International Institute of Information Technology, Bangalore.

Software Testing CSE 731
Project Report

TASK MANAGER-MUTATION TESTING

In the guidance of Prof. Meenakshi D Souza

Aditi Goel

MT2023034

Simrath Kaur

MT2023066

Contents

Project Overview

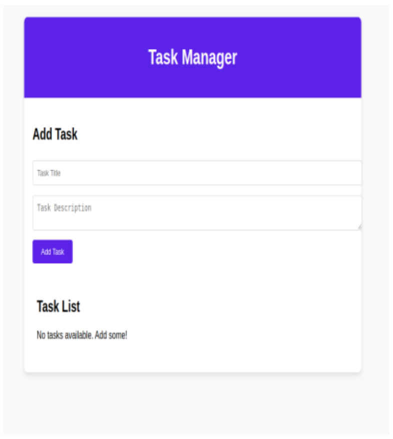
The Task Management System is a **full-stack application**, with the **backend implemented using Spring Boot**. It provides an API to manage tasks through CRUD operations. While the frontend handles user interaction, testing was conducted on the **backend** to ensure the functionality and reliability of the service layer and API endpoints.

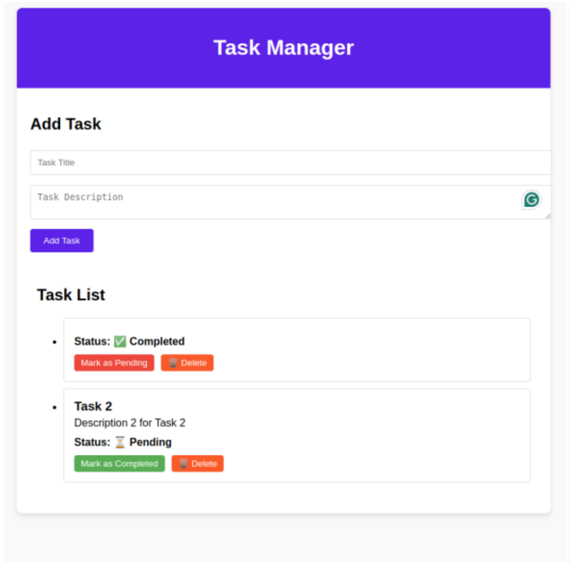
Backend Architecture:

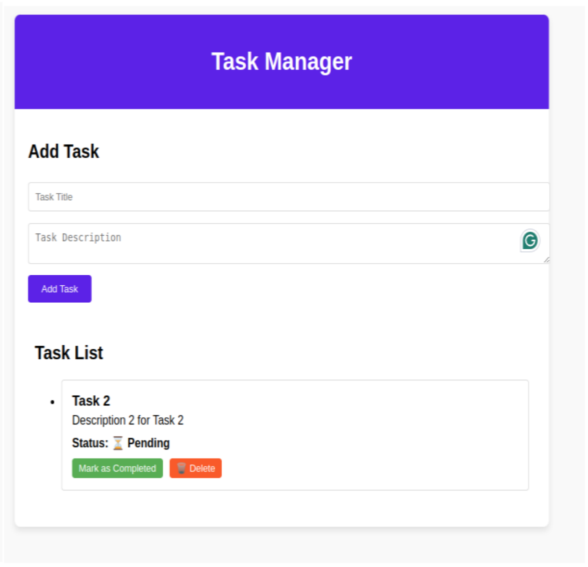
- **Controller Layer**: Handles incoming API requests and sends responses.
- **Service Layer**: Implements the business logic for task management.
- **Model Layer**: Defines the data structure (e.g., Task entity).
- **Repository Layer**: Manages database operations (not directly tested but mocked during testing).

Purpose of Testing:

- Validate backend functionality.
- Ensure code robustness against potential bugs using **unit tests** and **mutation testing**.







Testing Approach

Manual Test Design

We began by identifying edge cases and scenarios for key operations like task creation, retrieval, updates, and deletion:

1. Valid inputs (e.g., creating a task with all required fields).
2. Invalid inputs (e.g., missing fields, null values).
3. Boundary cases (e.g., task ID does not exist).
4. Error scenarios (e.g., database or server failure).

