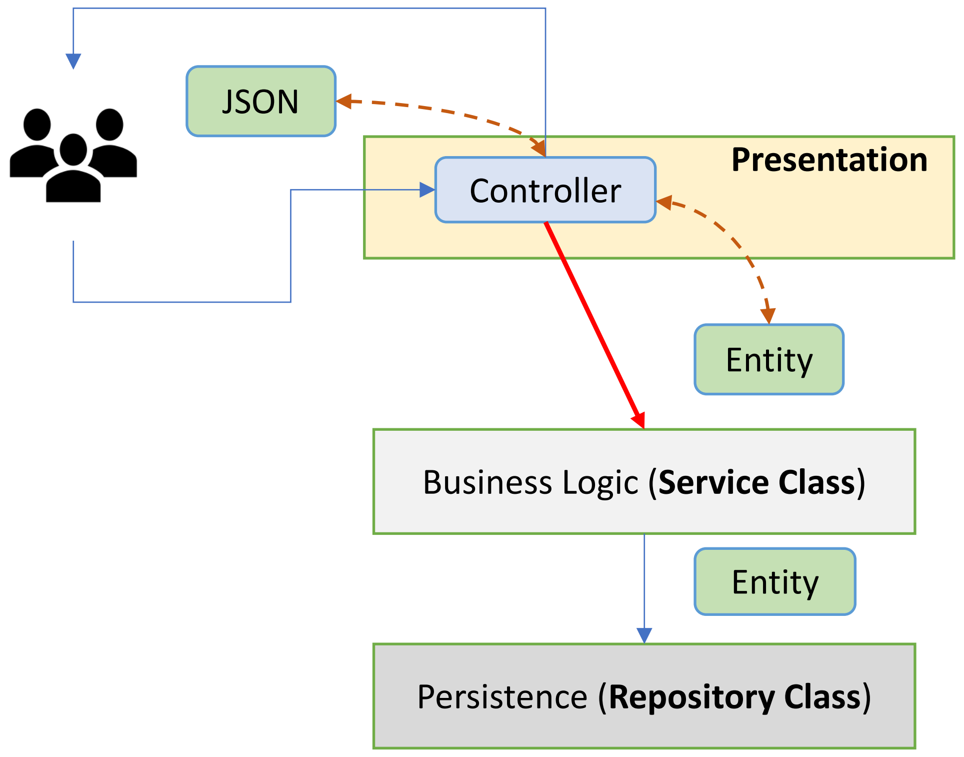
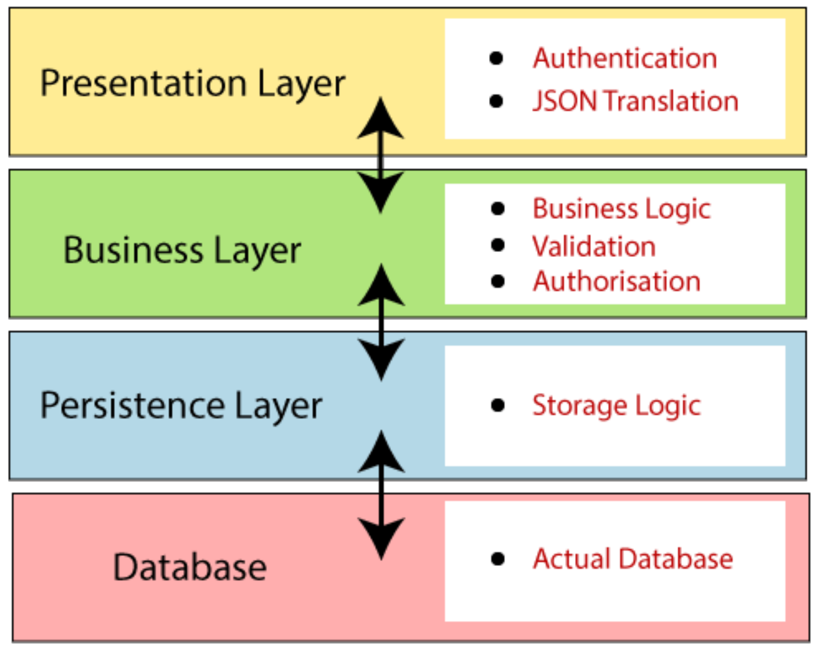
Spring RESTful API DTO

