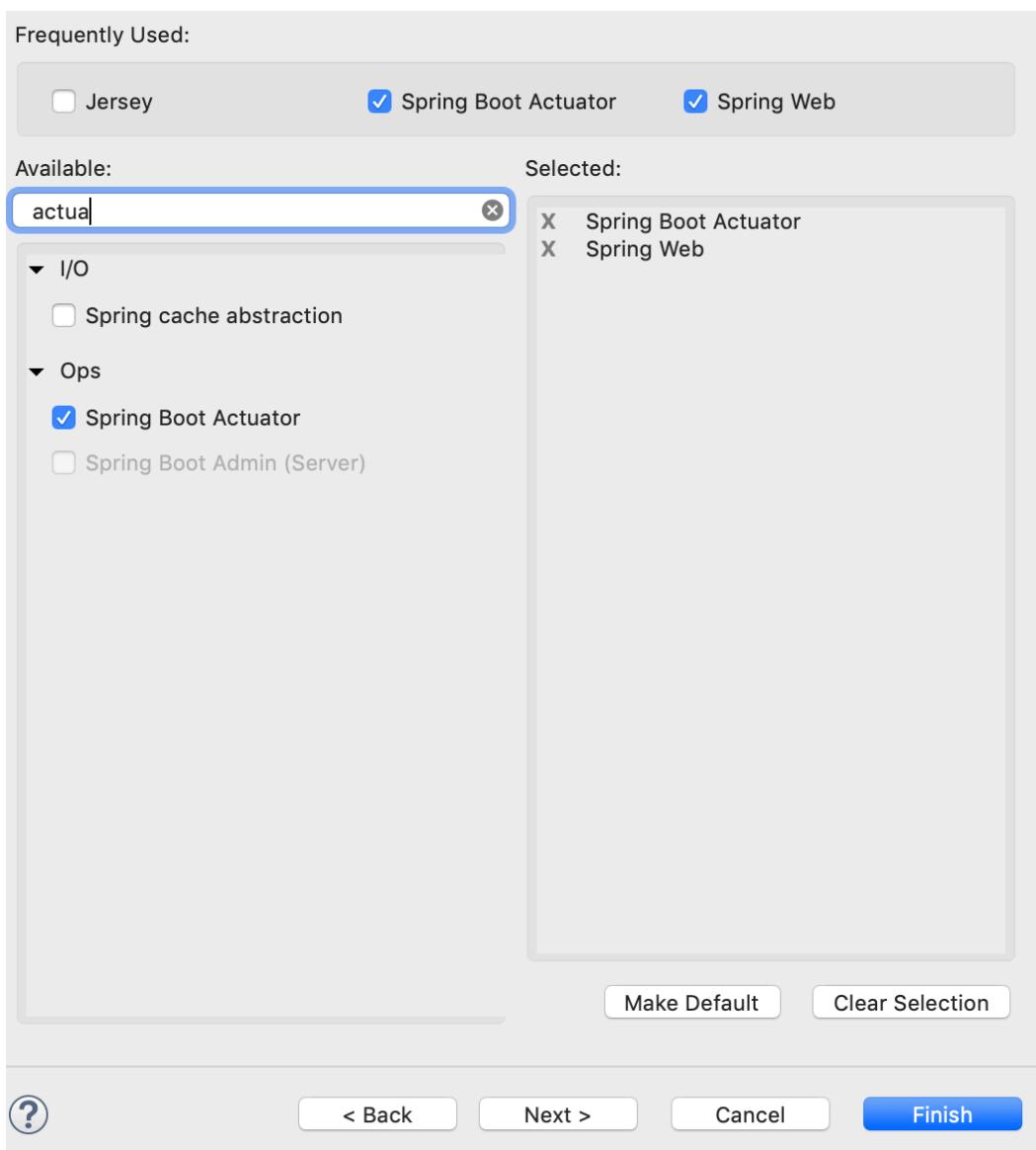
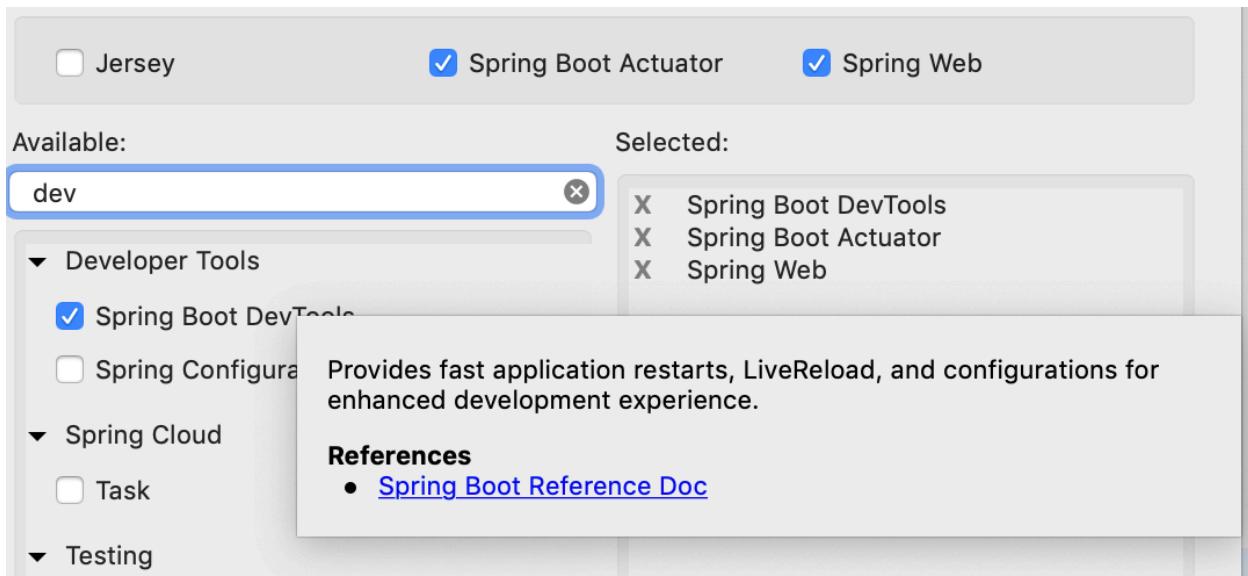


