



Tecnologia em Análise e Desenvolvimento de Sistemas

Desenvolvimento Web Back-end

Prática Compensatória

GraphQL - Bazar Books Mobile App

Nome	Matrícula
Ana Laura Angelieri	BP3038262
Évelin Ferreira da Silva	BP3039447
Giovanna Laura Cravo e Silva	BP3039391
Soraya Costa Soares	BP3039315

Sumário

1 Especificações	3
1.1 Entidades do sistema	3
1.2 Classes Java	4
Évelin - Address:	4
Book:	5
Soraya - Cart:	6
Soraya - CartItem:	6
Giovanna - Notification:	7
User:	7
Ana Laura - Author:	7
2 Evidências dos Testes Funcionais	8
2.1 Entidade Address	8
2.2 Entidade Author	11
2.3 Entidade Cart	14
2.4 Entidade CartItem	15
Entidade Notification	16

1 Especificações

1.1 Entidades do sistema

Entidade	Schema	Query	Mutation
Ana - Author	<pre> type Author { idAuthor: Int! name: String! imageUrl: String description: String books: [Book] } </pre>	<pre> getAllAuthors: [Author] getAuthorById(id: Int!): Author </pre>	<pre> createAuthor(authorInput: AuthorInput!): Author updateAuthor(id: Int!, authorInput: AuthorInput!): Author deleteAuthor(id: Int!): String </pre>
Évelin - Address	<pre> type Address { id: ID! street: String! number: Int! complement: String district: String! city: String! state: String! postalCode: String! latitude: Float! longitude: Float! user: User! } </pre>	<pre> getAllAddresses: [Address!]! getAddressById(id: ID!): Address getAddressesByUserId(userId: ID!): [Address!]! </pre>	<pre> createAddress(addressInput: AddressInput!): Address! updateAddress(id: ID!, addressInput: AddressInput!): Address deleteAddress(id: ID!): String! </pre>
Giovanna - Notification	<pre> type Notification { idNotification: Int! title: String! message: String! read: Boolean! sentDate: String user: User! } </pre>	<pre> getAllNotificationsByUserId(userId: Int!): [Notification!]! getUnreadNotificationsByUserId(userId: Int!): [Notification!]! </pre>	<pre> createFavoriteNotification(userId: Int!, bookId: Int!): Notification markNotificationAsRead(id: Int!): String </pre>

		getNotificationById(id: Int!): Notification	deleteNotification(idUser: Int!, id: Int!): String
Soraya - Cart	<pre> type Cart { id_cart: ID! userId: Int! items: [CartItem!]! total: Float! } </pre>	<pre> cartByUserId(userId: Int!): Cart </pre>	<pre> createCart(userId: Int!): Cart deleteCart(cartId: ID!): String </pre>
Soraya CartItem	<pre> type CartItem { id: ID! bookId: Int! unitPrice: Float! quantity: Int! subtotal: Float! } </pre>	<pre> cartItemsByCartId(cartId: Int!): [CartItem!]! </pre>	<pre> addCartItem(cartId: ID!, item: CartItemInput!): CartItem updateCartItemQuantity(cartId: ID!, itemId: ID!, quantity: Int!): CartItem removeCartItem(cartId: ID!, itemId: ID!): String </pre>

Tabela - Schema das Entidades

1.2 Classes Java

Évelin - Address:

```

// AddressController.java
package com.bazar.bazarbooks.controller;

import com.bazar.bazarbooks.dto.AddressInput;
import com.bazar.bazarbooks.model.Address;
import com.bazar.bazarbooks.service.AddressService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.graphql.data.method.annotation.Argument;
import org.springframework.graphql.data.method.annotation.QueryMapping;
import org.springframework.stereotype.Controller;
import org.springframework.util.List;

import java.util.List;

@Controller
public class AddressController {

    @Autowired
    private AddressService addressService;

    @QueryMapping
    public List<Address> getAllAddresses() {
        return addressService.getAllAddresses();
    }
}

// AddressInput.java
package com.bazar.bazarbooks.dto;

import java.util.List;

public class AddressInput {

    private String street;
    private int number;
    private String complement;
    private String district;
    private String city;
    private String state;
    private String postalCode;
    private double latitude;
    private int userId;

    public AddressInput() {}

    public AddressInput(String street, int number, String complement, String district,
        String city, String state, String postalCode, double latitude,
        double longitude, int userId) {
        this.street = street;
        this.number = number;
        this.complement = complement;
        this.district = district;
        this.city = city;
    }
}

// AddressService.java
package com.bazar.bazarbooks.service;

import com.bazar.bazarbooks.model.Address;
import com.bazar.bazarbooks.repository.AddressRepository;
import org.springframework.stereotype.Service;
import org.springframework.util.List;

import java.util.List;

@Service
public class AddressService {

    @Autowired
    private AddressRepository addressRepository;

    @Autowired
    private UserRepository userRepository;

    public List<Address> getAllAddresses() {
        return addressRepository.findAll();
    }

    public Address getAddressById(int id) {
        return addressRepository.findById(id).orElse(null);
    }

    public List<Address> getAddressByUserId(int userId) {
        return addressRepository.findByUserId(userId);
    }
}

// Address.java
package com.bazar.bazarbooks.model;

import java.util.List;

import javax.persistence.*;

@Entity
@Table(name = "address")
public class Address {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_address")
    private int idAddress;

    private String street;
    private int number;
    private String complement;
    private String district;
    private String city;
    private String state;
    private String postalCode;
    private double latitude;
    private double longitude;
}