By

Pichet Limvajiranan

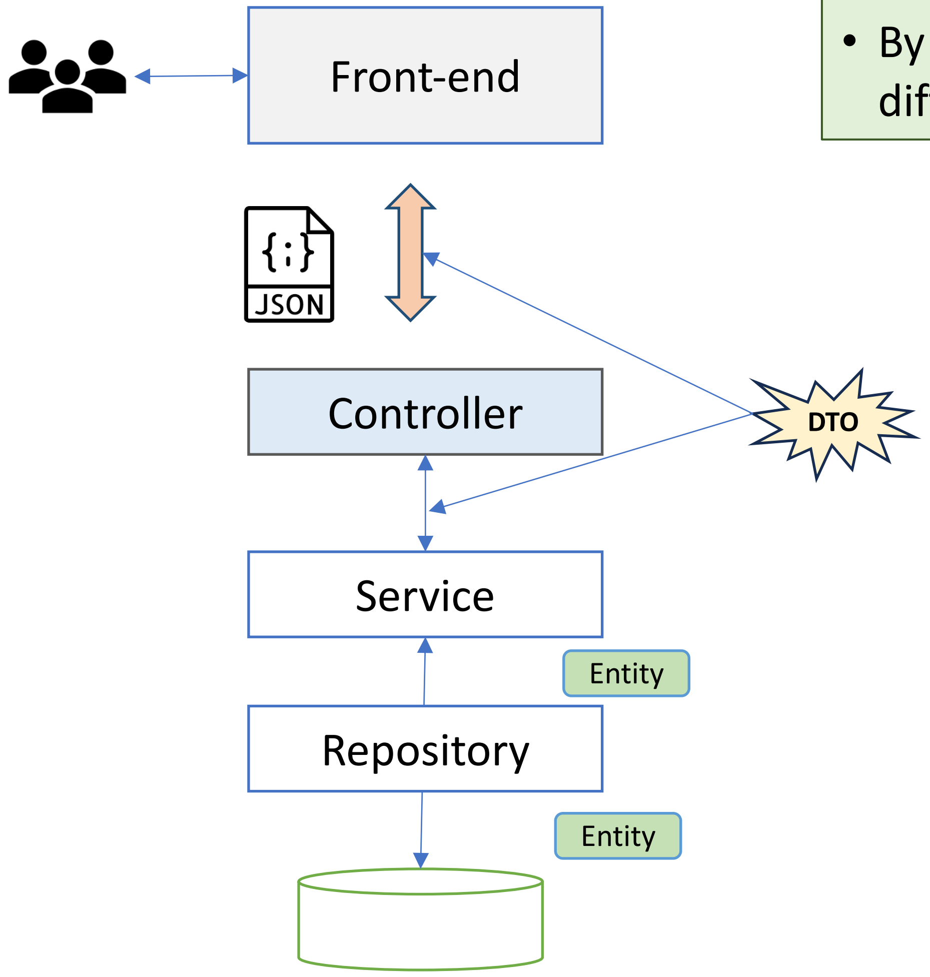
Spring Boot Layer Architectures

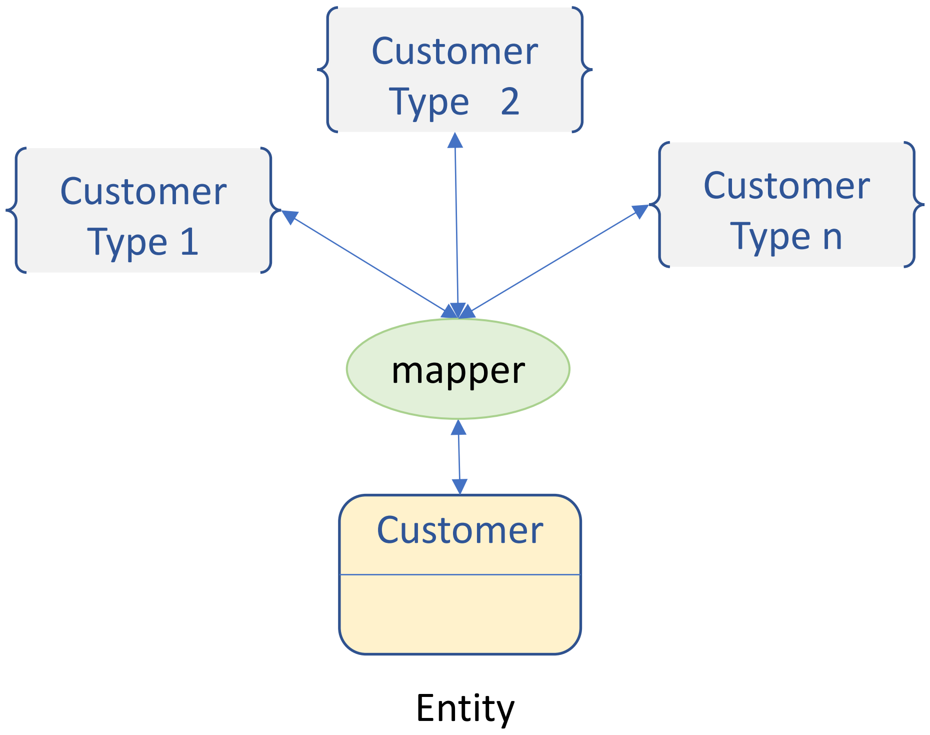

JSON
Presentation
Controller
Entity
Business Logic (**Service Class**)
Entity
Persistence (**Repository Class**)