# Create a ToDo REST API with CRUDS Operations

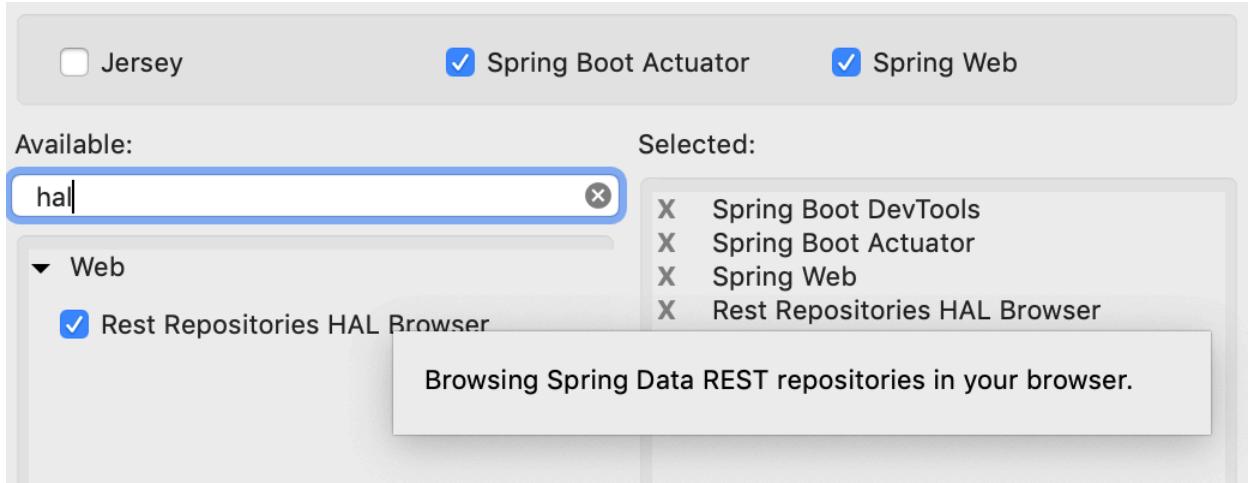
1. File → New → Spring Starter Project
2. Name = todo-cruds
3. Group = com.todo
4. Package = com.demo
5. Click on Next
6. Select “Spring Web”, “Actuator”, “DevTools” & “HAL Explorer” module



## 6.1



## 6.2



7. Click on Finish

8. Open pom.xml and change dependency for HAL Browser as “explorer”:

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-rest-hal-explorer</artifactId>
</dependency>
```

9. Create a “Todo” Model Object

```
import java.util.Date;

public class Todo {
    private int id;
    private String user;

    private String desc;

    private Date targetDate;
    private boolean isDone;