Automated Testing

Tools Used:

- **JUnit**: For writing and executing unit tests.
- **Mockito**: To mock dependencies and isolate testing to specific layers.

Layers Tested:

1. **Controller Layer**: Ensures proper API responses for different request scenarios.
2. **Service Layer**: Validates business logic, including edge-case handling.
3. **Model Layer**: Verifies data structure and validation logic.

Test Types:

- **Unit Tests**: Target specific methods in isolation.
- **Integration Tests**: Ensure proper flow between layers (controller and service).

Mutation Testing

Overview:

Mutation testing evaluates the quality of our tests by introducing small changes (mutants) to the code and observing whether the tests can detect and fail due to these changes.

Tool Used:

- **PIT (Pitest)**, a mutation testing tool for Java.

Full List of Mutators

Mutator	Level	Description
FALSE_RETURNS	Unit	Replaces return values with `false`.
NULL_RETURNS	Unit	Forces methods to return `null`.
MATH	Unit	Alters arithmetic operators.
INCREMENTS	Unit	Changes increment/decrement operations.
NEGATE_CONDITIONALS	Integration	Inverts logical conditions.
VOID_METHOD_CALLS	Integration	Removes void method calls.
CONDITIONALS_BOUNDARY	Integration	Alters conditional boundaries.

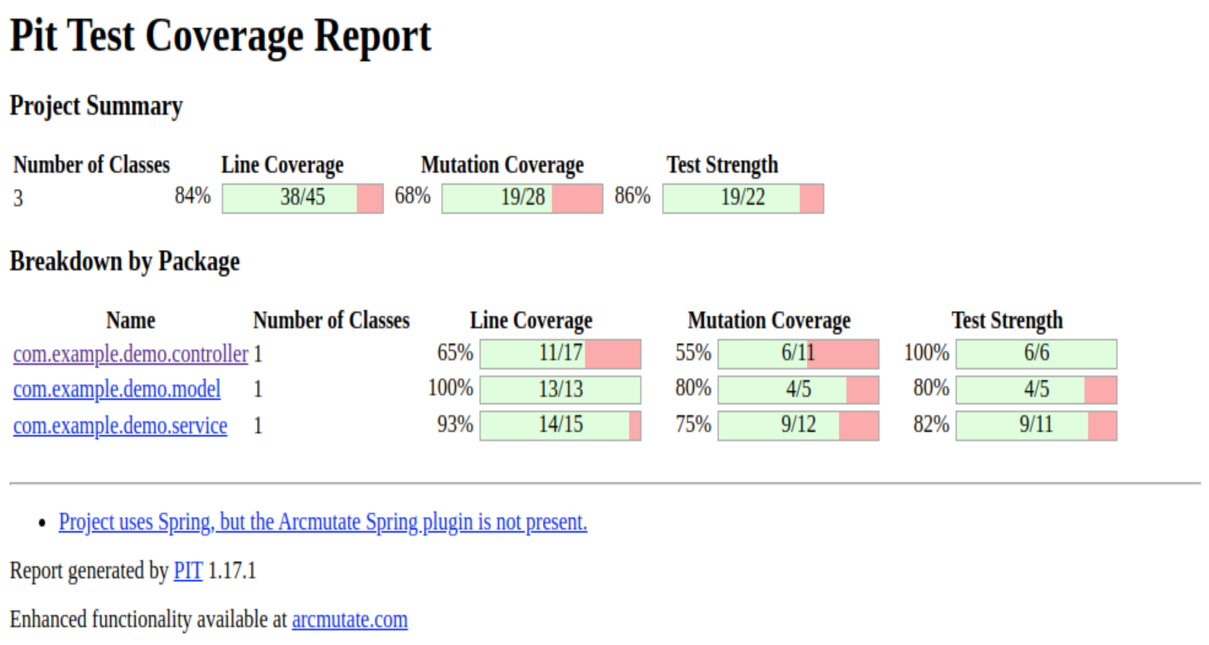
Test Results

Summary Table

Layer	Line Coverage	Mutation Coverage	Test Strength
Controller	65% (11/17)	55% (6/11)	100% (6/6)
Model	100% (13/13)	80% (4/5)	80% (4/5)
Service	93% (14/15)	75% (9/12)	82% (9/11)
Overall	84% (38/45)	68% (19/28)	86% (19/22)

Key Metrics

1. **Line Coverage**:
    - The percentage of code lines executed during tests.
    - Indicates the breadth of testing.
2. **Mutation Coverage**:
    - The percentage of mutants (introduced bugs) detected by the tests.
    - Reflects the depth of testing.
3. **Test Strength**:
    - The proportion of mutants killed relative to the total mutants created.