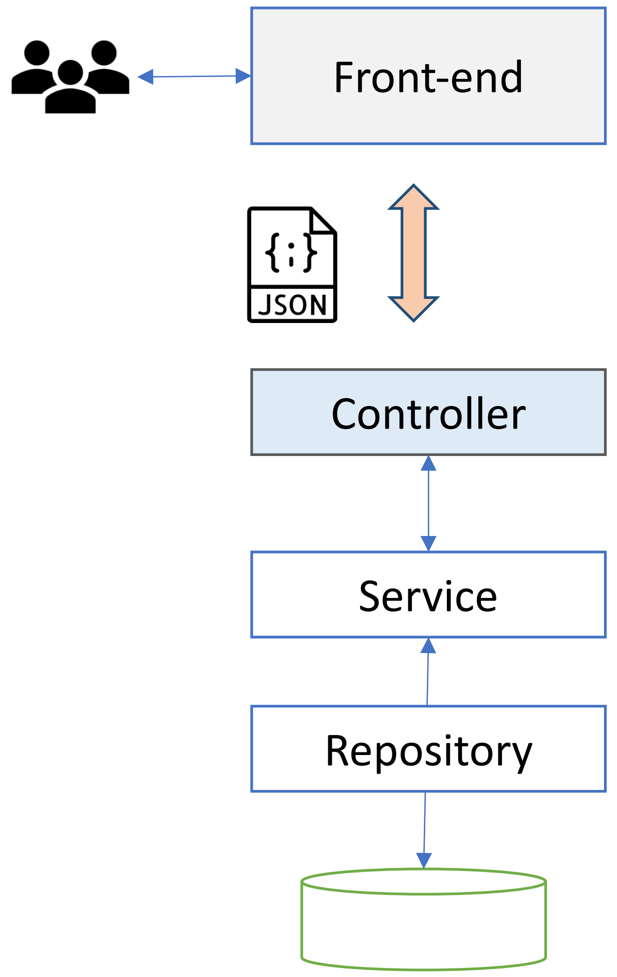
DTO: Data Transfer Object

- DTOs normally are created as POJOs.
- The data is mapped from the domain models to the DTOs.
- By using DTOs, we can provide as many different versions (with different structures) of our entities as we want.

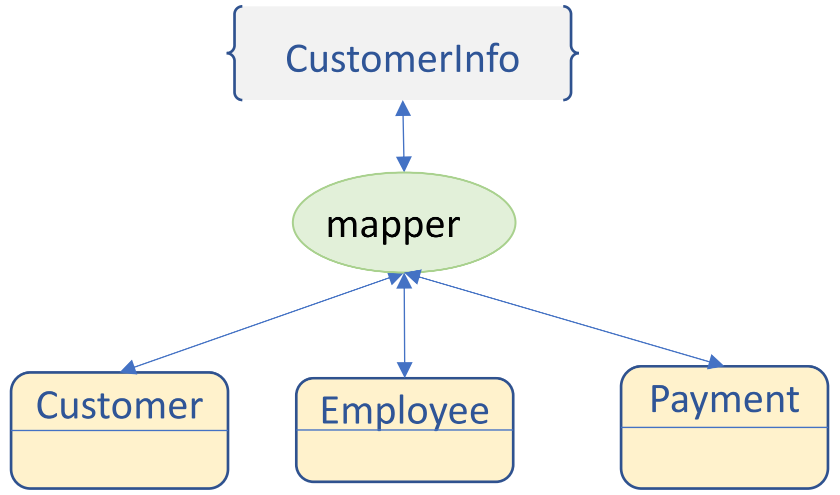




DTO: Data Transfer Object



DTO is a design pattern conceived to reduce the number of calls when working with remote interfaces.



Another advantage of using DTOs on RESTful APIs is that they can help hiding implementation details of domain objects (aka. entities). Exposing entities through endpoints can become a security issue if we do not carefully handle what properties can be changed through what operations.

Service Layer

```
@Service
public class CustomerService {
   @Autowired
   private CustomerRepository repository;
   public SimpleCustomerDTO getSimpleCustomerById(Integer id) {
      return repository.findById(id)
            .map(customer -> convertEntityToDto(customer))
            .orElseThrow(()->new ResponseStatusException(
                  HttpStatus.NOT_FOUND,  id +" Does Not Exist !!!" ));
   }
   private SimpleCustomerDTO convertEntityToDto(Customer customer) {
      SimpleCustomerDTO simpleCustomerDTO = new SimpleCustomerDTO();
      simpleCustomerDTO.setCustomerName(customer.getCustomerName());
        :
      simpleCustomerDTO.setSalesPerson(customer.getSalesRepEmployee().getFirstName()
                  + ' '+ customer.getSalesRepEmployee().getLastName());
      return simpleCustomerDTO;
   }
}
```

```
@Getter @Setter
public class SimpleCustomerDTO {
   private String customerName;
   private String phone;
   private String city;
   private String country;
   private String salesPerson;
}
```

```
@GetMapping("/{id}")
public SimpleCustomerDTO getCustomerById (@PathVariable Integer id) {
   return customerService.getSimpleCustomerById(id);
}
```

Model Mapper Library

- To **avoid** having to write **cumbersome/boilerplate code** to map DTOs into entities and vice-versa, we are going to use a library called **ModelMapper**.
- The goal of ModelMapper is to make object mapping easy by automatically determining how one object model maps to another.
- This library is quite powerful and accepts a whole bunch of configurations to streamline the mapping process, but it also favors convention over configuration by providing a default behavior that fits most cases.

```
<dependency>
   <groupId>org.modelmapper</groupId>
   <artifactId>modelmapper</artifactId>
   <version>3.1.1</version>
</dependency>
```

ModelMapper: How it works?