```

Classes Java para a Entidade de Address

Book:

```
BookController.java X
bazarbooks > src > main > java > com > bazar > bazarbooks > controller > BookController.java | Language Support for Java(TM) by Red Hat > {} com.bazar.b
14
15 @Controller
16 public class BookController {
17
18     @Autowired
19     private BookService bookService;
20
21     @GetMapping
22     public List<Book> getAllBooks() {
23         return bookService.getAllBooks();
24     }
25
26     @GetMapping
27     public Book getBookById(@Argument int id) {
28         return bookService.getBookById(id);
29     }
30
31     @PostMapping
32     public Book createBook(@Argument BookInput book) {
33         return bookService.createBook(book);
34     }
35
36 }

BookService.java X
bazarbooks > src > main > java > com > bazar > bazarbooks > service > BookService.java | Language Support for Java(TM) by Red Hat > {} com.bazar.bazar
9
10 import java.util.List;
11 import java.util.Optional;
12
13 @Service
14 public class BookService {
15
16     @Autowired
17     private BookRepository bookRepository;
18
19     public List<Book> getAllBooks() {
20         return bookRepository.findAll();
21     }
22
23     public Book getBookById(int id) {
24         return bookRepository.findById(id).orElse(null);
25     }
26
27     public Book createBook(BookInput bookInput) {
28         Book book = new Book();
29         book.setTitle(bookInput.getTitle());
30         book.setImageUrl(bookInput.getImageUrl());
31         book.setDescription(bookInput.getDescription());
32     }
33
34 }

BookInput.java X
bazarbooks > src > main > java > com > bazar > bazarbooks > dto > BookInput.java | BookInput
1
2 package com.bazar.bazarbooks.dto;
3
4 public class BookInput {
5     private String title;
6     private String imageUrl;
7     private String description;
8     private String price;
9     private double rating;
10    private int reviewCount;
11    private String store;
12
13    public BookInput() {}
14
15    public BookInput(String title, String imageUrl, String description, String price,
16                      double rating, int reviewCount, String store) {
17        this.title = title;
18        this.imageUrl = imageUrl;
19        this.description = description;
20        this.price = price;
21        this.rating = rating;
22        this.reviewCount = reviewCount;
23        this.store = store;
24    }
25
26 }

Book.java X
bazarbooks > src > main > java > com > bazar > bazarbooks > model > Book.java | Book
1
2 package com.bazar.bazarbooks.model;
3
4 import jakarta.persistence.Entity;
5 import jakarta.persistence.GeneratedValue;
6 import jakarta.persistence.GenerationType;
7 import jakarta.persistence.Id;
8 import lombok.Getter;
9 import lombok.NoArgsConstructor;
10 import lombok.Setter;
11
12 @Entity
13 @Getter
14 @Setter
15 @NoArgsConstructor
16 @AllArgsConstructor
17 public class Book {
18
19     @Id
20     @GeneratedValue(strategy = GenerationType.IDENTITY)
21     private int idBook;
22
23     private String title;
24
25 }
```

Classes Java para a Entidade de Book

Soraya - Cart:

```
CartController.java
21 public class CartController {
22     public CartDTO cartByUserId(@Argument int userId) {
23         Cart cart = cartService.getCartByUserId(userId);
24         List<CartItemDTO> itemDTOS = cart.getItems().stream()
25             .map(this::toDTO)
26             .collect(Collectors.toList());
27         return new CartDTO(cart.getId(), cart.getUser().getIdUser(), itemDTOS);
28     }
29
30     @GetMapping
31     public List<CartItemDTO> cartItemsByCartId(@Argument int cartId) {
32         return cartItemService.getItemsByCartId(cartId)
33             .stream()
34             .map(this::toDTO)
35             .collect(Collectors.toList());
36     }
37
38     // ===== MUTATIONS =====
39
40     @PostMapping
41     public CartDTO createCart(@Argument int userId) {
42         Cart cart = cartService.createCartIfNotExists(userId);
43         return new CartDTO(cart.getId(), cart.getUser().getIdUser(), List.of());
44     }
45 }

CartService.java
1 package com.bazar.bazarbooks.service;
2
3 import java.util.ArrayList;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import com.bazar.bazarbooks.model.Cart;
9 import com.bazar.bazarbooks.model.User;
10 import com.bazar.bazarbooks.repository.CartRepository;
11 import com.bazar.bazarbooks.repository.UserRepository;
12
13 import jakarta.transaction.Transactional;
14
15 @Service
16 public class CartService {
17
18     @Autowired
19     private CartRepository cartRepository;
20
21     @Autowired
22     private UserRepository userRepository;
23 }

CartDTO.java
1 package com.bazar.bazarbooks.dto;
2
3 import java.util.List;
4
5 public class CartDTO {
6     private int id_cart;
7     private int userId;
8     private List<CartItemDTO> items;
9     private double total;
10
11     public CartDTO(int id_cart, int userId, List<CartItemDTO> items) {
12         this.id_cart = id_cart;
13         this.userId = userId;
14         this.items = items;
15         this.total = items.stream().mapToDouble(CartItemDTO::getSubtotal).sum();
16     }
17
18     public int getIdCart() { return id_cart; }
19     public int getUserId() { return userId; }
20     public List<CartItemDTO> getItems() { return items; }
21     public double getTotal() { return total; }
22 }

Cart.java
1 package com.bazar.bazarbooks.model;
2
3 import java.util.List;
4
5 import jakarta.persistence.*;
6
7 @Entity
8 public class Cart {
9
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     private int id_cart;
13
14     @ManyToOne
15     @JoinColumn(name = "user_id", referencedColumnName = "idUser", nullable = true)
16     private User user;
17
18     @OneToMany(mappedBy = "cart", cascade = CascadeType.ALL, orphanRemoval = true)
19     private List<CartItem> items;
20
21     public Cart() {}
22
23     public Cart(User user) {}
24 }
```