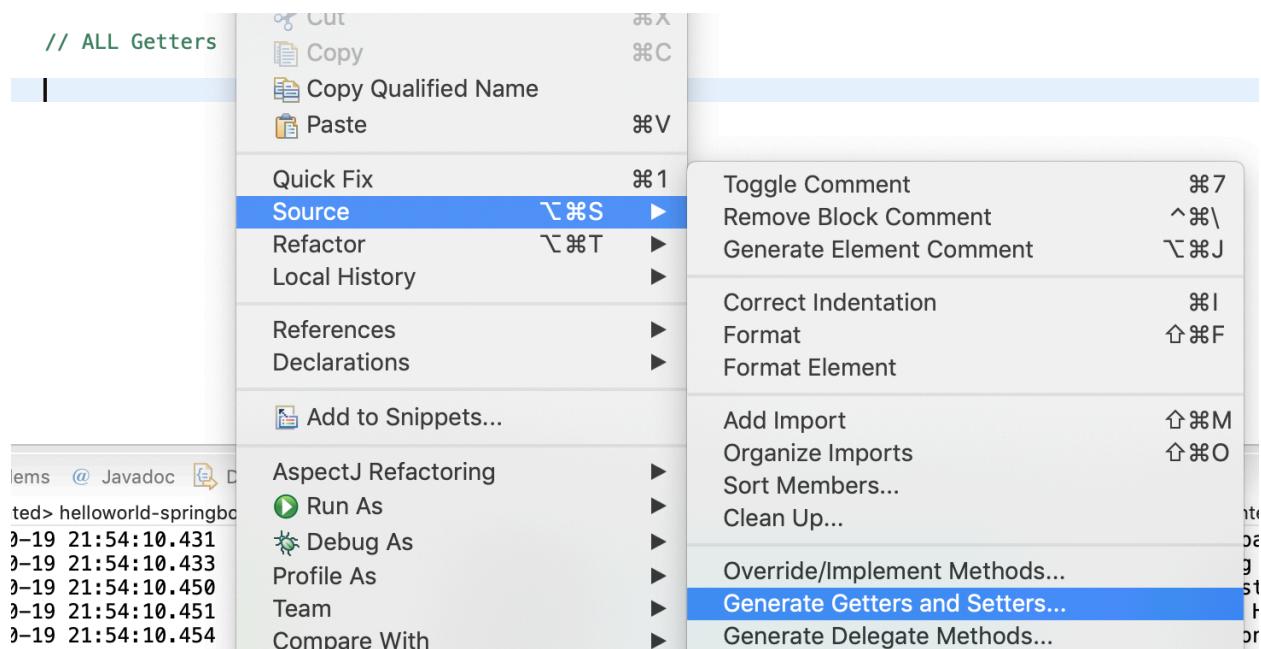
    public Todo() {
    }

    public Todo(int id, String user, String desc, Date targetDate, boolean isDone) {
        super();
        this.id = id;
        this.user = user;
        this.desc = desc;
        this.targetDate = targetDate;
        this.isDone = isDone;
    }