- ModelMapper consists of two separate processes
  - **The matching process**
    - Identifying eligible properties, transforming and tokenizing their names.
      - AccessLevels and NamingConventions (Type Mapping).
        - Methods are eligible based on configured
        - Eligible **methods take precedence over fields** with the same transformed property name.
        - **Only** source methods with **zero parameters** and **a non-void** return type are **eligible.**
  - The mapping process
    - Matched property values are converted from a source to destination object.
    - If a TypeMap exists for the source and destination types, mapping will occur according to the Mappings defined in the TypeMap.
    - Else if a Converter exists that is capable of converting the source object to the destination type, mapping will occur using the Converter.

Customer to SimpleCutomerDTO

```
@Entity
public class Customer {
   @Id
   private Integer id;
   private String customerName;
      :
      :
   private String postalCode;
   private String country;
   private BigDecimal creditLimit;
   ~~@JsonIgnore~~
   @ManyToOne(fetch = FetchType.EAGER)
   @JoinColumn(name = "salesRepEmployeeNumber")
   private Employee salesRepEmployee;
   @OneToMany(mappedBy = "customer")
   private Set<Payment> payments = new LinkedHashSet<>();
   @OneToMany(mappedBy = "customerNumber")
   private Set<Order> orders = new LinkedHashSet<>();
}
```

```
@Getter
@Setter
public class SimpleCustomerDTO {
   private String customerName;
   private String phone;
   private String city;
   private String country;
}
```

Eligible methods

```
@Entity
public class Customer {
    @Id
    private Integer id;
    private String customerName;
        :
        :
    private String postalCode;
    private String country;
    private BigDecimal creditLimit;
    ~~@JsonIgnore~~
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "salesRepEmployeeNumber")
    private Employee salesRepEmployee;
    @OneToMany(mappedBy = "customer")
    private Set<Payment> payments = new LinkedHashSet<>();
    @OneToMany(mappedBy = "customerNumber")
    private Set<Order> orders = new LinkedHashSet<>();
}
```

```
@Getter
@Setter
public class SimpleCustomerDTO {
    private String customerName;
    private String phone;
    private String city;
    private String country;

    public String getCountry() {
      return "Something";
    }

}
```

Using ModelMapper

```
modelMapper.map(entityObject, DTOClass.class);
```

```
@Autowired
private CustomerService service;
@GetMapping("/{id}")
public SimpleCustomerDTO getCustomerById (@PathVariable Integer id) {
   return service.getSimpleCustomerById(id);
}
```

```
@Service
public class CustomerService {
  @Autowired   private CustomerRepository repository;
  @Autowired   private ModelMapper modelMapper;
   public SimpleCustomerDTO getCustomer(int customerId) {
      Customer customer = repository.findById(customerId)
         .orElseThrow(()->new ResponseStatusException(
               HttpStatus.NOT_FOUND, customerId+ " does not exist !!!"));
      return modelMapper.map(customer, SimpleCustomerDTO.class);
   }
```

Deep/Nested Mappings

```java
@Setter @Getter
public class SimpleEmployeeDTO {
    private String lastName;
    private String firstName;
}
```

```java
public class SimpleCustomerDTO {
    private String customerName;
    private String phone;
    private String city;
    private String country;
    private SimpleEmployeeDTO salesRepEmployee;
}
```