Classes Java para a Entidade de Cart

Soraya - CartItem:

```
CartController.java
21 public class CartController {
22     public CartDTO cartByUserId(@Argument int userId) {
23         Cart cart = cartService.getCartByUserId(userId);
24         List<CartItemDTO> itemDTOS = cart.getItems().stream()
25             .map(this::toDTO)
26             .collect(Collectors.toList());
27         return new CartDTO(cart.getId(), cart.getUser().getIdUser(), itemDTOS);
28     }
29
30     @GetMapping
31     public List<CartItemDTO> cartItemsByCartId(@Argument int cartId) {
32         return cartItemService.getItemsByCartId(cartId)
33             .stream()
34             .map(this::toDTO)
35             .collect(Collectors.toList());
36     }
37
38     // ===== MUTATIONS =====
39
40     @PostMapping
41     public CartDTO createCart(@Argument int userId) {
42         Cart cart = cartService.createCartIfNotExists(userId);
43         return new CartDTO(cart.getId(), cart.getUser().getIdUser(), List.of());
44     }
45 }

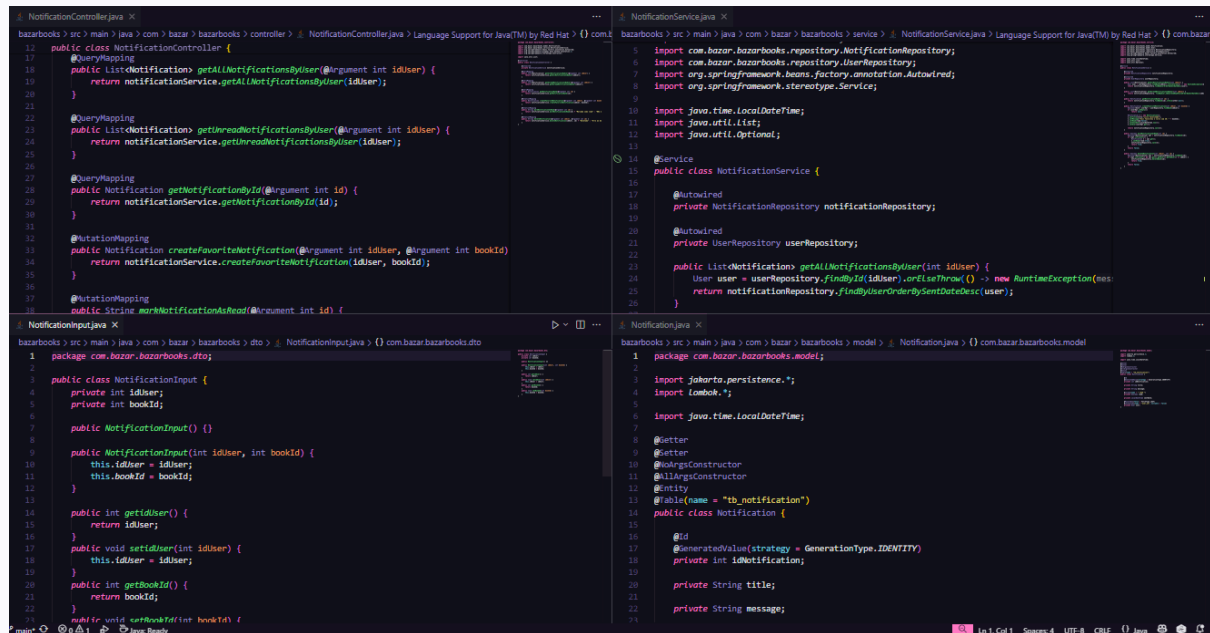
CartItemService.java
1 package com.bazar.bazarbooks.service;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import com.bazar.bazarbooks.model.Cart;
9 import com.bazar.bazarbooks.model.CartItem;
10 import com.bazar.bazarbooks.repository.CartItemRepository;
11 import com.bazar.bazarbooks.repository.CartRepository;
12
13 @Service
14 public class CartItemService {
15
16     @Autowired
17     private CartItemRepository cartItemRepository;
18
19     @Autowired
20     private CartRepository cartRepository;
21
22     // Método auxiliar para obter o cartinho ou lançar uma exceção se não existir
23 }

CartItemDTO.java
1 package com.bazar.bazarbooks.dto;
2
3 public class CartItemDTO {
4     private int id;
5     private int bookId;
6     private double unitPrice;
7     private int quantity;
8     private double subtotal;
9
10     public CartItemDTO(int id, int bookId, double unitPrice, int quantity) {
11         this.id = id;
12         this.bookId = bookId;
13         this.unitPrice = unitPrice;
14         this.quantity = quantity;
15         this.subtotal = unitPrice * quantity;
16     }
17
18     public int getId() { return id; }
19     public int getBookId() { return bookId; }
20     public double getUnitPrice() { return unitPrice; }
21     public int getQuantity() { return quantity; }
22     public double getSubtotal() { return subtotal; }
23 }

CartItem.java
1 package com.bazar.bazarbooks.model;
2
3 import jakarta.persistence.*;
4
5 @Entity
6 public class CartItem {
7
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10     private int id;
11
12     @ManyToOne(fetch = FetchType.LAZY)
13     @JoinColumn(name = "cart_id", nullable = false)
14     private Cart cart;
15
16     @Column(nullable = false)
17     private Integer bookId;
18
19     @Column(nullable = false)
20     private double unitPrice;
21
22     @Column(nullable = false)
23     private int quantity;
24 }
```

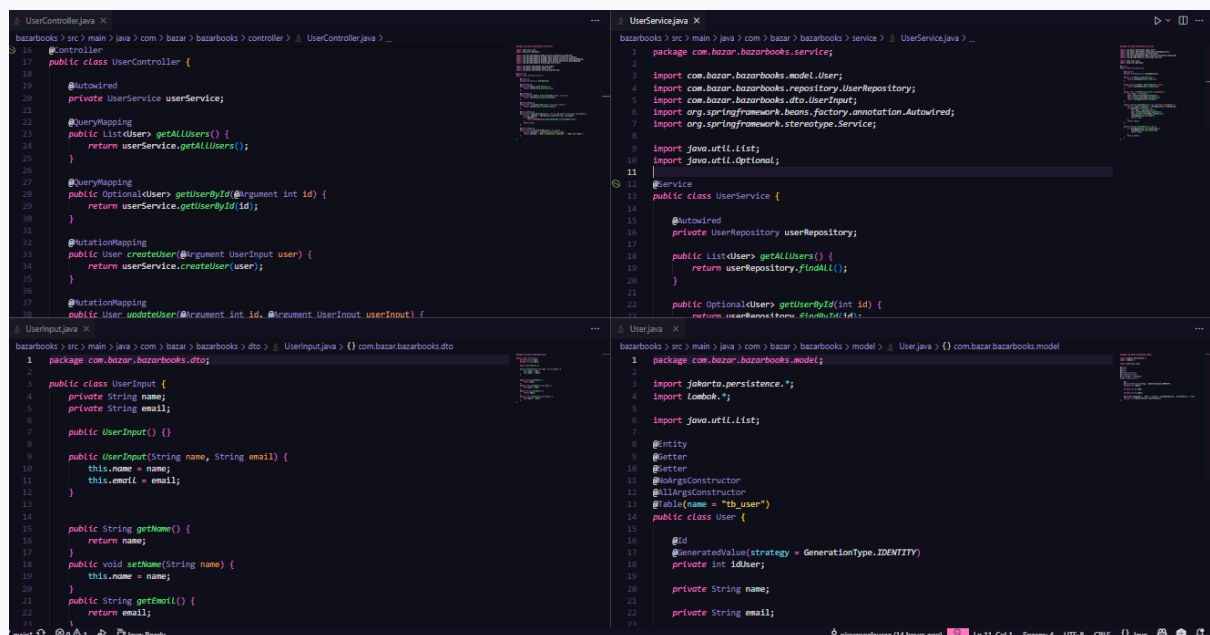
Classes Java para a Entidade de CartItem

Giovanna - Notification:



Classes Java para a Entidade de Notification

User:



Classes Java para a Entidade de User

Ana Laura - Author:

```

// AuthorController.java
public class AuthorController {
    @Autowired
    private AuthorRepository authorRepository;

    @GetMapping
    public List<Author> getAllAuthors() {
        return authorService.getAllAuthors();
    }

    @GetMapping
    public Author getAuthorById(@RequestParam int id) {
        return authorService.getAuthorById(id);
    }

    @PostMapping
    public Author createAuthor(@RequestBody AuthorInput authorInput) {
        Author author = authorService.createAuthor(authorInput);
        return author;
    }

    @PutMapping
    public Author updateAuthor(@RequestParam int id, @RequestBody AuthorInput authorInput) {
        boolean updated = authorService.updateAuthor(id, authorInput);
        if (updated) {
            return authorService.getAuthorById(id);
        }
        return null;
    }

    @DeleteMapping
    public boolean deleteAuthor(@RequestParam int id) {
        return authorService.deleteAuthor(id);
    }
}

// AuthorService.java
public class AuthorService {
    @Autowired
    private AuthorRepository authorRepository;

    @Autowired
    private BookRepository bookRepository;

    public List<Author> getAllAuthors() {
        return authorRepository.findAll();
    }

    public Author getAuthorById(int id) {
        return authorRepository.findById(id).orElse(null);
    }

    public Author createAuthor(AuthorInput authorInput) {
        Author author = new Author();
        author.setName(authorInput.getName());
        author.setDescription(authorInput.getDescription());
        author.setEmail(authorInput.getEmail());
        author.setBooks(authorInput.getBooks());
        return authorRepository.save(author);
    }

    @Transactional
    public boolean deleteAuthor(int id) {
        Optional<Author> optionalAuthor = authorRepository.findById(id);
        if (optionalAuthor.isPresent()) {
            Author author = optionalAuthor.get();
            List<Book> books = author.getBooks();
            if (books != null && !books.isEmpty()) {
                for (Book livro : books) {
                    livro.setAuthor(null);
                    bookRepository.save(livro);
                }
                bookRepository.flush();
            }
            return authorRepository.delete(author);
        }
        return false;
    }
}

// Author.java
@Entity
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private String email;
    private String description;
    @OneToMany(mappedBy = "author", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Book> books;
}