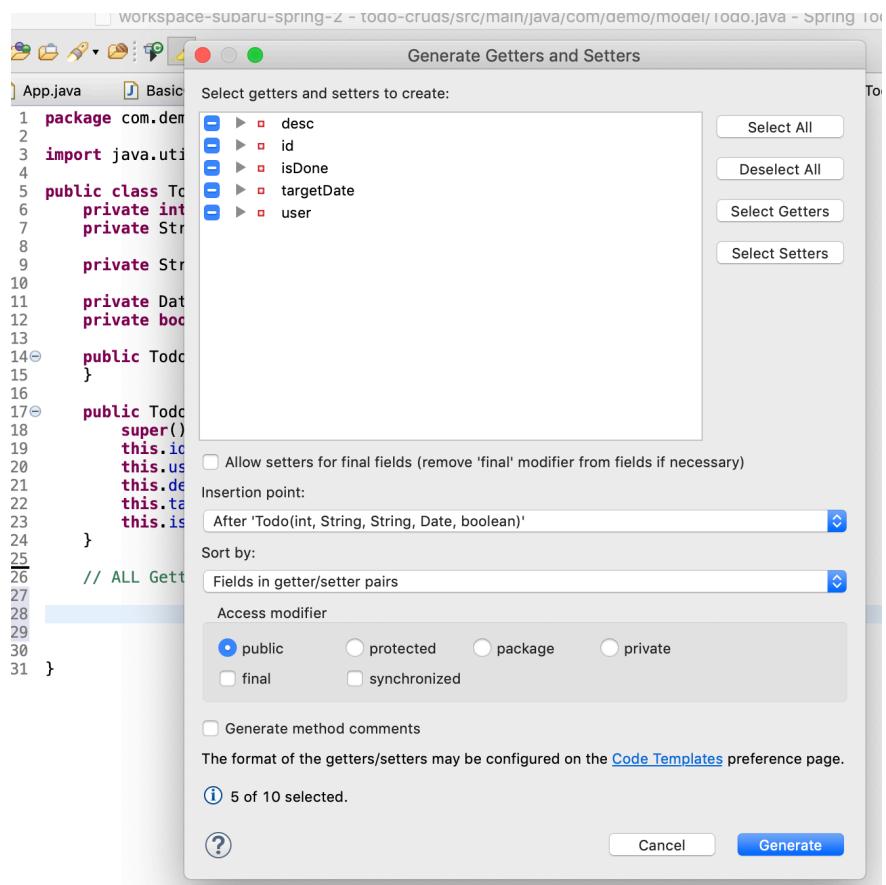
    // ALL Getters

}
```

10. Generate Getter methods ONLY for all the properties as shown here:



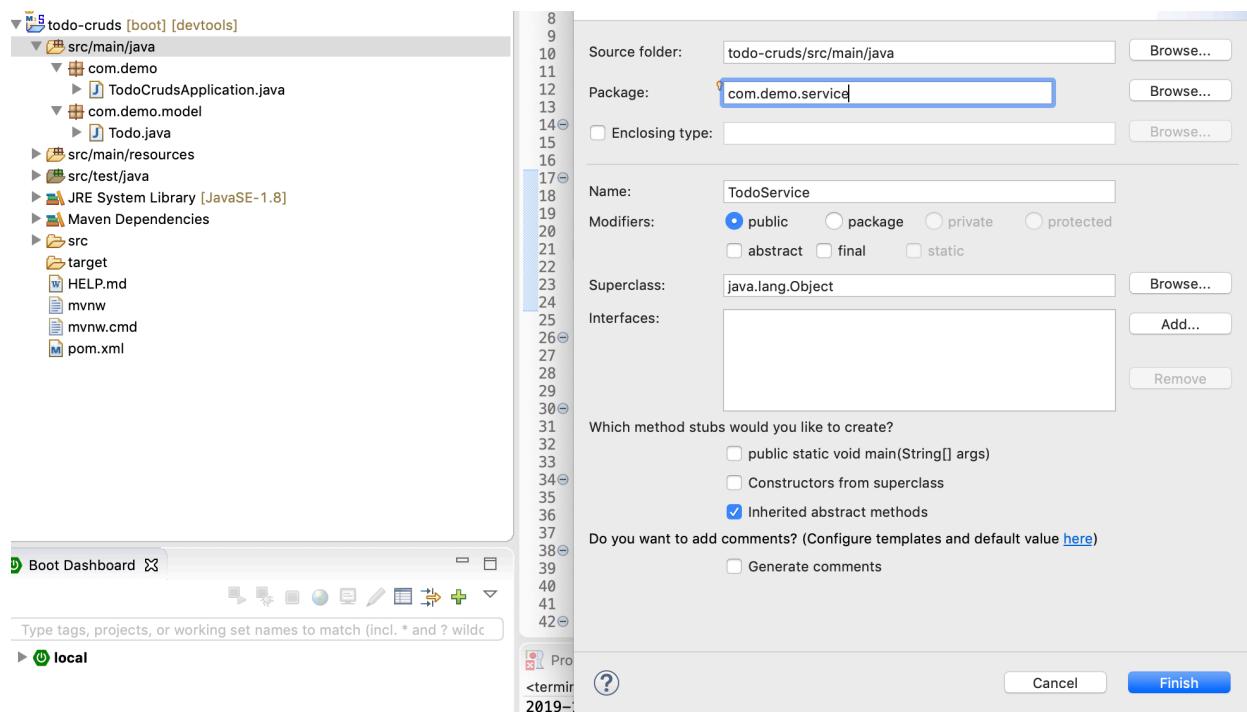
### 10.1 Click on “Select Getters” Button



## 10.2 Click on Generate button

### 11. Generate a Service class to handle all the data operations

So, lets create a TodoService.java



### 12. TodoService deals with in-memory data and performs all the operations which are connected to appropriate API resource.

So, lets add the in-memory data here:

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import org.springframework.stereotype.Service;

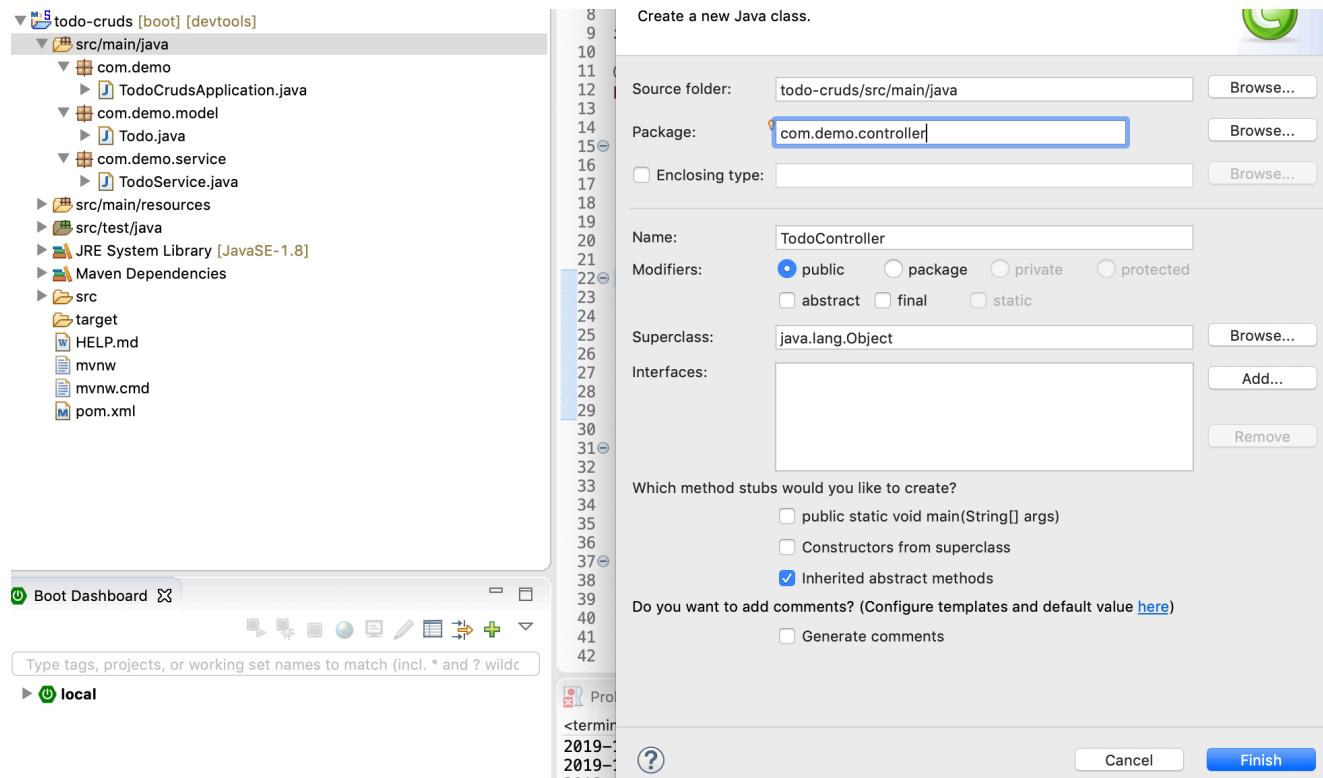
import com.demo.model.Todo;

@Service
public class TodoService {
    private static List<Todo> todos = new ArrayList<Todo>();
    private static int todoCount = 4;
    static {
        todos.add(new Todo(1, "Jack", "Learn Spring MVC", new Date(), false));
        todos.add(new Todo(2, "Jack", "Learn Struts", new Date(), false));
        todos.add(new Todo(3, "Jill", "Learn Hibernate", new Date(), false));
        todos.add(new Todo(4, "Jill", "Learn Spring MVC", new Date(), false));
    }

}
```

## 13. Retrieving a Todo List

We will create a new **RestController** annotation called **TodoController.java**.



