Workshop - Todo API

A Spring Boot RESTFul Api.

The purpose of the API is to manage tasks to be done.
This might be used at a company for keeping track of a projects flow. Making sure that the project is on schedule and workers know their task to be done.

Already implemented:

- Entity with Relationships
- Repositories

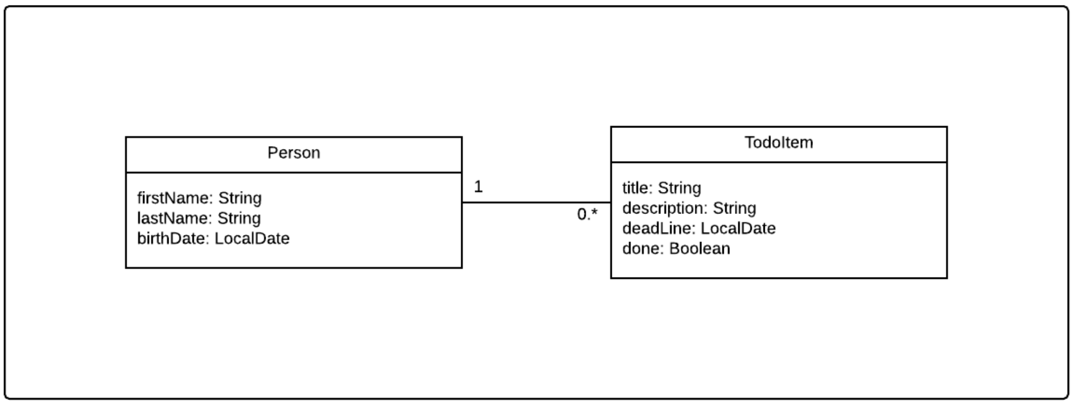
Topics

- RESTful Controller
- Service Pattern
- DTO Pattern
- Exception Handling
- Validation