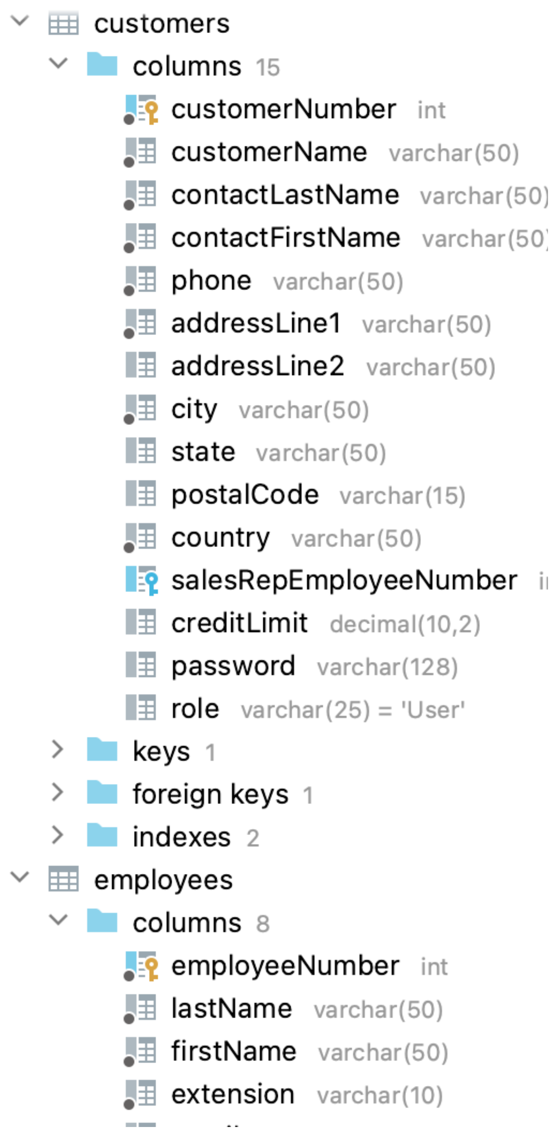
```json
{
    "customerName": "Atelier graphique",
    "phone": "40.32.2555",
    "city": "Nantes",
    "country": "France",
    "salesRepEmployee": {
        "lastName": "Hernandez",
        "firstName": "Gerard"
    }
}
```

Deep Mappings (2)

```java
@Setter @Getter
public class SimpleEmployeeDTO {
    private String lastName;
    private String firstName;
}
```

```json
{
    "customerName": "Atelier graphique",
    "phone": "40.32.2555",
    "city": "Nantes",
    "country": "France",
    "salesRepEmployeeFirstName": "Gerard",
    "salesRepEmployeeLastName": "Hernandez"
}
```

```java
public class SimpleCustomerDTO {
    private String customerName;
    private String phone;
    private String city;
    private String country;
    private String salesRepEmployeeFirstName;
    private String salesRepEmployeeLastName;
}
```



```
customers
  columns 15
    customerNumber  int
    customerName  varchar(50)
    contactLastName  varchar(50)
    contactFirstName  varchar(50)
    phone  varchar(50)
    addressLine1  varchar(50)
    addressLine2  varchar(50)
    city  varchar(50)
    state  varchar(50)
    postalCode  varchar(15)
    country  varchar(50)
    salesRepEmployeeNumber  i
    creditLimit  decimal(10,2)
    password  varchar(128)
    role  varchar(25) = 'User'
  keys 1
  foreign keys 1
  indexes 2
employees
  columns 8
    employeeNumber  int
    lastName  varchar(50)
    firstName  varchar(50)
    extension  varchar(10)
```

Deep Mappings (3)

```java
@Setter @Getter
public class SimpleEmployeeDTO {
   private String lastName;
   private String firstName;
   public String getName() {
      return firstName + " "+ lastName;
   }
}
```

```java
public class SimpleCustomerDTO {
   private String customerName;
   private String phone;
   private String city;
   private String country;
  @JsonIgnore
   private SimpleCustomerDTO salesRepEmployee;
   public String getSalesPerson()  {
      return salesRepEmployee==null ? "-": salesRepEmployee.getName();
   }
}
```

```
{
    "customerName": "Atelier graphique",
    "phone": "40.32.2555",
    "city": "Nantes",
    "country": "France",
    "salesPerson": "Gerard Hernandez"
}
```

Exercise

(1) Create Customer DTO

```
package sit.int204.classicmodelsservice.dtos

@Getter @Setter
public class SimpleCustomerDTO {
    private String customerName;
    private String phone;
    private String city;
    private String country;
    private String salesPerson;
}
```

Setup Model Mapper

- Add Dependency to Maven

```
<dependency>
   <groupId>org.modelmapper</groupId>
   <artifactId>modelmapper</artifactId>
~~   <version>3.1.1</version>~~     3.2.0
</dependency>
```

- Defined Bean for ModelMapper (in base package)

```
@Configuration
public class ApplicationConfig {
   @Bean
   public ModelMapper modelMapper() {
      return new ModelMapper();
   }
}
```

Using ModelMapper instead custom mapper

```
@Service
public class CustomerService {
   @Autowired
   private CustomerRepository repository;
   public Customer getCustomerById(Integer customerId) {
      return repository.findById(customerId).orElseThrow(()->new ResponseStatusException(
            HttpStatus.NOT_FOUND, "Customer id "+ customerId+ "Does Not Exist !!!"));
   }
}
```

```
{
      "customerName": "Blauer See Auto, Co.",
      "phone": "+49 69 66 90 2555",
      "city": "Frankfurt",
      "country": "Germany",
      "salesPerson": null
}
```

```
@RestController
public class CustomerController {
  @Autowired  private CustomerService service;
  @Autowired  private ModelMapper modelMapper;
   @GetMapping(“/{customerId}”)
   public SimpleCustomerDTO getSimpleCustomerById(@PathVariable Integer customerId) {
       return modelMapper.map(service.getCustomerById(customerId), SimpleCustomer.class);
   }
}
```