14. Add a service method to return all the todos for the **/todos** API

So, add a method in **TodoService.java**

```
public List<Todo> retrieveTodos(String user) {  
    List<Todo> filteredTodos = new ArrayList<Todo>();  
    for (Todo todo : todos) {  
        if (todo.getUser().equals(user))  
            filteredTodos.add(todo);  
    }  
    return filteredTodos;  
}
```

15. Add an API resource method in the **TodoController.java** for retrieving all the todos

The code for the **retrieveTodos()** method is as follows:

```
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RestController;  
  
import com.demo.model.Todo;  
import com.demo.service.TodoService;  
  
@RestController  
public class TodoController {  
  
    @Autowired  
    private TodoService todoService;  
  
    @GetMapping("/users/{name}/todos")  
    public List<Todo> retrieveTodos(@PathVariable String name) {  
        return todoService.retrieveTodos(name);  
    }  
}
```

## 16. Retrieving details for a specific Todo

### 16.1 TodoService.java

Add a method to find a todo

```
public Todo retrieveTodo(int id) {  
    for (Todo todo : todos) {  
        if (todo.getId() == id)  
            return todo;  
    }  
    return null;  
}
```

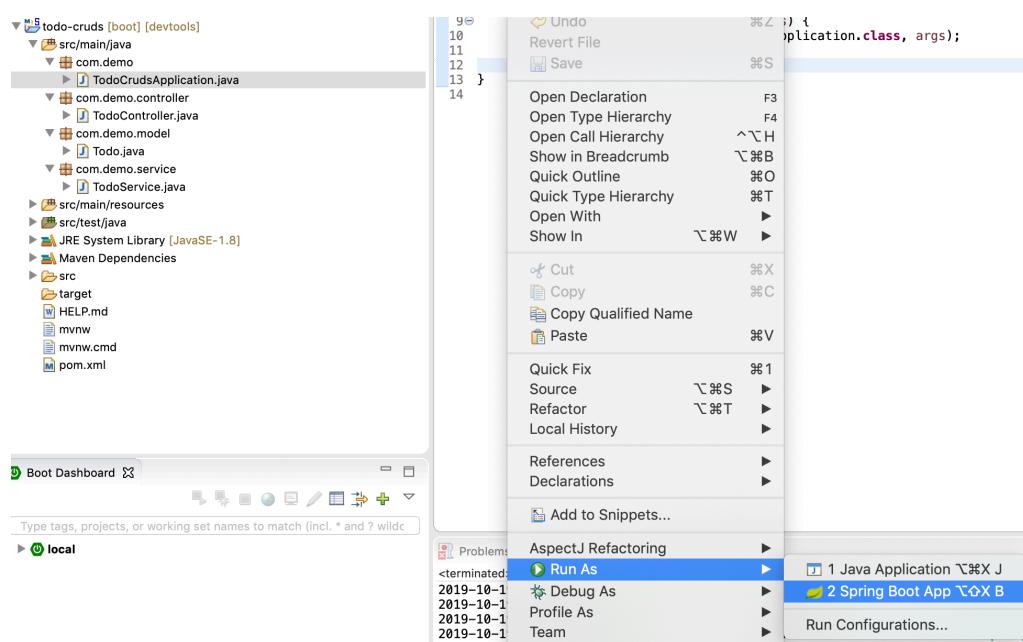
### 16.2 TodoController.java

Add an API resource method to find a Todo based on an ID

```
@GetMapping(path = "/users/{name}/todos/{id}")  
public Todo retrieveTodo(@PathVariable String name, @PathVariable int id) {  
    return todoService.retrieveTodo(id);  
}
```

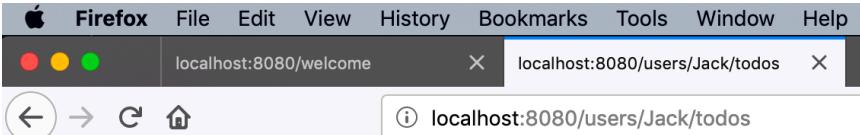
## 17. RUN the App to test these APIs

### 17.1



18. Observe the output for the list of todos for each user

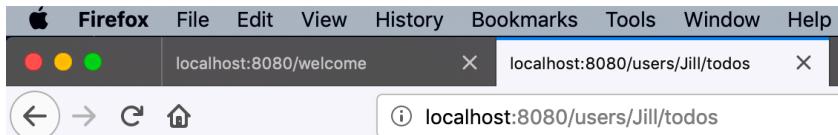
#### 18.1 Visit: <http://localhost:8080/users/Jack/todos>



The screenshot shows the Firefox browser window. The address bar has two tabs: "localhost:8080/welcome" and "localhost:8080/users/Jack/todos". The main content area displays a JSON array of todos for user "Jack".

```
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
▼ 0:
  id: 1
  user: "Jack"
  desc: "Learn Spring MVC"
  targetDate: "2019-10-20T02:57:18.924+0000"
  done: false
▼ 1:
  id: 2
  user: "Jack"
  desc: "Learn Struts"
  targetDate: "2019-10-20T02:57:18.924+0000"
  done: false
```