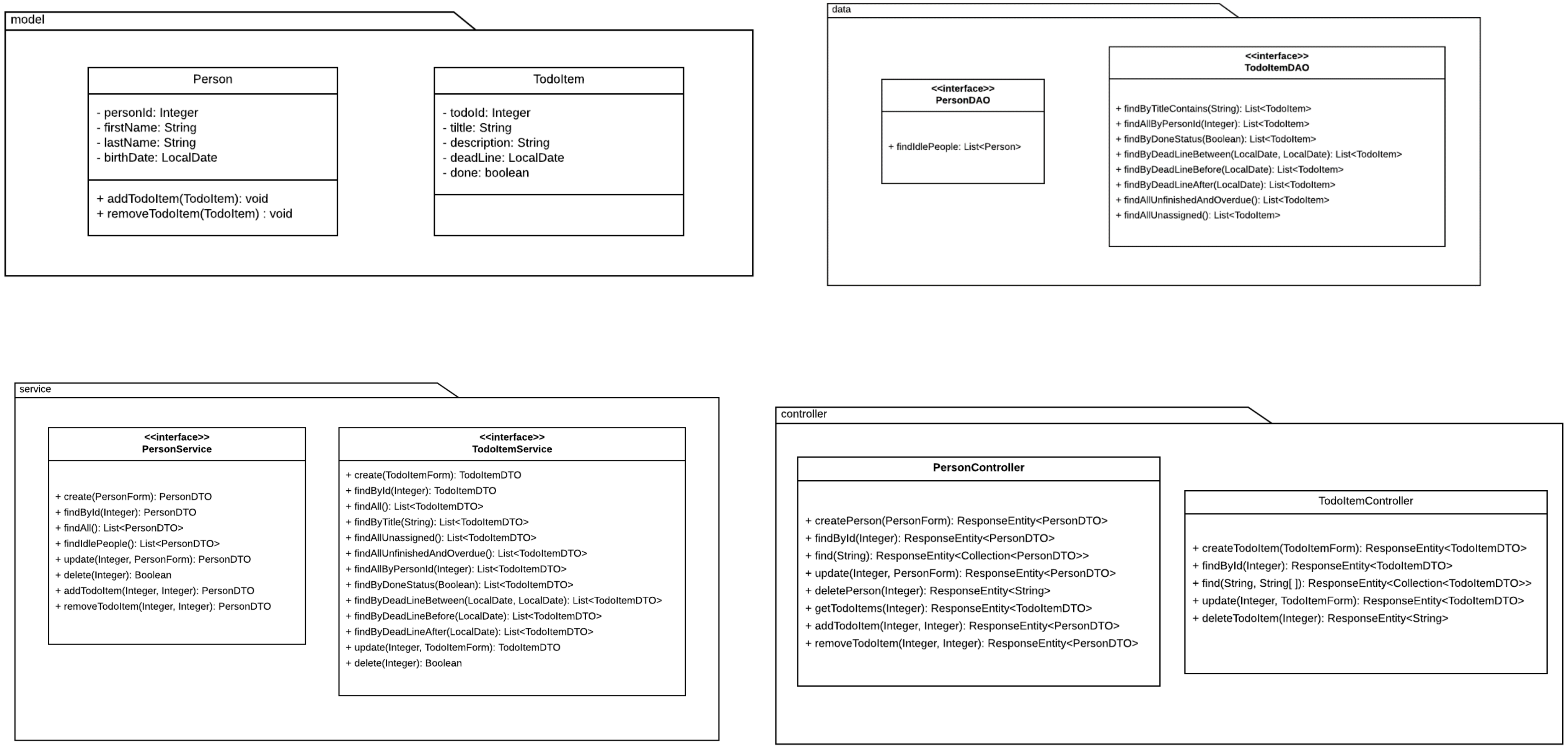
Task

Your task is to make a functional Task list API.
At the end you should be able to administrate tasks. Creating, changing, assigning and different ways to find tasks.

Domain Model



Diagram


model
Person
- personId: Integer
- firstName: String
- lastName: String
- birthDate: LocalDate
+ addTodoItem(TodoItem): void
+ removeTodoItem(TodoItem) : void
TodoItem
- todoId: Integer
- tiltle: String
- description: String
- deadLine: LocalDate
- done: boolean
data
<<interface>>
PersonDAO
+ findIdlePeople: List<Person>
<<interface>>
TodoItemDAO
+ findByTitleContains(String): List<TodoItem>
+ findAllByPersonId(Integer): List<TodoItem>
+ findByDoneStatus(Boolean): List<TodoItem>
+ findByDeadLineBetween(LocalDate, LocalDate): List<TodoItem>
+ findByDeadLineBefore(LocalDate): List<TodoItem>
+ findByDeadLineAfter(LocalDate): List<TodoItem>
+ findAllUnfinishedAndOverdue(): List<TodoItem>
+ findAllUnassigned(): List<TodoItem>
service
<<interface>>
PersonService
+ create(PersonForm): PersonDTO
+ findById(Integer): PersonDTO
+ findAll(): List<PersonDTO>
+ findIdlePeople(): List<PersonDTO>
+ update(Integer, PersonForm): PersonDTO
+ delete(Integer): Boolean
+ addTodoItem(Integer, Integer): PersonDTO
+ removeTodoItem(Integer, Integer): PersonDTO
<<interface>>
TodoItemService
+ create(TodoItemForm): TodoItemDTO
+ findById(Integer): TodoItemDTO
+ findAll(): List<TodoItemDTO>
+ findByTitle(String): List<TodoItemDTO>
+ findAllUnassigned(): List<TodoItemDTO>
+ findAllUnfinishedAndOverdue(): List<TodoItemDTO>
+ findAllByPersonId(Integer): List<TodoItemDTO>
+ findByDoneStatus(Boolean): List<TodoItemDTO>
+ findByDeadLineBetween(LocalDate, LocalDate): List<TodoItemDTO>
+ findByDeadLineBefore(LocalDate): List<TodoItemDTO>
+ findByDeadLineAfter(LocalDate): List<TodoItemDTO>
+ update(Integer, TodoItemForm): TodoItemDTO
+ delete(Integer): Boolean
controller
PersonController
+ createPerson(PersonForm): ResponseEntity<PersonDTO>
+ findById(Integer): ResponseEntity<PersonDTO>
+ find(String): ResponseEntity<Collection<PersonDTO>>
+ update(Integer, PersonForm): ResponseEntity<PersonDTO>
+ deletePerson(Integer): ResponseEntity<String>
+ getTodoItems(Integer): ResponseEntity<TodoItemDTO>
+ addTodoItem(Integer, Integer): ResponseEntity<PersonDTO>
+ removeTodoItem(Integer, Integer): ResponseEntity<PersonDTO>
TodoItemController
+ createTodoItem(TodoItemForm): ResponseEntity<TodoItemDTO>
+ findById(Integer): ResponseEntity<TodoItemDTO>
+ find(String, String[ ]): ResponseEntity<Collection<TodoItemDTO>>
+ update(Integer, TodoItemForm): ResponseEntity<TodoItemDTO>
+ deleteTodoItem(Integer): ResponseEntity<String>

