DTO (Data Transfer Object)

فصل المسؤوليات :

1. الكيانات مقابل كائنات نقل البيانات (DTOs):

- الكيانات (Entities): تمثل هيكل قاعدة البيانات وتعكس جداولها.
- كائنات نقل البيانات (DTOs): مصممة خصيصًا لنقل البيانات بين الطبقات، وتخفي التفاصيل غير الضرورية عن هيكل الكيانات.

2. الفصل بين الطبقات:

استخدام DTOs يضمن وجود فصل واضح بين طبقة منطق التخزين (مثل قواعد البيانات) وواجهة البرمجة الخارجية (API)، مما يعزز من قابلية تطوير النظام.

التغليف والأمان

1. البيانات الحساسة:

يمكن لـ DTOs استبعاد الحقول الحساسة مثل كلمات المرور أو المعرفات الداخلية، مما يضمن حماية البيانات في استجابات API.

2. التحكم في الوصول:

توفر DTOs تحكمًا دقيقًا في البيانات التي يمكن قراءتها أو تعديلها من قبل المستخدمين.

تحسين الأداء

1. نقل البيانات الانتقائي:

يتم تضمين الحقول ذات الصلة فقط في DTOs، مما يقلل من حجم البيانات المنقولة عبر الشبكة.

2. الهياكل المخصصة:

يمكن لـ DTOs الجمع بين البيانات من كيانات أو مصادر متعددة، مما يبسط التفاعل بين العميل والخادم.

المرونة لتطوير واجهات API

1. استقرار الواجهة البرمجية:

تمكن DTOs من تطوير واجهة API بشكل مستقل عن مخطط قاعدة البيانات.

2. مرونة التغيير:

إذا تغيرت قاعدة البيانات، يمكن تعديل منطق التعيين (Mapping) مع الحفاظ على هيكل DTO ثابت، مما يمنع حدوث تغييرات تؤثر على العملاء.

3. التحقق من البيانات

يمكن لـ DTOs تضمين قواعد التحقق الخاصة بمدخلات API، مما يفصلها عن منطق الأعمال وطبقة التخزين.

وضوح وصيانة الكود

1. تنظيم الكود:

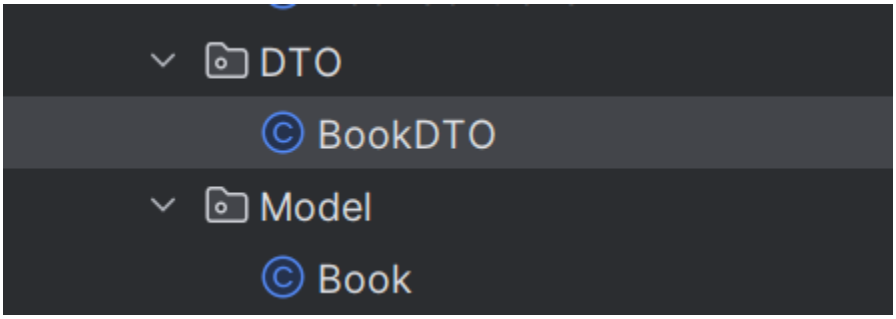
وجود طبقة منفصلة لـ DTOs يجعل قاعدة الكود أكثر تنظيمًا وسهولة في الفهم.

2. تقليل التعديلات:

تغييرات مثل تحديثات مخطط قاعدة البيانات تتطلب تعديلات أقل في الطبقات الأخرى.

التطبيق:

1- انشئ مجلد باسم DTO



2- انشئ ملف جافا باسم BookDTO, بداخله الصفات المسموح أن تظهر للمستخدم , مثل الكاتب , السعر , العنوان الخ.

```java
21 usages
@Data
@AllArgsConstructor
public class BookDTO {

    @NotEmpty(message = "Title is mandatory")
    private String title;

    @NotEmpty(message = "Author is mandatory")
    private String author;

    @NotNull(message = "Price cannot be null")
    @Positive(message = "Price must be a positive value")
    private int price;
}
```

3- عند اضافة اي Book سوف يأتي في طبقة التحكم (Controller Layer) بشكل كائن من نوع BookDTO

```java
// Add a new book
@PostMapping("add-book")
public ResponseEntity addBook(@RequestBody BookDTO bookDTO,
                              @RequestParam String isbn,
                              @RequestParam String publisherInternalNotes) {
    bookService.addBook(bookDTO, isbn, publisherInternalNotes);
    return ResponseEntity.ok(new ApiResponse("Added Successfully"));
}
```