```

Classes Java para a Entidade de Author

2 Evidências dos Testes Funcionais

As seguintes seções apresentam as evidências dos testes funcionais, realizados utilizando o Thunder Client no VS Code. Cada imagem corresponde a uma requisição e sua respectiva resposta, demonstrando o comportamento esperado (e alguns erros comuns) dos endpoints.

2.1 Entidade Address

The screenshot shows the GraphQL Playground interface. The URL is `http://localhost:8080/graphql`. The method is `POST`. The query is a `mutation` named `CreateNewAddress` that takes an `AddressInput!` argument. The response is a `200 OK` status with a size of 187 Bytes and a time of 223 ms. The response body is a JSON object with a `data` field containing a `createAddress` object with fields `id`, `street`, `number`, `city`, `state`, `postalCode`, and `user`.

```
mutation CreateNewAddress($addressInput: AddressInput!) {
  createAddress(addressInput: $addressInput) {
    id
    street
    number
    city
    state
    postalCode
    user {
      idUser
      name
    }
  }
}
```

```
{
  "data": {
    "createAddress": {
      "id": "1",
      "street": "Av. Maj. Fernando Valle",
      "number": 2013,
      "city": "Bragança Paulista",
      "state": "SP",
      "postalCode": "12903-000",
      "user": {
        "idUser": 1,
        "name": "João"
      }
    }
  }
}
```

Exemplo de mutation para criação de um Address

The screenshot shows the GraphQL Playground interface. The URL is `http://localhost:8080/graphql`. The method is `POST`. The query is a `query` named `GetAllAddresses` that returns a list of `Address` objects. The response is a `200 OK` status with a size of 166 Bytes and a time of 137 ms. The response body is a JSON object with a `data` field containing a `getAllAddresses` array with one object containing fields `id`, `street`, `number`, `city`, `state`, `postalCode`, and `user`.

```
query GetAllAddresses {
  getAllAddresses {
    id
    street
    number
    city
    state
    postalCode
    user {
      idUser
      name
    }
  }
}
```

```
{
  "data": {
    "getAllAddresses": [
      {
        "id": "1",
        "street": "Av. Maj. Fernando Valle",
        "number": 2013,
        "city": "Bragança Paulista",
        "state": "SP",
        "postalCode": "12903-000",
        "user": {
          "idUser": 1,
          "name": "João"
        }
      }
    ]
  }
}
```

Exemplo de query para listar todos os Address

The screenshot shows the GraphQL Playground interface. The top bar indicates a POST request to `http://localhost8080/graphql` with a status of 200 OK, size of 180 Bytes, and time of 230 ms. The 'Query' tab is active, showing the following GraphQL query:

```
1 mutation UpdateExistingAddress($id: ID!, $addressInput: AddressInput!) {  
2   updateAddress(id: $id, addressInput: $addressInput) {  
3     id  
4     street  
5     number  
6     complement  
7     city  
8     user {  
9       idUser  
10    }  
11  }  
12 }
```

The 'Variables' tab shows the input variables:

```
1 {  
2   "id": 1,  
3   "addressInput": {  
4     "street": "Av. Major Fernando Valle",  
5     "number": 2013,  
6     "complement": "Próximo ao Portão Principal",  
7     "district": "São Miguel",  
8     "city": "Bragança Paulista",  
9     "state": "SP",  
10    "postalCode": "12083-000",  
11    "latitude": -22.924862382065738,  
12    "longitude": -46.55851243195008,  
13    "userId": 1  
14  }  
15 }
```

The 'Response' tab shows the JSON output:

```
1 {  
2   "data": {  
3     "updateAddress": {  
4       "id": "1",  
5       "street": "Av. Major Fernando Valle",  
6       "number": 2013,  
7       "complement": "Próximo ao Portão Principal",  
8       "city": "Bragança Paulista",  
9       "user": {  
10        "idUser": 1  
11      }  
12    }  
13  }  
14 }
```

Exemplo de mutation para atualizar um Address

The screenshot shows the GraphQL Playground interface. The top bar indicates a POST request to `http://localhost8080/graphql` with a status of 200 OK, size of 59 Bytes, and time of 254 s. The 'Query' tab is active, showing the following GraphQL query:

```
1 mutation DeleteExistingAddress($id: ID!) {  
2   deleteAddress(id: $id)  
3 }
```

The 'Variables' tab shows the input variables:

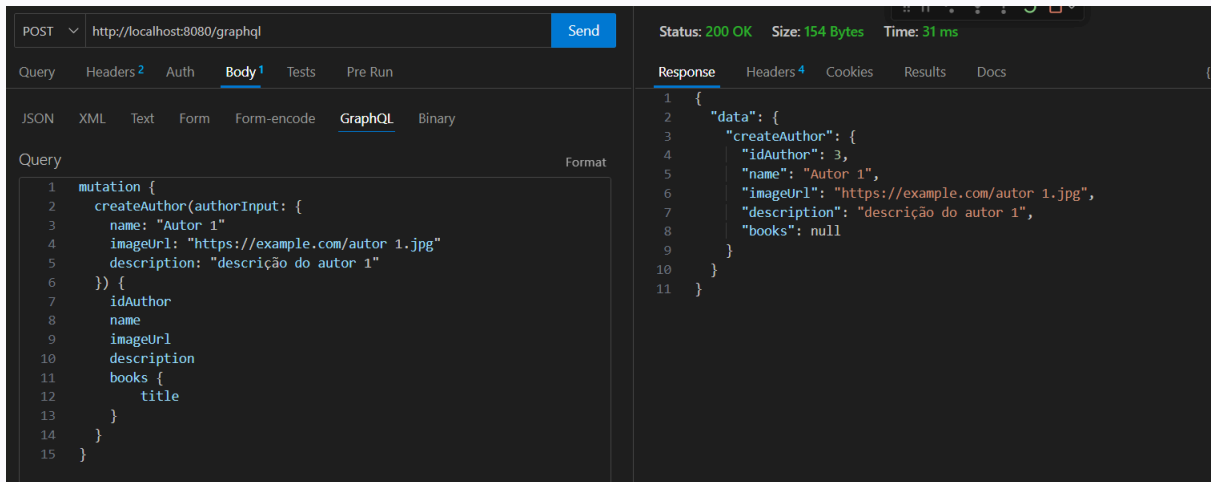
```
1 {  
2   "id": 1  
3 }
```

The 'Response' tab shows the JSON output:

