Mirante: Sistema de ensino-aprendizagem orientada a dados

Relatório Final de Desenvolvimento

Engenharia de Software Aplicada

Juno Takano

Jacareí, 11 de dezembro de 2023

Sumário

1. Introdução

Para este projeto, foi desenvolvido o protótipo de um sistema de armazenamento e análise de dados gerados no processo de ensino-aprendizagem. Para esta implementação, foram objetivos principais:

1. Armazenar exercícios e alternativas associadas a estes exercícios
2. Dentre as alternativas, armazenar a alternativa e/ou ordenação correta
3. Permitir inserção, edição e deleção através de *endpoints* e requisições HTTP
4. Disponibilizar uma interface gráfica onde estes exercícios possam ser respondidos
5. Registrar a resposta na forma de um evento

O projeto do sistema buscou manter uma estrutura de classes modular, com separação de funções, utilizando padrões de agregação para a modelagem de coleções de exercícios e alternativas. Isto foi possível pois as estruturas de dados são criadas através de técnicas de mapeamento objeto-relacional (ORM), e não manualmente definidas com uma sublinguagem de definição de dados. Esse gerenciamento é feito pela implementação Hibernate da especificação Jakarta Persistence API.

Para isto, o padrão de repositório foi utilizado. Estendendo esta interface, é possível instanciar objetos de repositórios que persistem os dados em um banco integrado ao sistema. Isto facilita a fase de desenvolvimento, mas pode ser desacoplado sem alterações no código devido ao mapeamento objeto-relacional.

Em seu núcleo, o sistema consiste em uma aplicação Java que utiliza o *framework* Spring para executar um servidor com múltiplos *endpoints* capazes de receber e retornar requisições HTTP através de estruturas de dados JSON, parâmetros e caminhos de URL.

A interface voltada para o usuário não está contida no sistema, que pode portanto ser executado em ambientes de servidor, conteinerizados, ou embarcados, contanto que sua arquitetura seja capaz de executar uma máquina virtual Java.

Além da portabilidade do módulo servidor, o desacoplamento da interface também permite que qualquer implementação de um *front-end* que seja ca-

paz de realizar e receber requisições HTTP poderá comunicar-se com o sistema.

Para fins de demonstração, o código fonte associado a este projeto fornece uma interface desenvolvida através de formulários HTML e JavaScript, que pode ser executada em um navegador web.

2. Mapa de Interfaces

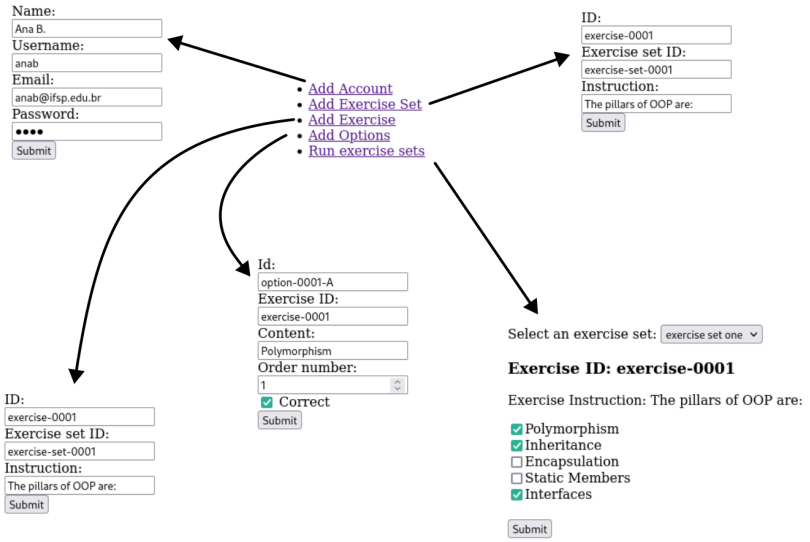

Figura 1: Mapa de interfaces. Para evitar a repetição, o menu aparece apenas uma vez. Este elemento de navegação está presente em todas as telas.

O mapa de interfaces acima mostra as diferentes telas utilizadas para comunicação com os *endpoints* `account`, `exercise-set`, `exercise` e `option`. A última tela, abaixo à direita, primeiro consome os dados do *endpoint* `options` para formatar um formulário com os exercícios, e então envia um evento para o *endpoint* `event` contendo as alternatives selecionadas e a data atual.