4- في طبقة الخدمة Service Layer سوف نقوم بإنشاء كائن من نوع Book و مليئة بالبيانات الموجودة في BookDTO و ايضا القادمة

```java
// 2- Add a new book
1 usage
public void addBook(BookDTO bookDTO, String isbn, String publisherInternalNotes) {
    Book book = new Book(
            id: null,                          // ID (auto-generated)
            bookDTO.getTitle(),                // Book title from the DTO
            bookDTO.getAuthor(),               // Book author from the DTO
            bookDTO.getPrice(),                // Book price from the DTO
            isbn,                              // ISBN provided as a parameter
            publisherInternalNotes             // Publisher's internal notes provided
    );
    // Save the new Book object to the database.
    bookRepository.save(book);

}
```

5- عند تحديث اي Book سوف يأتي في طبقة التحكم (Controller Layer) بشكل كائن من نوع BookDTO

```java
@PutMapping("update-book/{id}")
public ResponseEntity updateBook(@PathVariable Integer id, @RequestBody BookDTO bookDTO,
                                 @RequestParam String isbn,
                                 @RequestParam String publisherInternalNotes) {
    bookService.updateBook(id, bookDTO, isbn, publisherInternalNotes);
    return ResponseEntity.status(HttpStatus.OK).body("Book updated Successfully");
}
```

6- في طبقة الخدمة Service Layer سوف نقوم باسترجاع كائن من نوع Book عن طريق ID و تحديثه بالبيانات الموجودة في BookDTO و ايضا القادمة

```java
// 3- Update a book by ID

1 usage
public void updateBook(Integer id, BookDTO bookDTO, String isbn, String publisherInternalNotes) {
    // Retrieve the existing Book object from the database using the provided ID.
    Book book = bookRepository.findBookById(id);

    // Check if the book exists.
    // If not, throw an exception with an appropriate message.
    if (book == null)
        throw new ApiException("Book not found");

    // Update the book's properties with the new values from the provided parameters and DTO.
    book.setAuthor(bookDTO.getAuthor());              // Update the author from the DTO.
    book.setTitle(bookDTO.getTitle());                // Update the title from the DTO.
    book.setPublisherInternalNotes(publisherInternalNotes); // Update the publisher's internal notes.
    book.setIsbn(isbn);                               // Update the ISBN.

    // Save the updated Book object back to the database.
    bookRepository.save(book);
}
```

7- عند طلب استرجاع اي كائن Book من طبقة التحكم (Controller Layer) سوف يسترجع بشكل BookDTO

```java
//================================== Extra Endpoint ================================
    // Endpoint to get book by ISBN
    @GetMapping("/isbn/public/{isbn}")
    public ResponseEntity<BookDTO> findBookByIsbn(@PathVariable String isbn) {
        BookDTO bookDTO = bookService.findBookByIsbn(isbn);
        return ResponseEntity.ok(bookDTO);
    }


    // Get a book by ID
    @GetMapping("/get-book/{id}")
    public ResponseEntity<BookDTO> getBookById(@PathVariable Integer id) {
        return ResponseEntity.ok(bookService.getBookById(id));
    }
```

8- في طبقة الخدمة Service Layer سوف نقوم بإنشاء كائن من نوع BookDTO و مليئة بالبيانات الموجودة في Book ومن ثم يسترجع

```java
// get Book by Isbn
1 usage
public BookDTO findBookByIsbn(String isbn) {
    Book book = bookRepository.findBookByIsbn(isbn);
    if (book == null) {
        throw new ApiException("Book not found");
    } else {
        return new BookDTO(book.getTitle(), book.getAuthor(), book.getPrice());
    }
}
// Get a book by ID
1 usage
public BookDTO getBookById(Integer id) {
    Book book = bookRepository.findBookById(id);
    if (book == null) {
        throw new ApiException("Book not found");
    } else {
        return new BookDTO(book.getTitle(), book.getAuthor(), book.getPrice());
    }
}
```

9- عند طلب استرجاع قائمة من كائنات Book من طبقة التحكم (Controller Layer) سوف يسترجع بشكل قائمة من نوع BookDTO

```java
// Get all books
@GetMapping("get-all")
public ResponseEntity<List<BookDTO>> getAllBooks() {

    return ResponseEntity.ok(bookService.getAllBooks());
}
```

10- في طبقة الخدمة Service Layer سوف نقوم بإنشاء قائمة كائنات من نوع BookDTO و مليئة بالبيانات الموجودة في قائمة الكائنات Book ومن ثم تسترجع

```java
// 1- Retrieve all books as DTOs
1 usage
public List<BookDTO> getAllBooks() {
    List<Book> books = bookRepository.findAll();
    List<BookDTO> bookDTOs = new ArrayList<>();
    for (Book book : books) {
        BookDTO bookDTO = new BookDTO(book.getTitle(), book.getAuthor(), book.getPrice());
        bookDTOs.add(bookDTO);
    }
    return bookDTOs;
}
```