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
3	84%	38/45	68%	19/28	86%	19/22

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
com.example.demo.controller	1	65%	11/17	55%	6/11	100%	6/6
com.example.demo.model	1	100%	13/13	80%	4/5	80%	4/5
com.example.demo.service	1	93%	14/15	75%	9/12	82%	9/11

- Project uses Spring, but the Arcmutate Spring plugin is not present.

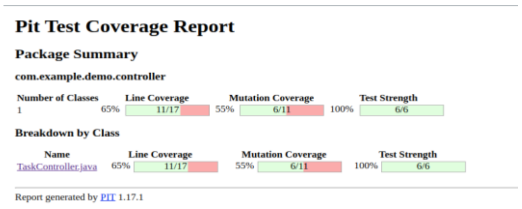
Report generated by PIT 1.17.1

Enhanced functionality available at arcmutate.com

Detailed Layer-Wise Analysis

Controller Layer

- **Line Coverage**: 65%
- **Mutation Coverage**: 55%
- **Test Strength**: 100%
- **Findings**:
    - Missed branch conditions in methods like `updateTask` and `deleteTask`.
    - Null-return scenarios were untested in some endpoints.
- **Actions Taken**:
    - Enhanced `TaskControllerTest` with additional test cases for edge cases (e.g., null task returns, invalid IDs).



Pit Test Coverage Report

Package Summary

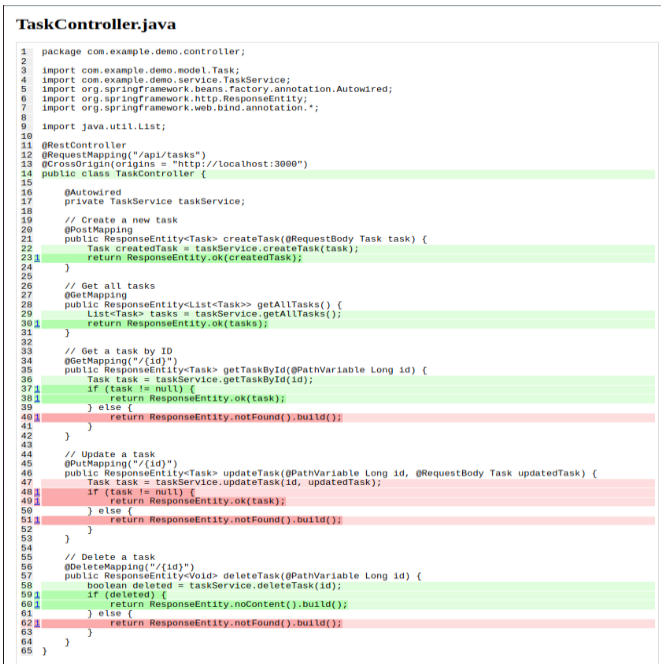
com.example.demo.controller

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	65%	11/17	55%	6/11	100%	6/6

Breakdown by Class

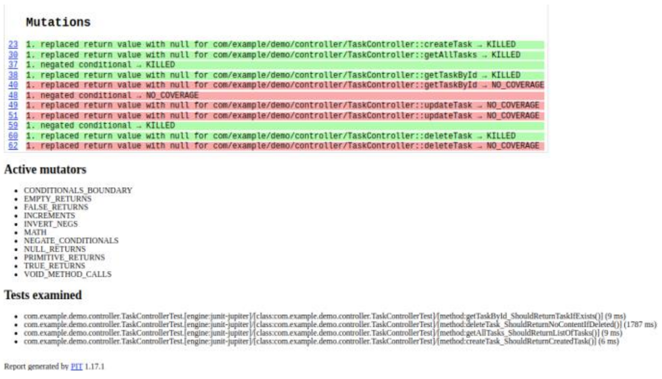
Name	Line Coverage		Mutation Coverage		Test Strength	
TaskController.java	65%	11/17	55%	6/11	100%	6/6

