vinculadas a este User
- *~ addAllocation(allocation):* Adiciona uma nova *Allocation* vinculada a este *User* à lista de *Allocations* do mesmo

Uma explicação rápida para os métodos da classe *Product* se encontra abaixo:

- *~ Product(id, name, requester, category):* Método packaged para que não possamos instanciar isso de outros lugares (restringir somente para a classe *Database*).
- *+ getId()*: Retorna o *Id* deste *Product*
- *+ getRequester()*: Retorna o *User* que solicitou que este *Product* seja avaliado
- *+ getCategory()*: Retorna a categoria deste *Product*
- *+ isAcceptable()*: Retorna *true* se a média de notas de todas suas alocações, definidas no seu atributo *allocations*, for maior que o valor contido na sua constante *ACCEPTABLE_GRADE* (por definição do trabalho, vale 0).
- *+ getGrade():* Retorna a *grade* dada ao *Product* (Média das *grades* que foram dadas a ele pelos *Users*)
- *~ addAllocation(allocation):* Adiciona uma nova *Allocation* que está vinculada a este de *Product* à lista de *Allocations* do mesmo

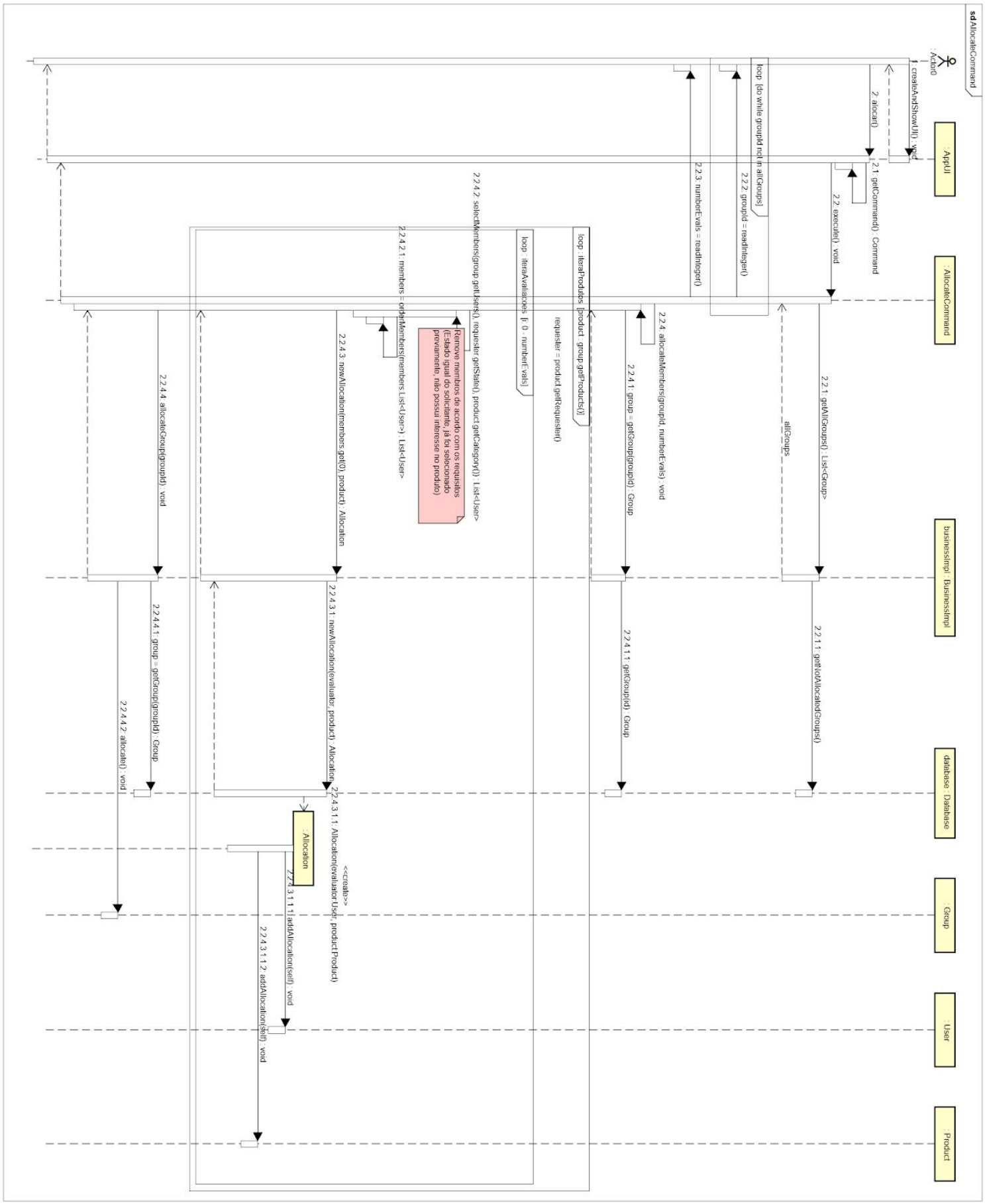
2.4 cosmetic.ui



Temos uma classe de UI básica, `AppUI` que é responsável por mostrar o menu inicial e sua UI, e após isso executar o comando que lhe foi solicitado, chamando um dos comandos contidos em um mapa que ela possui, com todos os comandos disponíveis.

3. DIAGRAMA DE SEQUÊNCIAS

3.1 AllocateCommand



O processo de alocação tem início quando o usuário atual seleciona a opção de "alocar" na UI, e então o comando é adquirido e executado.

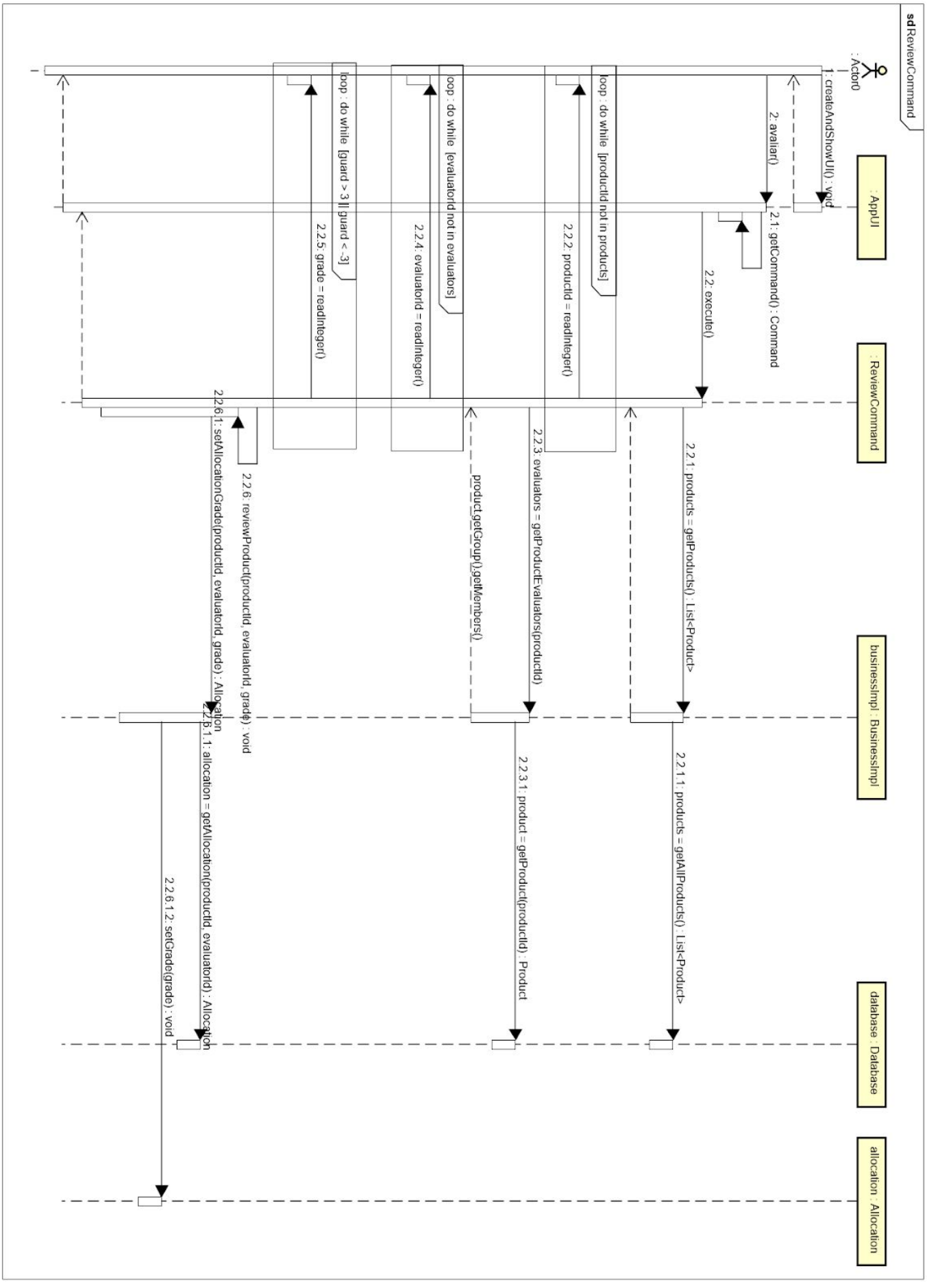
Durante a execução é feita uma consulta à base de dados para obter todos os grupos que estão cadastrados. O usuário é então inquirido sobre qual grupo deseja fazer a alocação, bem como quantos membros do comitê de avaliação devem ser alocados para cada produto.