3. Requisitos

Os requisitos do sistema foram elencados de acordo com sua prioridade e as dependências entre eles. A lista abaixo utiliza números menores para os itens de **maior** prioridade.

1. Criar uma conta
   1.2. Editar os detalhes da conta
   1.3. Apagar a conta
2. Armazenar um conjunto de exercícios
   2.1. Armazenar um exercício
   2.2. Atualizar um exercício
   2.3. Apagar um exercício
3. Fazer todo o conjunto de exercícios
   3.1. Obter os dados do conjunto de exercícios
   3.2. Formatar os dados em uma interface gráfica
   3.3. Retornar o índice de acertos para o backend
4. Armazenar um registro datado de respostas
   4.1. Armazenar cada conjunto datado de alternativas escolhidas
   4.2. **Não** deve ser possível alterar um evento
   4.2. **Não** deve ser possível apagar um evento

O levantamento dos requisitos levou em conta o escopo do protótipo e a importância de manter um registro de *eventos* que possa ser contrastado ao estado atual da aplicação em um dado momento, seja para identificar inconsistências ou para realizar análises por inferências e correlações, ou ainda para o cálculo de métricas como assiduidade e retenção, que são dados temporais de natureza diferente da acuidade.

4. Casos de uso

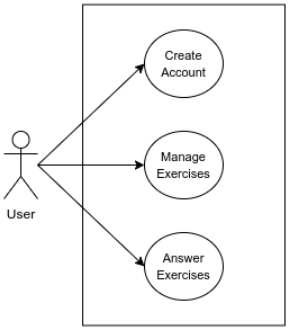

Figura 2: Diagrama de casos de uso.

Para o presente protótipo, o sistema possui apenas um agente envolvido: um mesmo usuário cria e responde às questões enviadas. As questões não são atualmente atreladas a uma conta, o que significa que os conjuntos criados por diferentes contas são compartilhados entre elas.

Passivamente, este usuário está ainda gerando um registro cronometrado de suas respostas, o que permite obter ainda outras informações que são objetivo deste trabalho como parte de um projeto mais amplo.

```
SELECT * FROM EVENT;
```

ID	TIMESTAMP	CONTENT	DESCRIPTION
1	2023-12-12 01:46:03.74	option-0001-A,option-0001-B,option-0001-E	exercise set response

(1 row, 4 ms)

Figura 3: Visão do banco de dados, mostrando o registro de um evento. O evento não estabelece relação entre as entidades que menciona.

Este caso de uso está mais relacionado à possibilidade de armazenar dados sobre seu aprendizado, e obter mais tarde dados sobre a frequência de estudo e a retenção do conhecimento em cada tópico. Isto contudo não é algo que, no presente protótipo, pode ser obtido de maneira já metrificada.

5. Estrutura de Classes

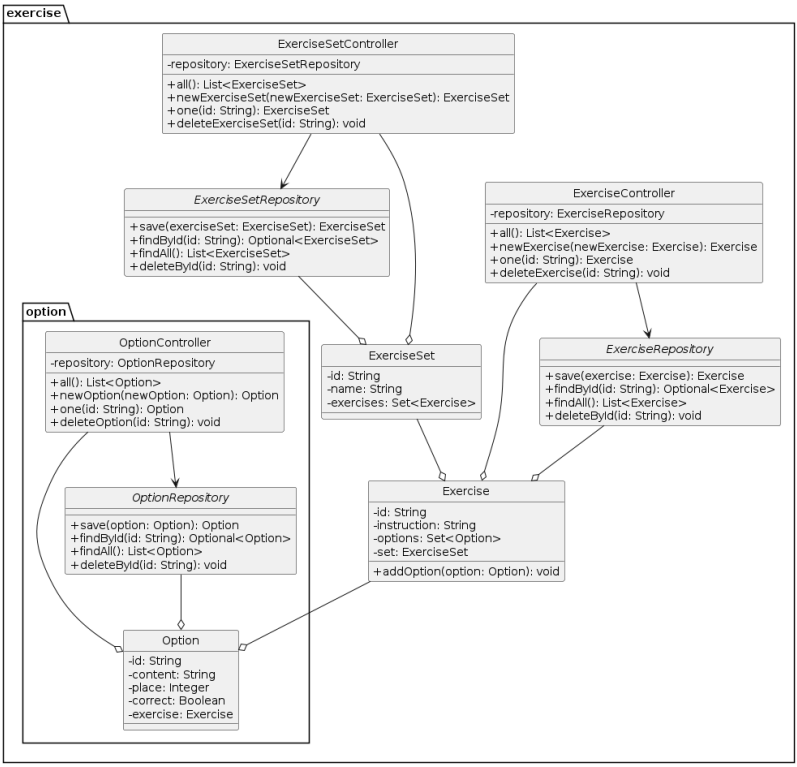

Figura 4: Diagrama de classes do subpacote `exercise`.