Report generated by PIT 1.17.1



TaskController.java

```
1    package com.example.demo.controller;
2
3    import com.example.demo.model.Task;
4    import com.example.demo.service.TaskService;
5    import org.springframework.beans.factory.annotation.Autowired;
6    import org.springframework.http.ResponseEntity;
7    import org.springframework.web.bind.annotation.*;
8
9    import java.util.List;
10
11   @RestController
12   @RequestMapping("/api/tasks")
13   @CrossOrigin(origins = "http://localhost:3000")
14   public class TaskController {
15
16       @Autowired
17       private TaskService taskService;
18
19       // Create a new task
20       @PostMapping
21       public ResponseEntity<Task> createTask(@RequestBody Task task) {
22           Task createdTask = taskService.createTask(task);
23 1         return ResponseEntity.ok(createdTask);
24       }
25
26       // Get all tasks
27       @GetMapping
28       public ResponseEntity<List<Task>> getAllTasks() {
29           List<Task> tasks = taskService.getAllTasks();
30 1         return ResponseEntity.ok(tasks);
31       }
32
33       // Get a task by ID
34       @GetMapping("/{id}")
35       public ResponseEntity<Task> getTaskById(@PathVariable Long id) {
36           Task task = taskService.getTaskById(id);
37 1         if (task != null) {
38 1             return ResponseEntity.ok(task);
39           } else {
40 1             return ResponseEntity.notFound().build();
41           }
42       }
43
44       // Update a task
45       @PutMapping("/{id}")
46       public ResponseEntity<Task> updateTask(@PathVariable Long id, @RequestBody Task updatedTask) {
47           Task task = taskService.updateTask(id, updatedTask);
48 1         if (task != null) {
49 1             return ResponseEntity.ok(task);
50           } else {
51 1             return ResponseEntity.notFound().build();
52           }
53       }
54
55       // Delete a task
56       @DeleteMapping("/{id}")
57       public ResponseEntity<Void> deleteTask(@PathVariable Long id) {
58           boolean deleted = taskService.deleteTask(id);
59 1         if (deleted) {
60 1             return ResponseEntity.noContent().build();
61           } else {
62 1             return ResponseEntity.notFound().build();
63           }
64       }
65   }
```