Part 1

Getting started.

a. Fork the project from given Repository link. This makes sure you have your own copy of the project and also have a connection to the original repo.
b. Setup your environment, properties and database connection.
c. Run the application and make sure it compiles and the database is created without major errors.

Part 2

DTO’s and Conversion Layer.

It’s time to implement some DTO’s to give the client meaningful objects to work with.
Alongside keeping the integrity of the entities.
You are going to add Dto’s for viewing and creating entities.

Create data transfer objects for updating and creating People and TodoItems.

1. Create a class called `PersonForm` in a package called `form`.
    a. Mirror the field types and names from Person class except Id and Task list.
    b. Make methods to ensure the fields created can be accessed outside the class.

2. Create a class called TodoItemForm in `form` package.
    a. Mirror the field types and names from TodoItem class except Id and assigned person.
    b. Make methods to ensure the fields created can be accessed outside the class.

Create data transfer objects for the client to view our entities in a meaningful way.
HEADSUP! There is a relationship which most like will give you recursive calls, make sure to manage it.

3. Create a class called PersonDto in `dto` package.
    a. Mirror the fields from Person class.
    b. Make methods to ensure the fields created can be accessed outside the class.

4. Create a class called TodoItemDto in `dto` package.
    a. Mirror the fields from TodoItem class.
    b. Make methods to ensure the fields created can be accessed outside the class.

When using DTO’s you often need to convert in different ways. Managing conversions can be done with frameworks or by yourself.

5. Create a service class to convert objects in a meaningful way.
    a. Create methods for converting Form’s to the entity equivalent.
    b. Create methods for converting entity to Dto.
    Ex. If you have a CarForm convert to Car entity, or Car entity to CarDto.
    c. Make sure the methods created work as intended.

Part 3

Entity Service Layer

The Service layer is managing the business logic in your application.
This layer helps controller and data access object to focus on what they do best. (SoC / SRP)
(Presentation layer for client requests | Data access Layer for managing the database calls.)

1. Implement the service layer.
    a. Create interfaces regarding the diagram.
    b. Implement the interfaces. Make use of the data access objects and the conversion service created earlier.

Part 4

Controller Layer

The Controller is the presentational layer or the end point of you Java program.
The client will use these to interact with the application.

1. Implement the controller layer.
    a. Create interfaces regarding the diagram.
    b. Implement the interfaces. Make use of the service layer created earlier.
    Build the controller paths with this as base: *`/todo/api/v1/`*
    `Use Case Ex. Localhost:8080/todo/api/v1/person/2` to request person with id 2

    c. Make sure your method works as intended.

(Extra) Manage the Exceptions thrown in the program, do it in a meaningful way for the client.

    d. Create a controller advice class.

(Extra) Add validation for the forms.