O diagrama de classes acima mostra o subpacote `exercise` com seu subpacote `option`. As relações de agregação enfatizam a responsabilidade das classes controladoras e dos repositórios em colecionar, armazenar e distribuir as estruturas de dados.

Outro padrão que fica bastante visível no diagrama é a hierarquia de dependências que vai de um nível mais alto, na classe `ExerciseSetController`, até um nível mais baixo, onde chega na unidade atômica da aplicação, `Option`.

É interessante ressaltar que, quando os dados são obtidos a partir dos *endpoints*, cada alternativa ainda possui todas as estruturas de dados associadas a ela, permitindo que, a partir dessa mesma requisição, não seja necessário inquirir a todos os outros *endpoints* as informações necessárias para com-

preender a qual exercício, e então a qual conjunto de exercícios, cada alternativa pertence.

O código abaixo, que mostra a resposta crua em `JSON` obtida do *endpoint* `option`, exemplifica a estrutura de uma alternativa (*option*), note que no campo `exercise_id`, está aninhada não apenas a estrutura de dados referente ao exercício que agrega esta alternativa como também, aninhada a este, também a estrutura do conjunto de exercícios que os agrega.


```
{
  "id": "0058",
  "content": "Polymorphism",
  "place": 2,
  "correct": true,
  "exercise_id": {
    "id": "ex0005T",
    "instruction": "The three pillars of OOP are:",
    "set_id": {
      "id": "exset0002T",
      "name": "Programming Paradigms Exercise Set"
    }
  }
}
```

5.1. Subpacote `event`

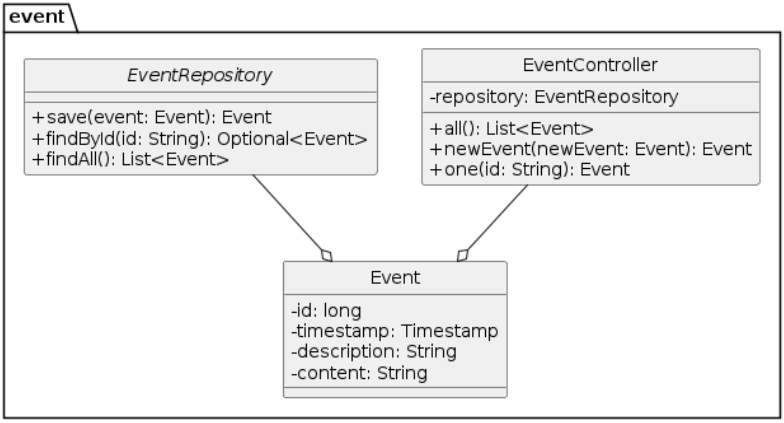

Figura 5: Diagrama de classes do subpacote `exercise`.

O subpacote de eventos não estabelece relações diretas com as outras entidades do sistema. No detalhe acima, ele aparece mostrando apenas as agregações feitas por seu controlador e repositório. Quando inserido, o evento contém no campo `content` uma descrição textual do que especificamente define o evento.

No caso do evento que aparece neste protótipo, tratam-se dos identificadores das alternativas escolhidas. Não se trata de uma chave estrangeira, mas do identificador na forma de texto puro, separado por vírgulas. Embora isto signifique que alterações não-documentadas podem causar perda de informação, também significa que o registro preciso do que aconteceu naquele ponto no tempo não será perdido. Aliado a outros dados complementares, este registro de eventos pode portanto ser uma ferramenta essencial no processo de análise.

