package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.mapper.BookMapper;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.service.BookService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookService bookService;

@Autowired

private BookMapper bookMapper;

// CREATE

@PostMapping

public ResponseEntity<BookDTO> addBook(@Valid @RequestBody BookDTO bookDTO) {

Book book = bookMapper.toBook(bookDTO);

Book savedBook = bookService.addBook(book);

return new ResponseEntity<>(bookMapper.toBookDTO(savedBook), HttpStatus.CREATED);

}

// READ

@GetMapping("/{id}")

public ResponseEntity<BookDTO> getBookById(@PathVariable Long id) {

Book book = bookService.getBookById(id)

.orElseThrow(() -> new RuntimeException("Book not found with id " + id));

return ResponseEntity.ok(bookMapper.toBookDTO(book));

}

@GetMapping

public List<BookDTO> getAllBooks() {

return bookService.getAllBooks().stream()

.map(bookMapper::toBookDTO)

.collect(Collectors.toList());

}

// UPDATE

@PutMapping("/{id}")

public ResponseEntity<BookDTO> updateBook(@PathVariable Long id, @Valid @RequestBody BookDTO bookDTO) {

Book updatedBook = bookService.updateBook(id, bookMapper.toBook(bookDTO));

return ResponseEntity.ok(bookMapper.toBookDTO(updatedBook));

}

// DELETE

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteBook(@PathVariable Long id) {

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.mapper.CustomerMapper;

import com.example.bookstoreapi.model.Customer;

import com.example.bookstoreapi.service.CustomerService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@Autowired

private CustomerMapper customerMapper;

// CREATE

@PostMapping

public ResponseEntity<CustomerDTO> addCustomer(@Valid @RequestBody CustomerDTO customerDTO) {

Customer customer = customerMapper.toCustomer(customerDTO);

Customer savedCustomer = customerService.addCustomer(customer);

return new ResponseEntity<>(customerMapper.toCustomerDTO(savedCustomer), HttpStatus.CREATED);

}

// READ

@GetMapping("/{id}")

public ResponseEntity<CustomerDTO> getCustomerById(@PathVariable Long id) {

Customer customer = customerService.getCustomerById(id)

.orElseThrow(() -> new RuntimeException("Customer not found with id " + id));

return ResponseEntity.ok(customerMapper.toCustomerDTO(customer));

}

@GetMapping

public List<CustomerDTO> getAllCustomers() {

return customerService.getAllCustomers().stream()

.map(customerMapper::toCustomerDTO)

.collect(Collectors.toList());

}

// UPDATE

@PutMapping("/{id}")

public ResponseEntity<CustomerDTO> updateCustomer(@PathVariable Long id, @Valid @RequestBody CustomerDTO customerDTO) {

Customer updatedCustomer = customerService.updateCustomer(id, customerMapper.toCustomer(customerDTO));

return ResponseEntity.ok(customerMapper.toCustomerDTO(updatedCustomer));

}

// DELETE

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteCustomer(@PathVariable Long id) {

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}

package com.example.bookstoreapi.dto;

import lombok.Data;

import javax.validation.constraints.Min;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

@Data

public class BookDTO {

private Long id;

@NotNull(message = "Title is required")

@Size(min = 1, max = 255, message = "Title must be between 1 and 255 characters")

private String title;

@NotNull(message = "Author is required")

@Size(min = 1, max = 255, message = "Author must be between 1 and 255 characters")

private String author;

@NotNull(message = "Price is required")

@Min(value = 0, message = "Price must be a positive number")

private Double price;

@NotNull(message = "ISBN is required")

@Size(min = 10, max = 13, message = "ISBN must be between 10 and 13 characters")

private String isbn;

}

package com.example.bookstoreapi.dto;

import lombok.Data;

import javax.validation.constraints.Email;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

@Data

public class CustomerDTO {

private Long id;

@NotNull(message = "Name is required")

@Size(min = 1, max = 255, message = "Name must be between 1 and 255 characters")

private String name;

@NotNull(message = "Email is required")

@Email(message = "Email should be valid")

private String email;

@NotNull(message = "Address is required")

@Size(min = 1, max = 255, message = "Address must be between 1 and

package com.example.bookstoreapi.model;

import lombok.Data;

import javax.persistence.\*;

@Data

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private Double price;

private String isbn;

@Version

private int version; // This field is used for optimistic locking

}

package com.example.bookstoreapi.model;

import lombok.Data;

import javax.persistence.\*;

@Data

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String address;

@Version

private int version; // This field is used for optimistic locking

}