Deep Mapping (Testing for each DTO)

1

```
@Setter
@Getter
public class SimpleEmployeeDTO {
    private String lastName;
    private String firstName;
}
```

2

```
@Setter
@Getter
public class SimpleEmployeeDTO {
    private String lastName;
    private String firstName;
    public String getName() {
        return firstName + " "+ lastName;
    }
}
```

3

```
@Setter
public class SimpleEmployeeDTO {
    private String lastName;
    private String firstName;
    public String getName() {
        return firstName + " "+ lastName;
    }
}
```

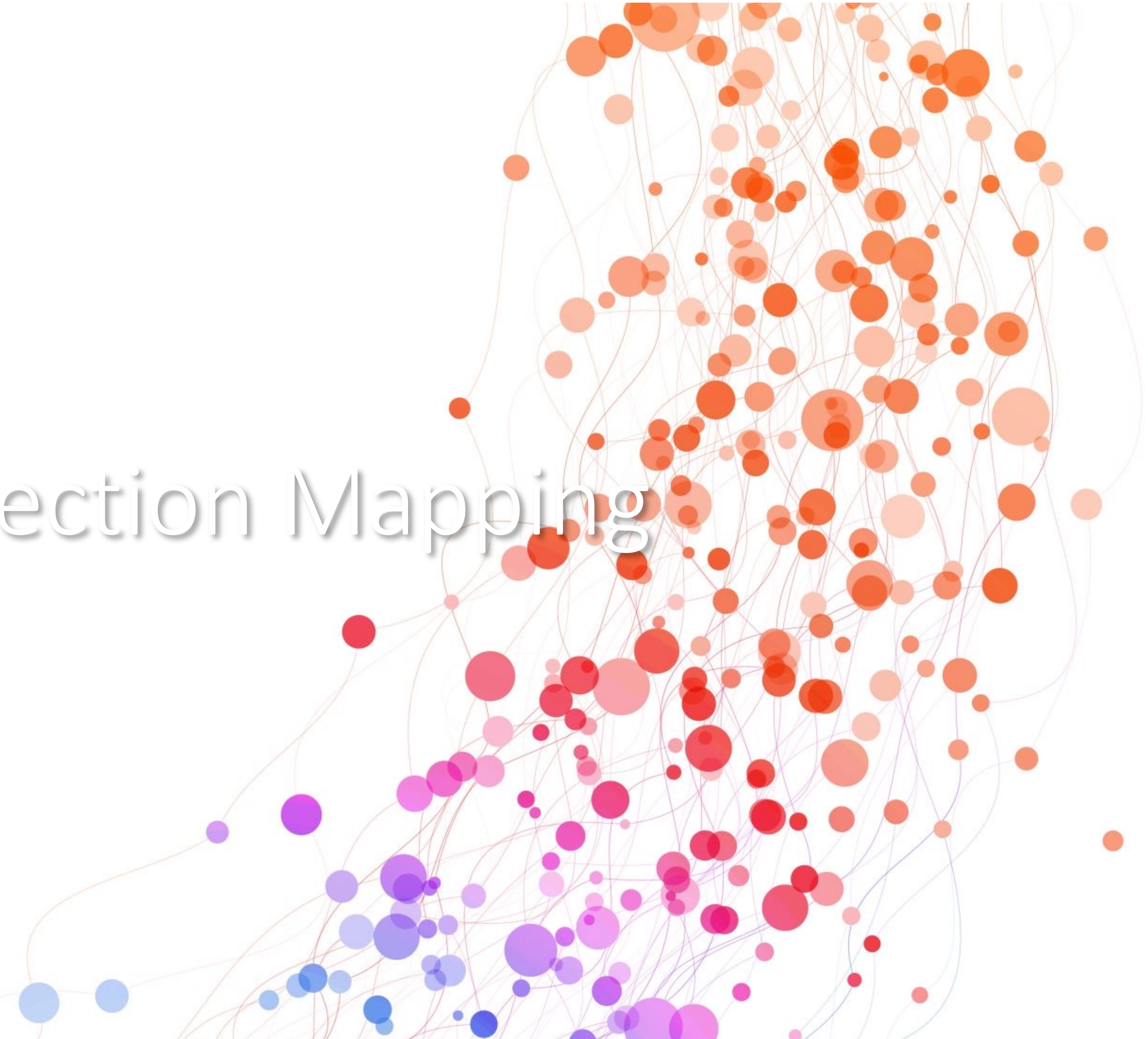
```
@Getter @Setter
public class SimpleCustomerDTO {
    private String customerName;
    :
    private SimpleEmployeeDTO salesRepEmployee;
}
```

Modify SimpleCustomerDTO

```
@Getter @Setter
public class SimpleCustomerDTO {
   private String customerName;
   :
   private String salesRepEmployeeFirstName;
   private String salesRepEmployeeLastName;
}
```

```
@Getter @Setter
public class SimpleCustomerDTO {
   private String customerName;
   :
  @JsonIgnore
  private SimpleEmployeeDTO sales;
  public String getSalesPerson() {
      return sales == null ? "-" : sales.getName();
  }
}
```