5.2. Subpacote `account`

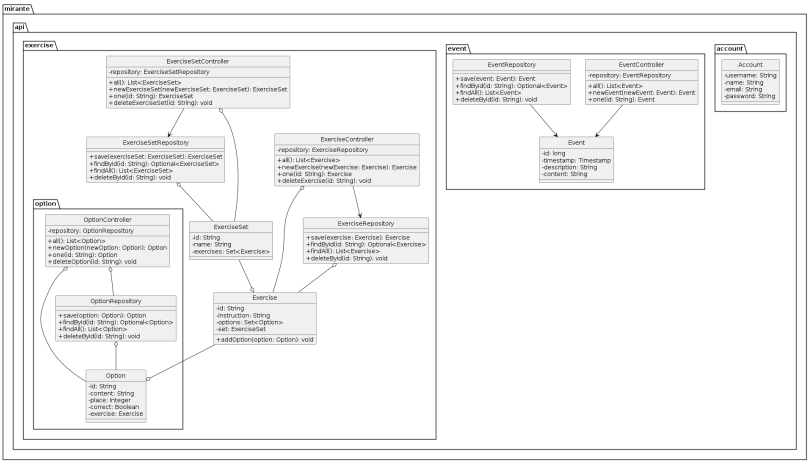

Figura 6: Diagrama de classes do subpacote `account`.

O subpacote `account` contém a classe, seu controlador e repositório., `Account` define os dados `name`, `username`, `email`, 'password'.

O sistema prevê como requisito a possibilidade de criar contas, mas em sua versão atual não oferece um mecanismo de autenticação que permita, por exemplo, limitar o acesso a determinados conjuntos de exercício de acordo com a autorização para acessá-los.

6. Diagramas de sequência dos casos de uso

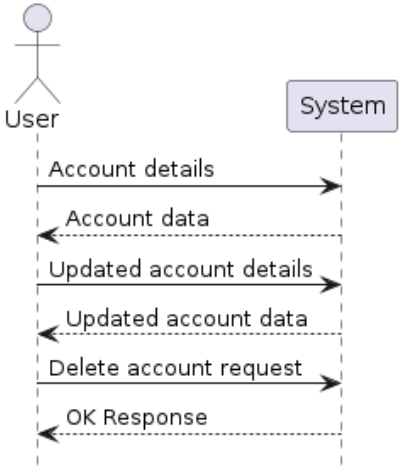

Figura 7: Diagrama de sequência para o caso de uso *"Create Account"*.

O diagrama mostra as interações possíveis antes e depois de uma conta ser criada. Uma vez enviados os detalhes de uma conta, o sistema pode responder com os dados da própria conta, que confirmam a criação. O usuário ou cliente então pode enviar a mesma requisição para atualizar os dados da conta, ou uma requisição HTTP `DELETE` para apagar a conta.

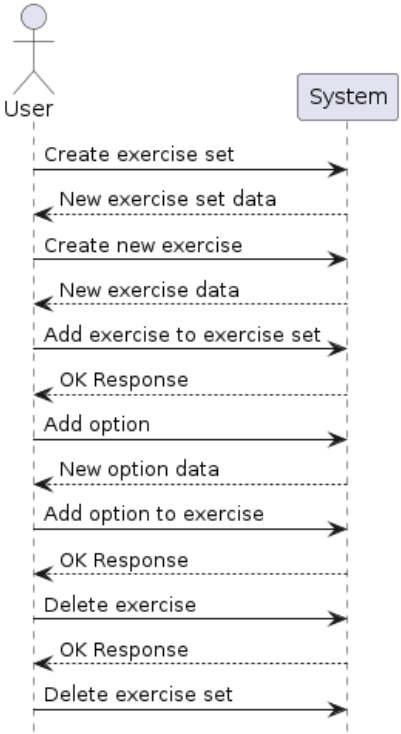

Figura 8: Diagrama de sequência para o caso de uso *"Manage Exercises"*.

Mostra as ações possíveis no gerenciamento de exercícios. Após a criação de um conjunto de exercícios, é possível atrelar múltiplos exercícios a ele e a estes, alternativas. É possível então obter todas as alternativas em formato JSON para que o usuário ou cliente possa respondê-las. As requisições também podem ser utilizadas para atualizar ou apagar qualquer uma destas entidades.

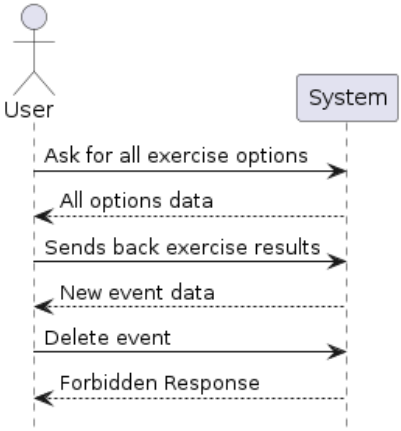

Figura 9: Diagrama de sequência para o caso de uso *"Answer Exercises"*.