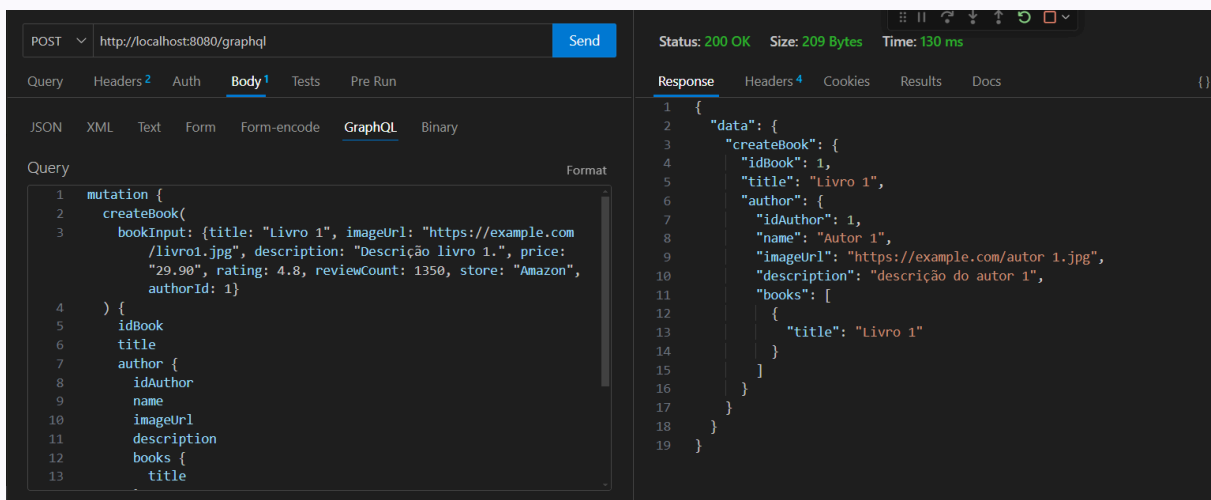
```
1 {  
2   "data": {  
3     "deleteAddress": "Endereço apagado com sucesso!"  
4   }  
5 }
```

Exemplo de mutation para deletar um Address

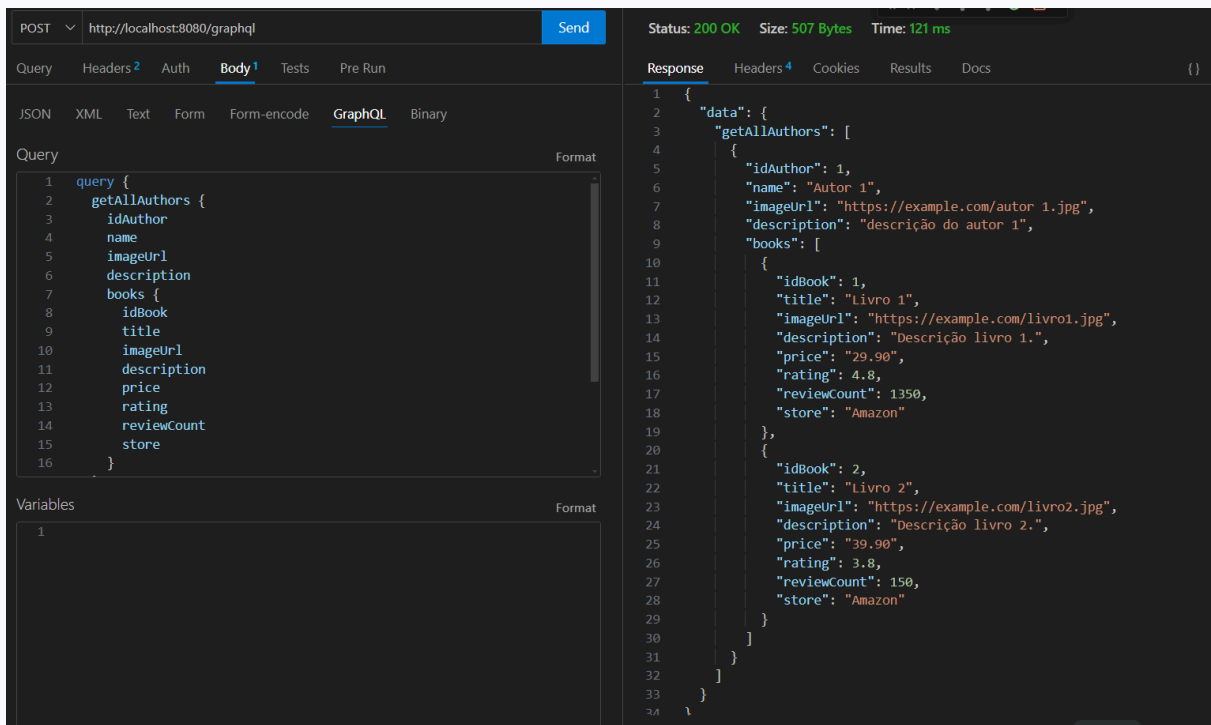
2.2 Entidade Author



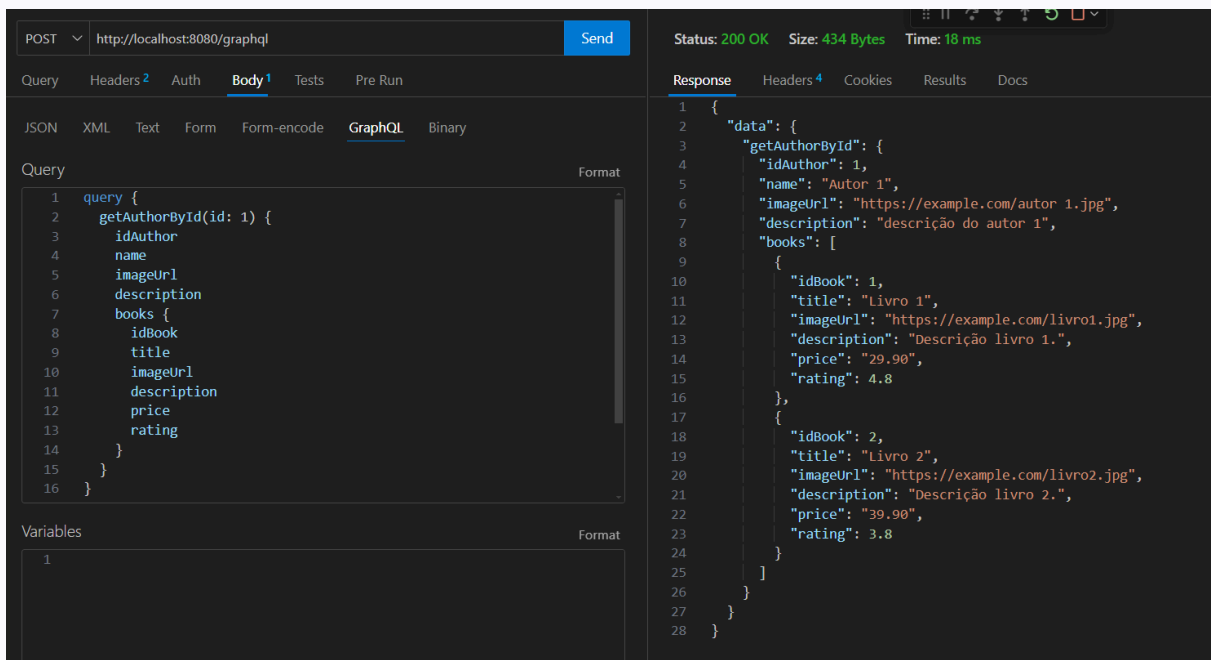
Exemplo de mutation para criar um Author