Após obter estas informações a função `allocateMembers` é chamada e funciona de maneira um pouco diferente a demonstrada no enunciado, já que no grupo ela seria mais eficientemente executada seguindo o nosso algoritmo, sendo que os resultados são equivalentes. Nessa função, é iniciado um *loop* que itera sobre todos os produtos que estão vinculados àquele grupo. Dentro deste há outro loop, que desta vez itera sobre a quantidade de membros do comitê que deverão ser alocados por produto, conforme o usuário informou.

A função `selectMembers` é chamada, selecionando da lista de membros do grupo aqueles que são "compatíveis" com o produto em questão. Esta função também chama `orderMembers`, que por sua vez ordena a lista resultante com base na quantidade de alocações vinculadas a cada usuário. O primeiro usuário da lista é escolhido, e uma nova alocação é criada, sendo colocada na lista do usuário, do produto e na lista da base de dados.

Este *loop* é repetido até que não haja mais produtos ou não haja mais usuários disponíveis para realizar a alocação. Após o final deste *loop*, o grupo é marcado como "alocado" significando que seus membros já foram alocados aos produtos designados.

3.2 ReviewCommand

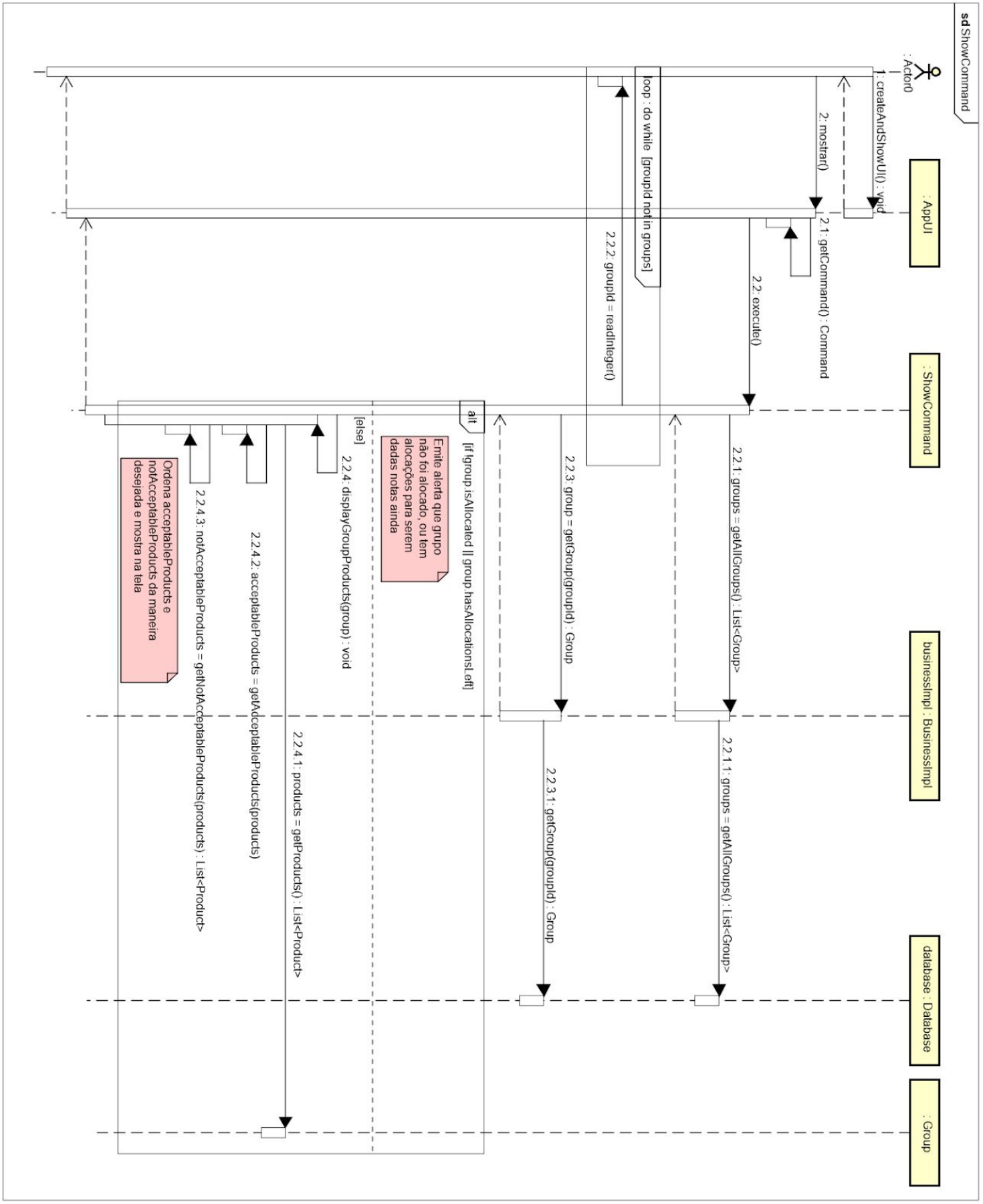


O processo de avaliação tem início quando o usuário atual seleciona a opção de "avaliar" na UI, e então o comando é adquirido e executado.