A resposta de exercícios é o único caso de uso onde o usuário retorna dados para o sistema na forma de um evento. Uma demonstração deste caso de uso está disponível no código fonte associado, com uma interface gráfica e ainda um script para carregar os dados de teste, que pode ser executado usando a ferramenta `Hurl`.[1]

[1] Hurl - Run and Test HTTP Requests

7. Diagramas de sequência do sistema

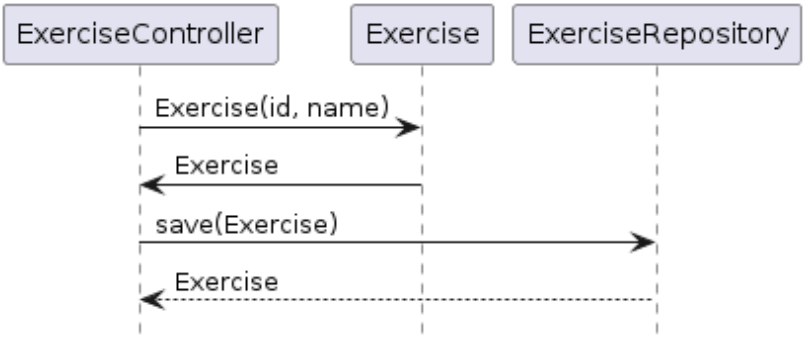

Figura 10: Detalhe do diagrama de sequência mostrando a inserção de uma entidade do tipo `Exercise` no sistema.

Como a princípio todas estas entidades estão desacopladas, elas não precisam ser instanciadas de forma aninhada. É preciso, porém, que a entidade que contém as unidades menores já exista no banco de dados. Por exemplo, para inserir um exercício associado a um conjunto de exercícios, é preciso que o identificador do conjunto já exista para que seja feita a associação.

De outra forma, é possível criar um exercício que ainda não possui nenhuma associação com qualquer conjunto específico, mas não preemptivamente estabelecer uma relação com uma determinada chave para só então criar o conjunto correspondente a ela.

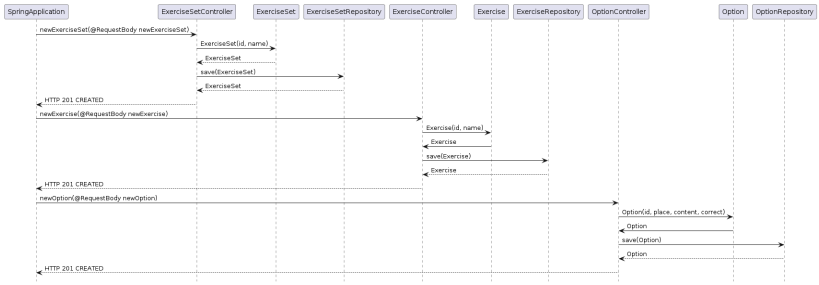

Figura 11: Diagrama de sequência do sistema, mostrando todo o ciclo até a criação de uma alternativa. Nota-se o desacoplamento entre as classes.

O diagrama de sequência acima mostra de maneira mais abrangente o ciclo até que seja possível inserir uma alternativa no sistemma que esteja atrelada a um exercício que, por sua vez, está atrelado a um conjunto de exercícios.

Cabe ressaltar o baixo acoplamento entre as diferentes entidades. As relações entre elas se dão no nível do banco apenas, e elas podem ser obtidas a partir dos seus repositórios. Não é necessário aninhar as entidades em seus construtores na hora de instanciá-las.

Apesar de haver dependência entre cada uma delas, suas instâncias vivem separadas e podem ser sempre repopuladas a partir do estado persistido no banco. O ciclo de vida das entidades do banco independe do que está atualmente na aplicação.

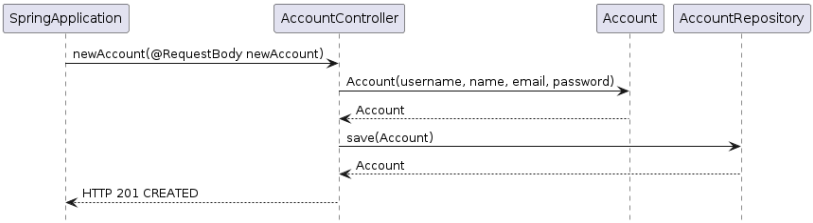

Figura 12: Diagrama de sequência da criação de uma conta. Esta implementação segue o mesmo padrão de projeto adotado nas demais classes.