#### 18.2 Visit: <http://localhost:8080/users/Jill/todos>

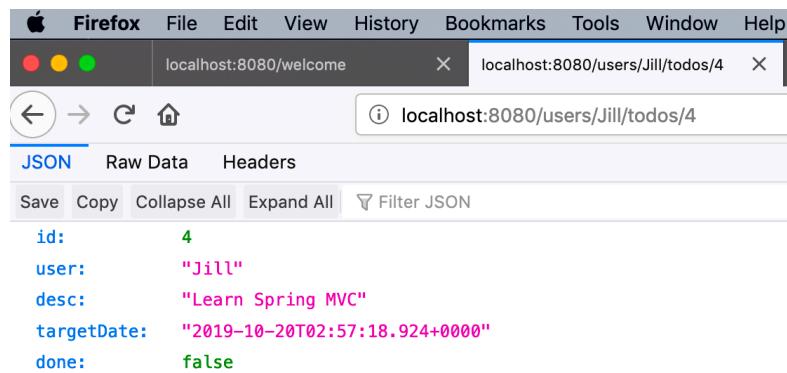


The screenshot shows the Firefox browser window. The address bar has two tabs: "localhost:8080/welcome" and "localhost:8080/users/Jill/todos". The main content area displays a JSON array of todos for user "Jill".

```
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
▼ 0:
  id: 3
  user: "Jill"
  desc: "Learn Hibernate"
  targetDate: "2019-10-20T02:57:18.924+0000"
  done: false
▼ 1:
  id: 4
  user: "Jill"
  desc: "Learn Spring MVC"
  targetDate: "2019-10-20T02:57:18.924+0000"
  done: false
```

## 19. Find a specific todo task of a user

19.1 Visit: <http://localhost:8080/users/Jill/todos/4>



The screenshot shows a Firefox browser window with two tabs open. The left tab is titled 'localhost:8080/welcome' and the right tab is titled 'localhost:8080/users/Jill/todos/4'. The right tab's address bar also displays the URL 'localhost:8080/users/Jill/todos/4'. Below the tabs, there are three buttons: 'JSON', 'Raw Data', and 'Headers'. The 'JSON' button is selected. At the bottom of the browser window, there are buttons for 'Save', 'Copy', 'Collapse All', 'Expand All', and 'Filter JSON'. The main content area displays a JSON object:

```
id: 4
user: "Jill"
desc: "Learn Spring MVC"
targetDate: "2019-10-20T02:57:18.924+0000"
done: false
```

20. Similarly, add all the other operations in TodoService.java for respective API resource method

### TodoService.java

```
public Todo addTodo(String name, String desc, Date targetDate, boolean isDone) {
    Todo todo = new Todo(++todoCount, name, desc, targetDate, isDone);
    todos.add(todo);
    return todo;
}

public Todo retrieveTodo(int id) {
    for (Todo todo : todos) {
        if (todo.getId() == id)
            return todo;
    }
    return null;
}

public Todo update(Todo todo) {
    Todo deletedTodo = deleteById(todo.getId());
    if (deletedTodo == null)
        throw new RuntimeException("Todo not found");
    todos.add(todo);
    return todo;
}

public Todo deleteById(int id) {
    Todo todo = retrieveTodo(id);

    if (todo == null)
        throw new RuntimeException("Todo not found");

    if (todos.remove(todo))
        return todo;
    }

    throw new RuntimeException("Delete Unsuccessful");
}
```

## 21. Add all the Resource API methods in the **RestController**

Here you go with all those API methods:

```
@DeleteMapping("/users/{name}/todos/{id}")
    public ResponseEntity<Void> deleteTodo(@PathVariable
String name, @PathVariable int id) {

    Todo todo = todoService.deleteById(id);

    System.out.println(todo);

    if (todo != null) {
        return ResponseEntity.noContent().build();
    }

    return ResponseEntity.notFound().build();
}

@PutMapping("/users/{name}/todos/{id}")
public ResponseEntity<Todo> updateTodo(@PathVariable
String name, @PathVariable int id, @RequestBody Todo todo) {

    System.out.println(todo);

    todoService.update(todo);

    return new ResponseEntity<Todo>(todo, HttpStatus.OK);
}

@PostMapping("/users/{name}/todos")
    ResponseEntity<?> add(@PathVariable String name,
@RequestBody Todo todo) {
    Todo createdTodo = todoService.addTodo(name,
todo.getDesc(), todo.getTargetDate(), todo.isDone());
    if (createdTodo == null) {
        return ResponseEntity.noContent().build();
    }
    URI location =
ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
        .buildAndExpand(createdTodo.getId()).toUri();
    return ResponseEntity.created(location).build();
}
```