Collection Mapping



Mapping Lists with ModelMapper

```
List<EmployeeDTO> dtos = employees .stream() .map(employee ->
modelMapper.map(employee, EmployeeDTO.class)) .collect(Collectors.toList());
```

```
@GetMapping("")
public List<EmployeeDTO> getEmployees() {
    List<Employee> employeeList = repository.findAll();
     return employeeList.stream()
            .map(e -> modelMapper.map( e, EmployeeDTO.class))
            .collect(Collectors.toList());
}
```

General-purpose parameterized method

```
   @GetMapping("")
   public List<EmployeeDTO> getEmployees() {
      List<Employee> employeeList = repository.findAll();
       return return mapList(employeeList, EmployeeOfficeDTO.class);
  }
```

```
public static <S, T> List<T> mapList(List<S> source, Class<T> targetClass) {
   return source.stream()
         .map(entity -> modelMapper.map(entity, targetClass))
         .collect(Collectors.toList());
}
```

Convert DTO to Entity

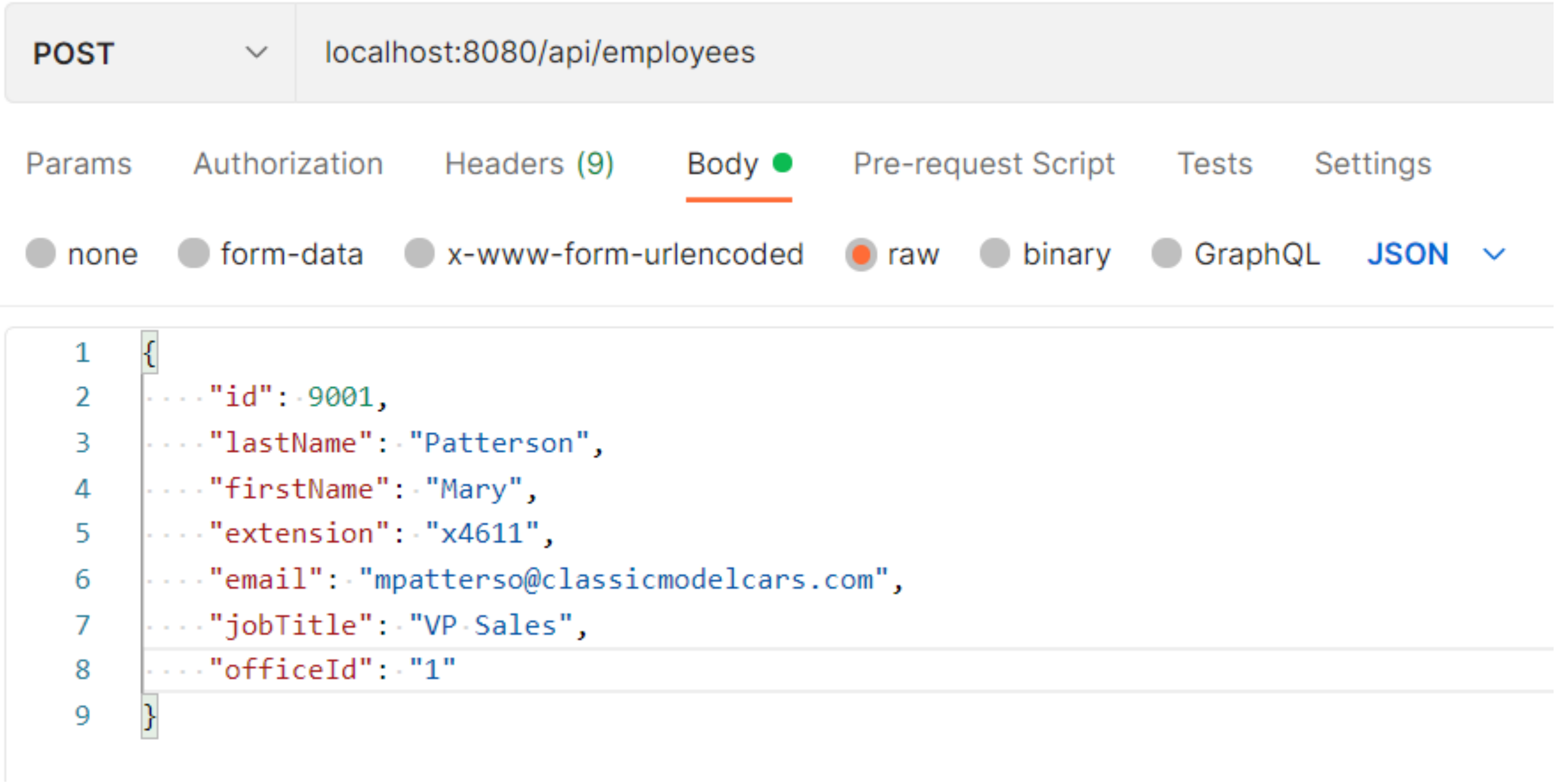
```java
@Getter   @Setter
@NoArgsConstructor
@AllArgsConstructor
public class EmployeeDTO {
    private Integer id;
    private String lastName;
    private String firstName;
    private String extension;
    private String email;
    private String jobTitle;
    private String officeId;
}
```

```java
@Entity
@Table(name = "employees")
public class Employee {
    @Id
    @Column(name = "employeeNumber", nullable = false)
    private Integer id;

    @Column(name = "lastName", nullable = false, length = 50)
    private String lastName;
```

```java
@PostMapping("")
public Employee create(@RequestBody EmployeeDTO newEmployee) {
    Employee employee = modelMapper.map(newEmployee, Employee.class);
    return repository.saveAndFlush(employee);
}
```