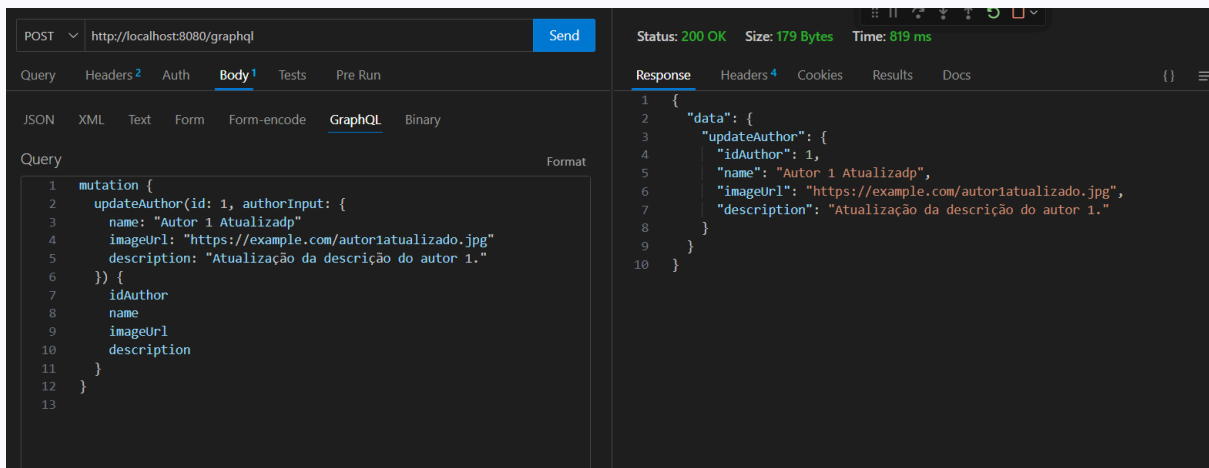
Exemplo de mutation para criar um Book que recebe um Author



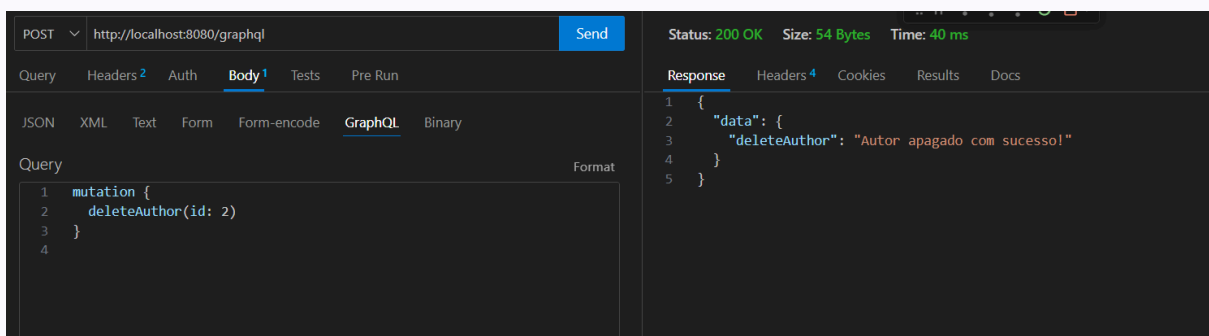
Exemplo de query para listar todos os Authors



Exemplo de query para listar o Author com id específico



Exemplo de mutation para atualizar um Author



Exemplo de mutation para deletar um Author

2.3 Entidade Cart

The screenshot shows the GraphQL Playground interface. The URL is `http://localhost8080/graphql` and the method is `POST`. The query is a mutation to create a cart for a user with ID 1. The response is a 200 OK status with 73 bytes and a time of 16 ms. The response body is a JSON object containing the created cart details.

```
POST http://localhost8080/graphql Send
```

Query Headers² Auth Body¹ Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

Query

```
1 mutation {
2   createCart(userId: 1) {
3     id_cart
4     userId
5     total
6     items {
7       id
8       bookId
9       quantity
10    }
11  }
12 }
```

Format

Status: 200 OK Size: 73 Bytes Time: 16 ms

Response Headers⁴ Cookies Results Docs

```
1 {
2   "data": {
3     "createCart": {
4       "id_cart": "1",
5       "userId": 1,
6       "total": 0.0,
7       "items": []
8     }
9   }
10 }
```

Exemplo de mutation para insert de Cart

The screenshot shows the GraphQL Playground interface. The URL is `http://localhost8080/graphql` and the method is `POST`. The query is a query to retrieve the cart for a user with ID 1. The response is a 200 OK status with 212 bytes and a time of 14 ms. The response body is a JSON object containing the cart details and its items.

```
POST http://localhost8080/graphql Send
```

Query Headers² Auth Body¹ Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

Query

```
1 query {
2   cartByUserId(userId: 1) {
3     id_cart
4     userId
5     total
6     items {
7       id
8       bookId
9       unitPrice
10      quantity
11      subtotal
12    }
13  }
14 }
```