22. **TodoCrudsApplication.java** →  
Right Click → Run As →  
Spring Boot App

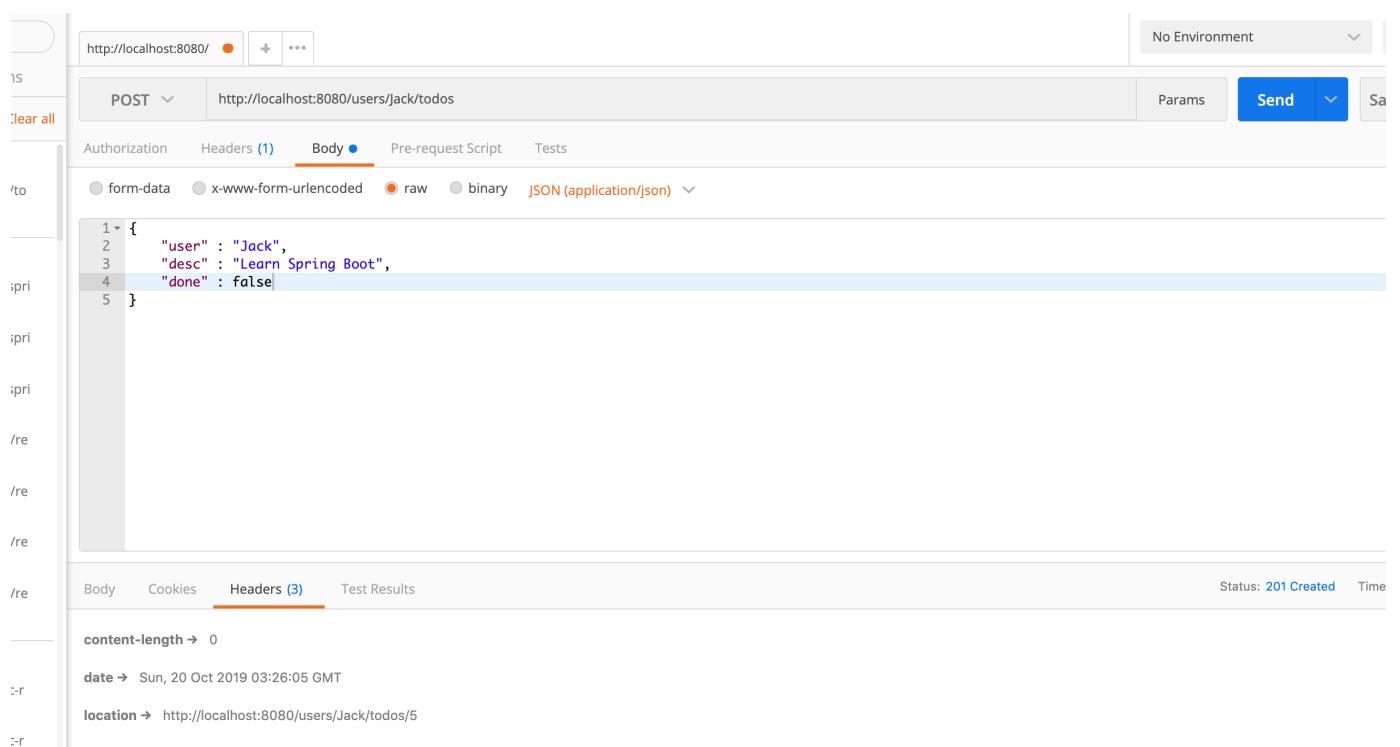
## 23. OUTPUT

### 23.1 Create a new Todo using **POSTMAN**

**URL:** <http://localhost:8080/users/Jack/todos>  
**Data:**

```
{  
    "user" : "Jack",  
    "desc" : "Learn Spring Boot",  
    "done" : false  
}
```

**Method:** POST

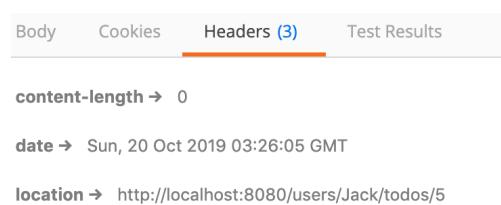


The screenshot shows the Postman application interface. The URL field contains `http://localhost:8080/users/Jack/todos`. The method is set to `POST`. In the `Body` tab, the `raw` radio button is selected, and the JSON payload is defined as follows:

```
1 {  
2     "user" : "Jack",  
3     "desc" : "Learn Spring Boot",  
4     "done" : false  
5 }
```

Below the body, the `Headers` tab is selected, showing three headers: `content-length`, `date`, and `location`. The status bar at the bottom right indicates `Status: 201 Created`.