O controlador recebe uma requisição HTTP de tipo `POST` no endpoint `/exercise-set`. Se a requisição está de acordo com a estrutura esperada, isto é, se há um construtor que permita instanciar um objeto da classe 'ExerciseSet' com a estrutura recebida, este objeto é instanciado como parâmetro do próprio método `newExerciseSet` que está anotado como o método que mapeia o *endpoint* `/exercise-set` para requisições do tipo `POST`.

O controlador chama, então, o método `add` da sua instância de `ExerciseSetRepository`, que retorna[2] o próprio objeto salvo.

O método `newExerciseSet` do controlador então retorna esse mesmo objeto, que por fim confirma a criação do objeto. Como resposta à requisição HTTP recebida, o controlador envia o código de status `201 CREATED`.

[2]Ver Interface JpaRepository (Spring Data JPA Parent 3.2.0 API)

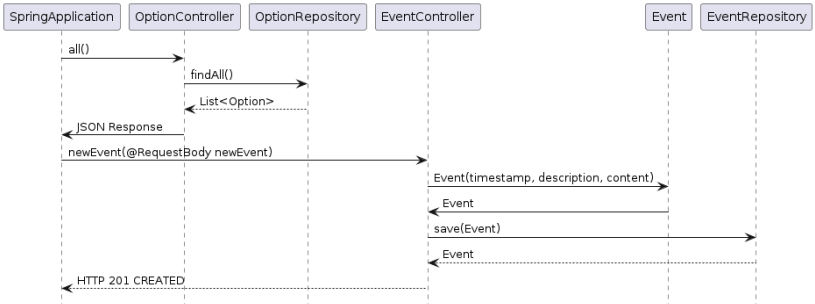

Figura 13: Diagrama de sequência mostrando a obtenção das alternativas e, em seguida, o envio de um evento com o resultado. Este diagrama se refere especificamente ao caso de uso *"Answer Exercises"*.

Se neste momento for feita uma nova requisição, desta vez de tipo `GET`, para o mesmo *endpoint*, `/exercise-set`, o controlador irá chamar o método `findAll` do repositório, que retorna o conjunto[3] de todas as entidades do tipo `ExerciseSet`. Este conjunto é representado na resposta à requisição na forma de um *array* contendo uma representação em `JSON` dos elementos constantes na tabela de entidades `ExerciseSet`. Este comportamento se repete para as demais entidades.

Se a requisição `GET` for feita para o *endpoint* `/exercise-set/{id}`, onde `{id}` é o identificador único utilizado para chavear esta entidade e suas relações no banco de dados, o controlador chama o método `findById` do repositório, que retorna o objeto encontrado ou `Optional#empty()` caso o identificador não corresponda a nenhum objeto.

Caso tenha sido encontrado um objeto, uma representação dele é retornada na forma de uma estrutura `JSON`. Em caso negativo, é retornado um *array* vazio, representado literalmente no corpo da resposta como `[]`. O comportamento desta requisição também se repete nas demais entidades.

[3] Mais especificamente, uma coleção que implemente a interface `Iterable`.

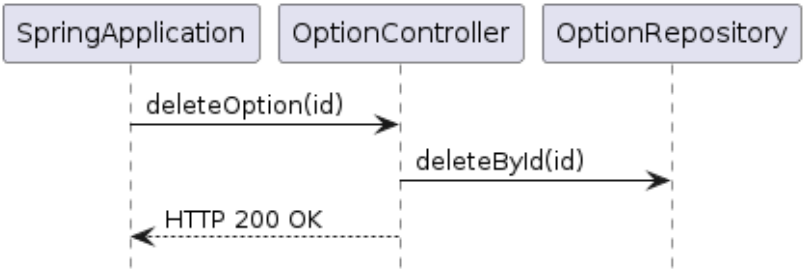

Figura 14: Diagrama de sequência do sistema, mostrando um evento de deleção. Não há retorno no método `deleteById`.

Resta ainda a possibilidade de uma requisição de tipo `DELETE` para o mesmo *endpoint* `/exercise-set/{id}`. O controlador irá chamar o método `deleteById`, passando a ele o identificador recebido no caminho da URL. O método não fornece nenhum retorno. Caso o identificador não seja encontrado, a requisição é apenas ignorada. Embora esteja disponível em quase todas as demais entidades do sistema, não é possível solicitar a deleção de uma entidade `Event`.