Format

Variables

```
1 {
```

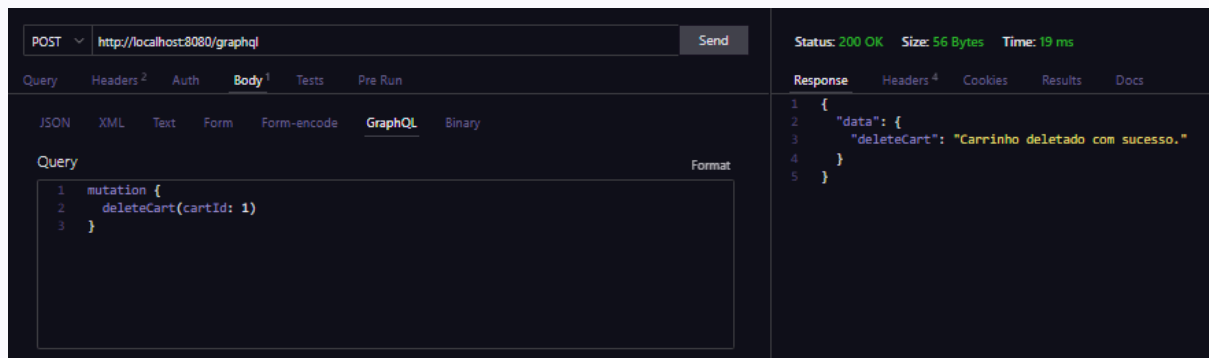
Format

Status: 200 OK Size: 212 Bytes Time: 14 ms

Response Headers⁴ Cookies Results Docs

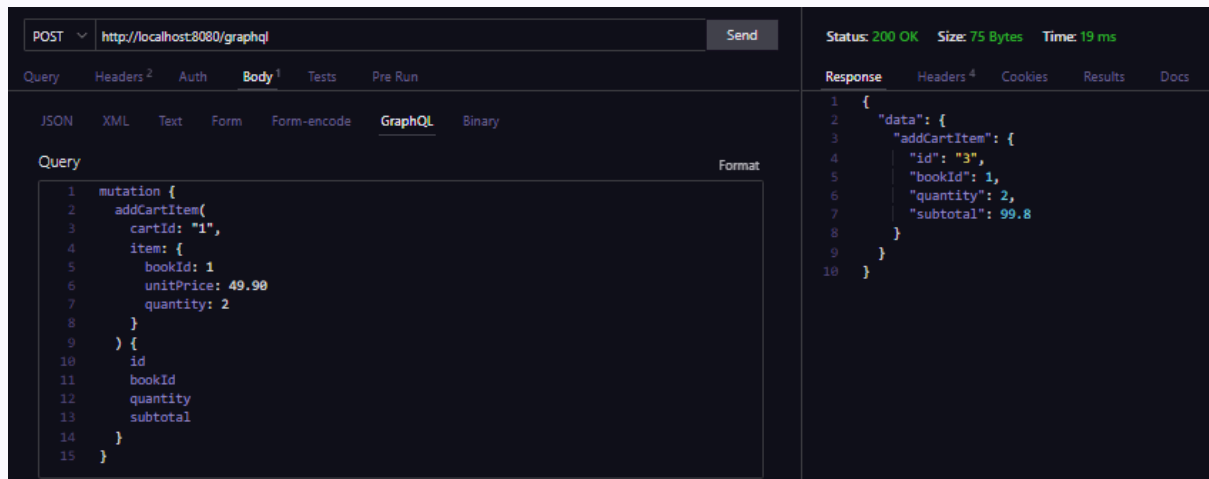
```
1 {
2   "data": {
3     "cartByUserId": {
4       "id_cart": "1",
5       "userId": 1,
6       "total": 199.6,
7       "items": [
8         {
9           "id": "1",
10          "bookId": 1,
11          "unitPrice": 49.9,
12          "quantity": 2,
13          "subtotal": 99.8
14        },
15        {
16          "id": "2",
17          "bookId": 1,
18          "unitPrice": 49.9,
19          "quantity": 2,
20          "subtotal": 99.8
21        }
22      ]
23    }
24  }
25 }
```

Exemplo de query para Cart

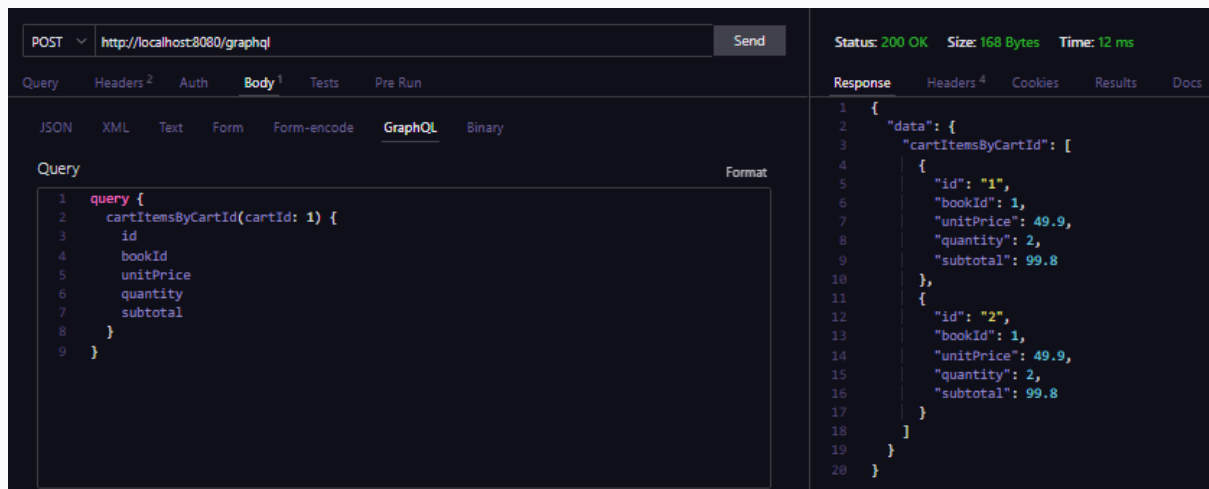


Exemplo de mutation para exclusão de Cart

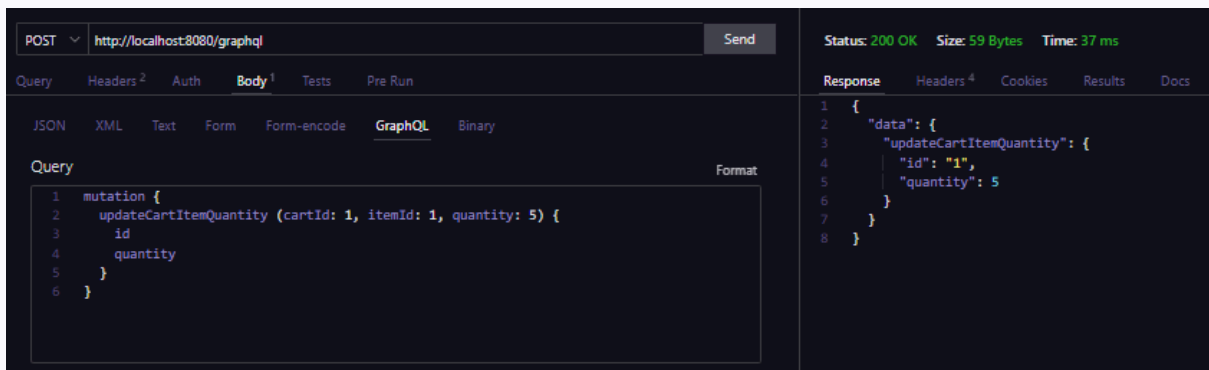
2.4 Entidade CartItem



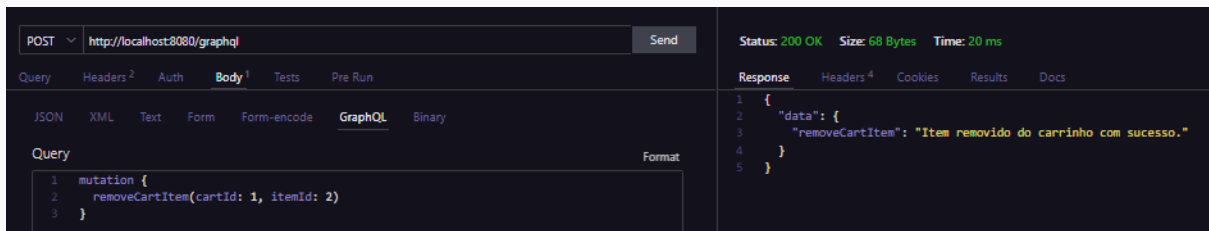
Exemplo de mutation para inserção de CartItem