### 23.2 Click on **Send** button and observe the response in **Headers** Tab



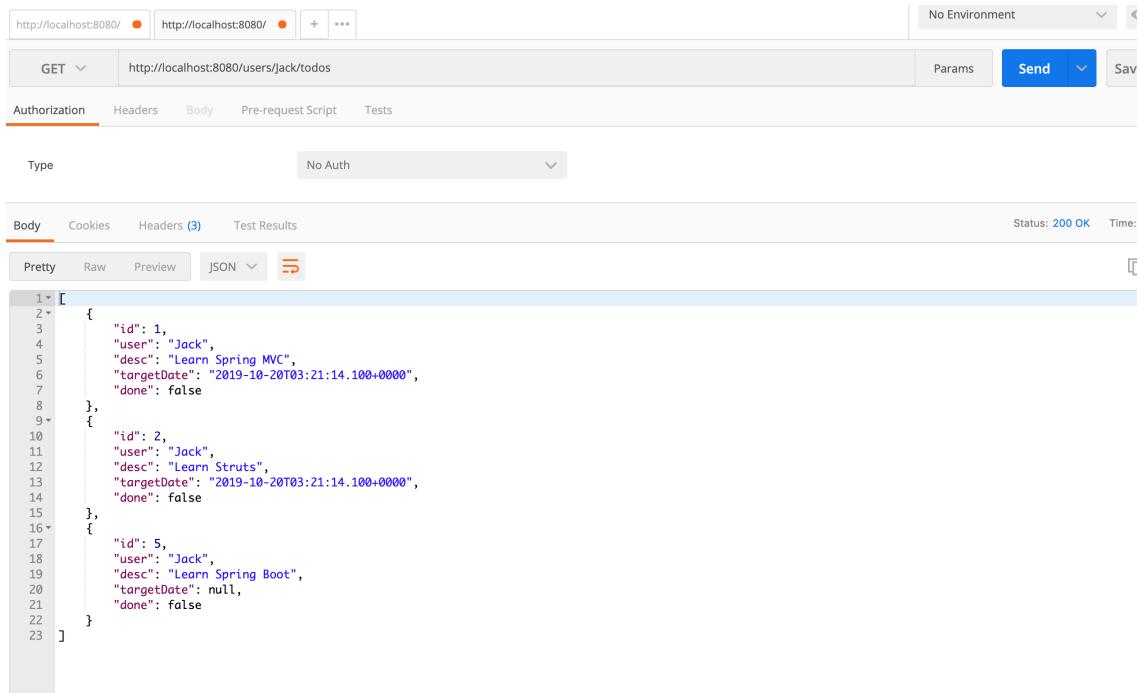
The screenshot shows the Headers tab from the previous Postman session. It displays the following response headers:

- `content-length` → 0
- `date` → Sun, 20 Oct 2019 03:26:05 GMT
- `location` → <http://localhost:8080/users/Jack/todos/5>

### 23.3 List all the Todos for a user in POSTMAN

**Method:** GET

**URL:** *http://localhost:8080/users/Jack/todos*



The screenshot shows the POSTMAN application interface. At the top, there are two tabs: 'http://localhost:8080/' and 'http://localhost:8080/todos'. The 'Todos' tab is active. Below the tabs, the URL 'http://localhost:8080/users/Jack/todos' is entered. On the right side, there are buttons for 'Send' and 'Save'. Under the URL input, there are tabs for 'Authorization', 'Headers', 'Body', 'Pre-request Script', and 'Tests'. The 'Authorization' tab is selected. Below the tabs, there is a dropdown menu set to 'No Auth'. In the main body area, there are tabs for 'Body', 'Cookies', 'Headers (3)', and 'Test Results'. The 'Body' tab is selected. It displays a JSON response with three items. The JSON is formatted with line numbers from 1 to 23. The response is as follows:

```
1  [
2   {
3     "id": 1,
4     "user": "Jack",
5     "desc": "Learn Spring MVC",
6     "targetDate": "2019-10-20T03:21:14.100+0000",
7     "done": false
8   },
9   {
10    "id": 2,
11    "user": "Jack",
12    "desc": "Learn Struts",
13    "targetDate": "2019-10-20T03:21:14.100+0000",
14    "done": false
15  },
16  {
17    "id": 5,
18    "user": "Jack",
19    "desc": "Learn Spring Boot",
20    "targetDate": null,
21    "done": false
22  }
23 ]
```

At the top right of the main area, it says 'Status: 200 OK'.

## 23.4 Edit a Todo task for a User

**Method:** PUT

**URL:** *http://localhost:8080/users/Jack/todos/2*

The screenshot shows the Postman application interface. At the top, there are three tabs for environments: 'http://localhost:8080/' (selected), 'http://localhost:8080/' (disabled), and 'http://localhost:8080/' (disabled). On the right, there are buttons for 'Send' and 'Params'. Below the tabs, the method is set to 'PUT' and the URL is 'http://localhost:8080/users/Jack/todos/2'. Under the 'Body' tab, which is selected, there is a JSON editor. The JSON payload is:

```
1 {  
2   "id": 2,  
3   "user": "Jack",  
4   "desc": "Learn Struts 2 Framework",  
5   "targetDate": "2019-10-20T03:46:44.124+0000",  
6   "done": false  
7 }
```

Below the body editor, there are tabs for 'Body', 'Cookies', 'Headers (3)', and 'Test Results'. The status bar at the bottom right indicates 'Status: 200 OK'. The 'Body' tab is currently active.

## 23.5 Add a Todo Task with Date

The screenshot shows the Postman application interface. At the top, there are three tabs with the URL `http://localhost:8080/` and a red error icon. To the right of the tabs are buttons for `+` and `...`. Below the tabs, the method is set to `POST` and the endpoint is `http://localhost:8080/users/Jill/todos`. The `Body` tab is selected, indicated by an orange underline. Below it, there are four options: `form-data`, `x-www-form-urlencoded`, `raw`, and `binary`. The `raw` option is selected and highlighted with an orange circle. To the right of the selection is the `JSON (application/json)` label. The JSON payload is displayed in the body:

```
1 {  
2   "user" : "Jill",  
3   "desc" : "Learn Spring Boot",  
4   "targetDate": "2019-10-20T03:46:44.124+0000",  
5   "done" : false  
6 }
```

Body Cookies Headers (3) Test Results

**content-length** → 0

**date** → Sun, 20 Oct 2019 03:59:52 GMT

**location** → <http://localhost:8080/users/Jill/todos/6>