Durante a execução a lista de produtos é resgatada da base de dados, e então é apresentada ao usuário, esperando que o mesmo forneça o Id de um dos produtos contidos nela. Após receber um Id de produto válido, a lista de membros que estão alocados a este produto é apresentada, e novamente se espera que o usuário informe o Id do membro com o qual deseja avaliar. Quando informado Id's de produto e usuário válidos o programa espera que o usuário informe a nota que deseja dar, que deve estar entre -3 e 3.

Dada a nota, a alocação correspondente a este usuário e este produto é atualizada, através de um acesso à base de dados usando `BusinessImpl`. A alocação presente nas listas de alocações do usuário e do produto também são atualizadas.

3.3 ShowCommand



O processo de *display* dos produtos tem início quando o usuário atual seleciona a opção de "mostrar" na UI, e então o comando é adquirido e executado.

Através de *BusinessImpl* é feito um acesso à base de dados, onde a lista de grupos é resgatada. A lista é apresentada ao usuário e uma resposta com o Id do grupo desejado é esperada. Caso este grupo não tenha sido alocado ou existam alocações de usuário com produto sem uma nota é emitido um alerta ao usuário.

Caso contrário, a lista de produtos alocados à este grupo, é iterada sobre, com a função *getAcceptableProducts*, retornando todos os produtos que possuem uma nota média maior ou igual a 0, que está definido. A função *getNotAcceptableProducts*, como o nome sugere, obtém aqueles produtos que restaram da lista, e ambas as listas podem ser impressas na tela com a ordenação desejada, através de funções de ordenamento de lista built-ins do Java.