Exemplo de query para CartItem

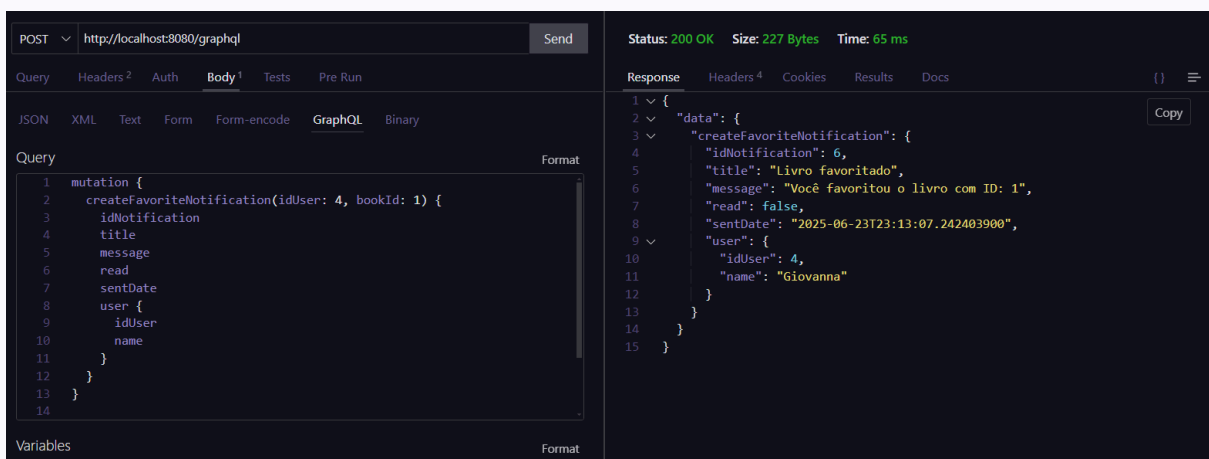


Exemplo de mutation para atualização de CartItem



Exemplo de mutation para exclusão de CartItem

Entidade Notification



Exemplo de mutation para inserção de Notification

The screenshot shows a GraphQL Playground interface. The URL is `http://localhost:8080/graphql`. The query is a `query` named `getAllNotificationsByUser` with a variable `idUser: 4`. The response is a JSON object with a `data` field containing an array of notifications for the user with ID 4.

```
POST http://localhost:8080/graphql Send
```

Query Headers Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

Query

```
1 query {
2   getAllNotificationsByUser(idUser: 4) {
3     idNotification
4     title
5     message
6     read
7     sentDate
8   }
9 }
10
```

Variables

```
1
```

Status: 200 OK Size: 333 Bytes Time: 94 ms

Response Headers Cookies Results Docs

```
1 {
2   "data": {
3     "getAllNotificationsByUser": [
4       {
5         "idNotification": 7,
6         "title": "Livro favorito",
7         "message": "Você favoritou o livro com ID: 1",
8         "read": true,
9         "sentDate": "2025-06-23T23:14:12.520493"
10      },
11      {
12        "idNotification": 6,
13        "title": "Livro favorito",
14        "message": "Você favoritou o livro com ID: 1",
15        "read": false,
16        "sentDate": "2025-06-23T23:13:07.242404"
17      }
18    ]
19  }
20 }
```

Exemplo de query para Notification - getAllNotifications

The screenshot shows a GraphQL Playground interface. The URL is `http://localhost:8080/graphql`. The query is a `mutation` named `markNotificationAsRead` with a variable `id: 7`. The response is a JSON object with a `data` field containing a `markNotificationAsRead` field with the value `"Marcada como lida"`.

```
POST http://localhost:8080/graphql Send
```

Query Headers Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

Query

```
1 mutation {
2   markNotificationAsRead(id: 7)
3 }
4
```

Status: 200 OK Size: 55 Bytes Time: 336 ms

Response Headers Cookies Results Docs

```
1 {
2   "data": {
3     "markNotificationAsRead": "Marcada como lida"
4   }
5 }
```

Exemplo de query para Notification - markNotificationAsRead

The screenshot shows a GraphQL Playground interface. The URL is `http://localhost:8080/graphql`. The query is a `query` named `getUnreadNotificationsByUser` with a variable `idUser: 4`. The response is a JSON object with a `data` field containing an array of unread notifications for the user with ID 4.

```
POST http://localhost:8080/graphql Send
```

Query Headers Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

Query

```
1 query {
2   getUnreadNotificationsByUser(idUser: 4) {
3     idNotification
4     title
5     read
6   }
7 }
8
```

Status: 200 OK Size: 104 Bytes Time: 104 ms

Response Headers Cookies Results Docs

```
1 {
2   "data": {
3     "getUnreadNotificationsByUser": [
4       {
5         "idNotification": 6,
6         "title": "Livro favorito",
7         "read": false
8       }
9     ]
10  }
11 }
```

Exemplo de query para Notification - getUnreadNotificationByUser

The screenshot shows a GraphQL query being executed in a playground. The query is a GET request to `http://localhost:8080/graphql`. The query text is:

```
1 query {
2   getNotificationById(id: 7) {
3     idNotification
4     title
5     message
6     user {
7       name
8     }
9   }
10 }
11
```

The response is a 200 OK status with a size of 153 Bytes and a time of 20 ms. The response body is:

```
1 {
2   "data": {
3     "getNotificationById": {
4       "idNotification": 7,
5       "title": "Livro favorito",
6       "message": "Você favoritou o livro com ID: 1",
7       "user": {
8         "name": "Giovanna"
9       }
10     }
11   }
12 }
```

Exemplo de query para Notification - `getNotificationById`

The screenshot shows a GraphQL mutation being executed in a playground. The mutation is a POST request to `http://localhost:8080/graphql`. The mutation text is:

```
1 mutation {
2   deleteNotification(idUser: 4, id: 7)
3 }
4
```

The response is a 200 OK status with a size of 42 Bytes and a time of 263 ms. The response body is:

```
1 {
2   "data": {
3     "deleteNotification": "Deletada"
4   }
5 }
```

Exemplo de query para Notification - `deleteNotification`