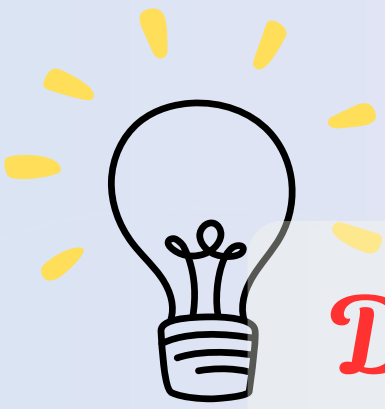
Suraj Kumar

Java & Spring Boot



DTO (Data Transfer Object)

Swipe for more

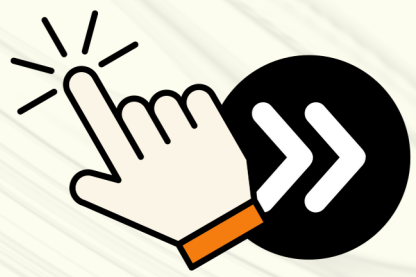
DTO (Data Transfer Object)

A DTO (Data Transfer Object) is a simple object used to transfer data between layers of an application without exposing the internal entity structure.

Key Benefits:

- Encapsulation – Prevents exposing entity models.
- Security – Avoids exposing sensitive fields.
- Performance – Reduces unnecessary data in API responses.

Swipe for more



The Problem Without DTOs

What happens when we expose entities directly in APIs?

- **Overexposure** – Sensitive data (e.g., passwords) may be exposed.
- **Unstructured Data** – Clients receive extra, unused fields.
- **Tight Coupling** – API changes impact database schema.

```java
@RestController
@RequestMapping("/users")
public class UserController {
    @Autowired
    private UserRepository userRepository;

    @GetMapping
    public List<User> getUsers() {
        return userRepository.findAll(); // Exposes full entity!
    }
}
```

Swipe for more

Using DTOs in Spring Boot

- DTO separates entity from API response.
- Restricts unnecessary data exposure.
- Simplifies API versioning & validation.

Swipe for more

Define an Entity Class

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String email;
    private String password; // Should not be exposed!

    // Getters and Setters
}
```

Explanation:

- The User entity contains sensitive data, which should not be exposed directly.

Swipe for more

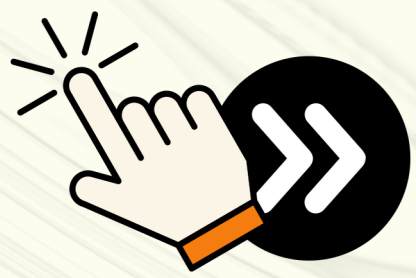
Create a DTO Class

```
public class UserDTO {
    private String username;
    private String email;

    // Constructor
    public UserDTO(String username, String email) {
        this.username = username;
        this.email = email;
    }

    // Getters
}
```

Explanation:

- The DTO contains only the necessary fields (username & email).
- Excludes sensitive data like password.

Swipe for more



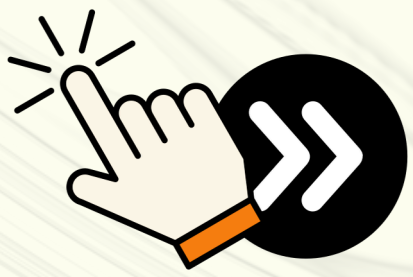
Convert Entity to DTO in Service Layer

```
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public List<UserDTO> getAllUsers() {
        return userRepository.findAll()
                .stream()
                .map(user -> new UserDTO(user.getUsername(), user
                    .getEmail()))
                .collect(Collectors.toList());
    }
}
```

Explanation:

- Uses Stream API to map entities to DTOs efficiently.

Swipe for more



Return DTO in Controller

```
@RestController
@RequestMapping("/users")
public class UserController {
    @Autowired
    private UserService userService;

    @GetMapping
    public List<UserDTO> getUsers() {
        return userService.getAllUsers();
    }
}
```

Explanation:

- The API now returns only DTOs, ensuring security & optimized data.

Swipe for more

Suraj Kumar

If you find this helpful, please like and share it with your friends