Request Example


POST localhost:8080/api/employees

```json
{
    "id": 9001,
    "lastName": "Patterson",
    "firstName": "Mary",
    "extension": "x4611",
    "email": "mpatterso@classicmodelcars.com",
    "jobTitle": "VP Sales",
    "officeId": "1"
}
```

Exercise

Mapping Lists with ModelMapper

```
List<EmployeeDTO> dtos = employees .stream() .map(employee ->
modelMapper.map(employee, EmployeeDTO.class))
.collect(Collectors.toList());
```

```
@GetMapping("")
public List<EmployeeDTO> getEmployees() {
    List<Employee> employeeList = repository.findAll();
     return employeeList.stream()
            .map(e -> modelMapper.map( e, EmployeeDTO.class))
            .collect(Collectors.toList());
}
```

General-purpose parameterized method

```
  @GetMapping("")
  public List<EmployeeDTO> getEmployees() {
    List<Employee> employeeList = repository.findAll();
     return return mapList(employeeList, EmployeeOfficeDTO.class, modelMapper);
 }
```

```
public static <S, T> List<T> mapList(List<S> source, Class<T> targetClass, ModelMapper modelMapper) {
   return source.stream()
          .map(entity -> modelMapper.map(entity, targetClass))
          .collect(Collectors.toList());
}
```

Generic PageDTO Example

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class PageDTO<T> {
    private List<T> content;
    private Boolean last;
    private Boolean first;
    private Integer totalPages;
    private Integer totalElements;
    private Integer size;
    @JsonIgnore
    private Integer number;
    public Integer getPage() {
        return number;
    }
}
```

```
{
  "content": [
    {
      "productCode": "S10_1678",
      "productName": "1969 Harley Davidson Ultimate Chopper",
      "productLine": "Motorcycles",
      "productScale": "1:10",
       "price": 95.7
    },
    :
    :
    {
      "productCode": "S10_1949",
      "productName": "1952 Alpine Renault 1300",
      "productLine": "Classic Cars",
      "productScale": "1:10",
      "price": 214.3
    }
  ],
  "first": true,
  "totalPages": 12,
  "totalElements": 111,
  "size": 10,
  "page": 0
}
```

Create Singleton ListMapper Service

package sit.int204.classicmodelsservice.utils

```
public class ListMapper {
   private static final ListMapper listMapper = new ListMapper();
   private ListMapper() { }
   public <S, T> List<T> mapList(List<S> source,  Class<T> targetClass, ModelMapper modelMapper) {
      return source.stream().map(entity -> modelMapper.map(entity, targetClass))
            .collect(Collectors.toList());
   }
   public static ListMapper getInstance() {
      return listMapper;
   }
   public <S, T> PageDTO<T> toPageDTO(Page<S> source, Class<T> targetClass,
                                   ModelMapper modelMapper) {
      PageDTO<T> page = modelMapper.map(source, PageDTO.class);
      page.setContent(mapList(source.getContent(), targetClass, modelMapper));
      return page;
   }
}
```

Create EmployeeDTO & Modify Application config

```
@Getter   @Setter
@NoArgsConstructor
@AllArgsConstructor
public class EmployeeDTO {
   private Integer id;
   private String lastName;
   private String firstName;
   private String extension;
   private String email;
   private String jobTitle;
   private String officeId;
}
```

```
@Configuration
public class ApplicationConfig {
  :
  @Bean
   public ListMapper listMapper() {
      return ListMapper.getInstance();
   }
}
```

```
@Entity
@Table(name = "offices")
public class Office {
   @Id
   @Column(name = "officeCode")
   private String id;
   @Column(name = "city", nullable = false, length = 50)
```

Create Employee Controller & Service

```
@Service
public class EmployeeService {
   @Autowired    private  EmployeeRepository repository;
   public Employee save(Employee employee) {
       return repository.saveAndFlush(employee);
   }
}
```

```
@Autowired     private ModelMapper modelMapper;
@Autowired     private ListMapper listMapper;
  :
@PostMapping("")
public Employee create(@RequestBody EmployeeDTO newEmployee) {
   Employee e = modelMapper.map(newEmployee, Employee.class);
   return employeeService.save (e);
}
```