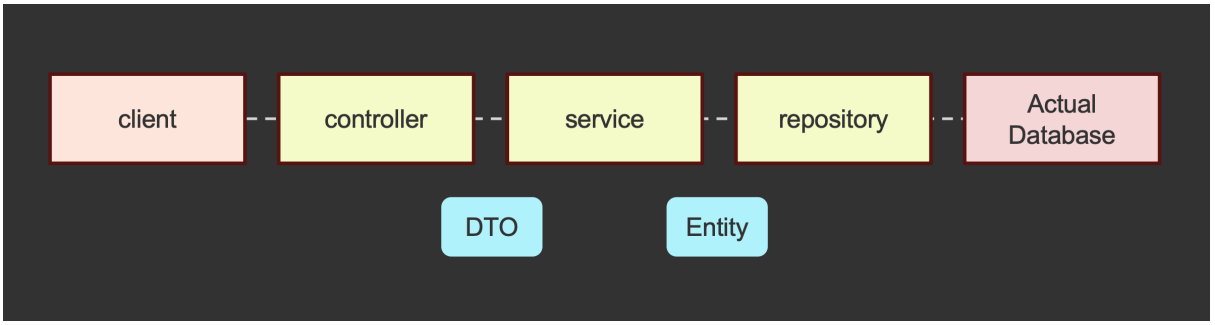
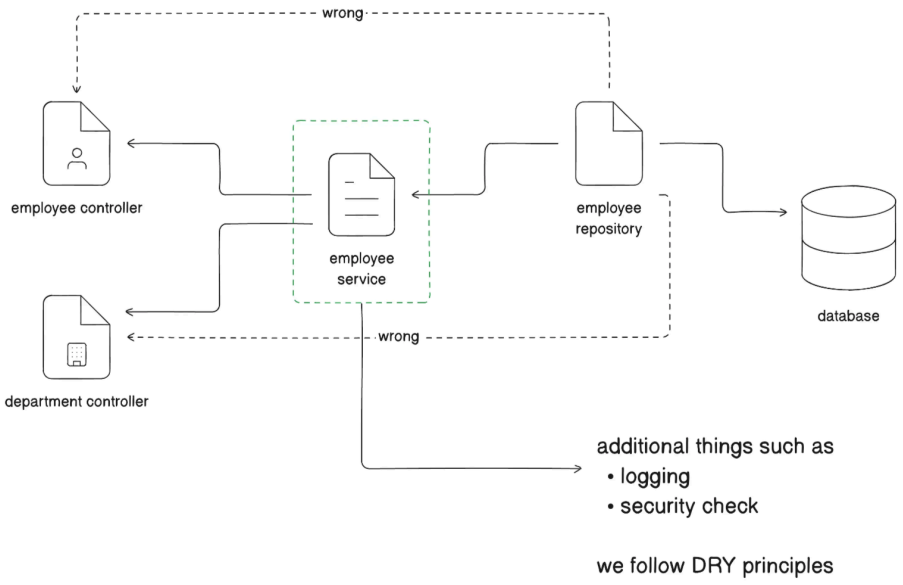
Service Layer

Spring Boot Web Project Structure



Key roles of Service Layer

- The service layer acts as a bridge between the persistence layer (responsible for data access) and the presentation layer (handling user interaction)
- It encapsulates the business logic of the application, orchestrates interactions between different components, and provides a clean interface for external clients to interact with the system
- By abstracting away the complexities of data access and business operations, the service layer promotes modularity, maintainability, and scalability.



Earlier if you recall, we have used Repository directly in the controller which is a wrong practice and doesn't follow the DRY principles as shown in the above image. We loose the benefit of reusing the functions in multiple controllers and also loose on the opportunity to add additional things such as logging, security check, etc.

To move ahead, we are extracting out that part involving into a separation of concern called as service layer which deals with it and acts as the intermediary b/w controller and repository layer.

That should lead us to modify our controller in the following fashion,

```
@RestController
@RequestMapping("/employees")
public class EmployeeController {

    private final EmployeeService employeeService;

    public EmployeeController(EmployeeService employeeService) {
        this.employeeService = employeeService;
    }

    @GetMapping(path = "/{employeeId}")
    public EmployeeDTO getEmployeeById(@PathVariable Long employeeId)
```

```
{
        return employeeService.getEmployeeById(employeeId);
    }

    @GetMapping
    public List<EmployeeDTO> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

    @PostMapping
    public EmployeeDTO createNewEmployee(@RequestBody EmployeeDTO
inputEmployee) {
        return employeeService.createNewEmployee(inputEmployee);
    }
}
```

In the above code if you observe we have simplified the controller layer logic and passing that responsibility of dealing with repository layer into a different service file of Employee. And the return type is NOT an entity rather a POJO class object which is more simplified and adhere to what we only want to expose to the outside world in response format.

At the service layer now all the magic takes place and would have a logic such as,

```
@Service
public class EmployeeService {

    private final EmployeeRepository employeeRepository;
    private final ModelMapper modelMapper;

    public EmployeeService(
        EmployeeRepository employeeRepository,
        ModelMapper modelMapper
      ) {
        this.employeeRepository = employeeRepository;
        this.modelMapper = modelMapper;
    }
```

```
    public EmployeeDTO getEmployeeById(Long employeeId) {
        EmployeeEntity employeeEntity = employeeRepository
                                                .findById(employeeId)
                                                 .orElse(null);

        return modelMapper.map(employeeEntity, EmployeeDTO.class);
    }

    public List<EmployeeDTO> getAllEmployees() {
        List<EmployeeEntity> employeeEntities = employeeRepository.findAll
();

        return employeeEntities
                .stream()
                .map(employeeEntity -> modelMapper.map(employeeEntity, Empl
oyeeDTO.class))
                .collect(Collectors.toList());
    }

    public EmployeeDTO createNewEmployee(EmployeeDTO inputEmployee)
{
        EmployeeEntity toSaveEntity = modelMapper.map(inputEmployee, Em
ployeeEntity.class);

        EmployeeEntity savedEmployeeEntity = employeeRepository.save(toS
aveEntity);

        return modelMapper.map(savedEmployeeEntity, EmployeeDTO.class);
    }
}
```

Wait, in this above code did you observe something new? `MapperConfig` is added: this is acting as the layer that will auto-translate for you and map the fields one-by-one and take only what you want to send as part of DTO and avoid giving too much info we get from Entity class.

We configure it in the following way, by adding the dependency

```
<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>3.2.6</version>
</dependency>
```

```
@Configuration
public class MapperConfig {

    @Bean
    public ModelMapper getModelMapper() {
        return new ModelMapper();
    }

}
```