Mutations

```
23 1. replaced return value with null for com/example/demo/controller/TaskController::createTask → KILLED
30 1. replaced return value with null for com/example/demo/controller/TaskController::getAllTasks → KILLED
37 1. negated conditional → KILLED
38 1. replaced return value with null for com/example/demo/controller/TaskController::getTaskById → KILLED
40 1. replaced return value with null for com/example/demo/controller/TaskController::getTaskById → NO_COVERAGE
48 1. negated conditional → NO_COVERAGE
49 1. replaced return value with null for com/example/demo/controller/TaskController::updateTask → NO_COVERAGE
51 1. replaced return value with null for com/example/demo/controller/TaskController::updateTask → NO_COVERAGE
59 1. negated conditional → KILLED
60 1. replaced return value with null for com/example/demo/controller/TaskController::deleteTask → KILLED
62 1. replaced return value with null for com/example/demo/controller/TaskController::deleteTask → NO_COVERAGE
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

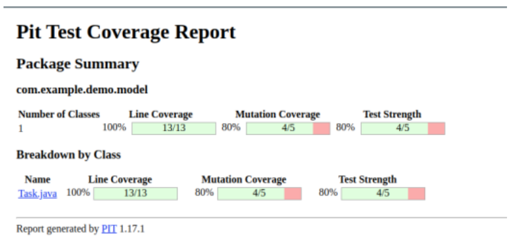
Tests examined

- com.example.demo.controller.TaskControllerTest.[engine:junit-jupiter]/[class:com.example.demo.controller.TaskControllerTest]/[method:getTaskById_ShouldReturnTaskIfExists()] (9 ms)
- com.example.demo.controller.TaskControllerTest.[engine:junit-jupiter]/[class:com.example.demo.controller.TaskControllerTest]/[method:deleteTask_ShouldReturnNoContentIfDeleted()] (1787 ms)
- com.example.demo.controller.TaskControllerTest.[engine:junit-jupiter]/[class:com.example.demo.controller.TaskControllerTest]/[method:getAllTasks_ShouldReturnListOfTasks()] (9 ms)
- com.example.demo.controller.TaskControllerTest.[engine:junit-jupiter]/[class:com.example.demo.controller.TaskControllerTest]/[method:createTask_ShouldReturnCreatedTask()] (6 ms)

Report generated by PIT 1.17.1

Model Layer

- **Line Coverage**: 100%
- **Mutation Coverage**: 80%
- **Test Strength**: 80%
- **Findings**:
  - Most scenarios tested, but one mutant involving edge-case validation was missed.
- **Actions Taken**:
  - Added model-specific validation tests.



Pit Test Coverage Report

Package Summary

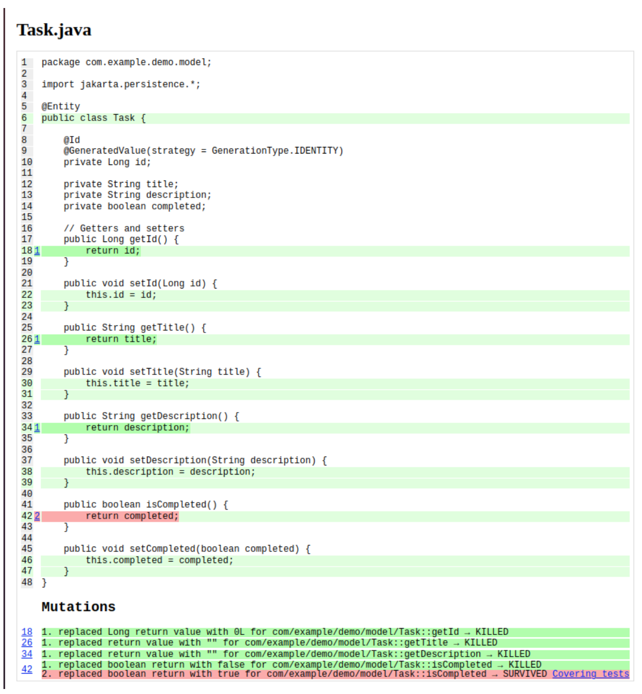
com.example.demo.model

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	100%	13/13	80%	4/5	80%	4/5

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Task.java	100%	13/13	80%	4/5	80%	4/5

Report generated by PIT 1.17.1

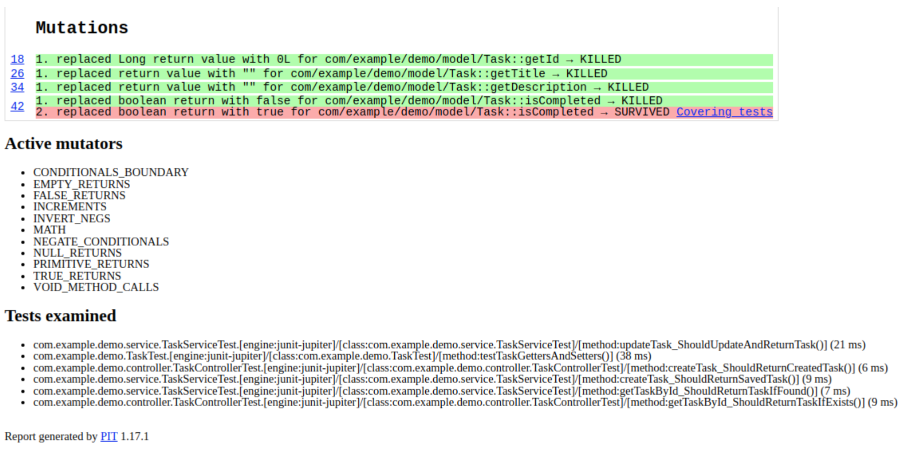


Task.java

```
1    package com.example.demo.model;
2
3    import jakarta.persistence.*;
4
5    @Entity
6    public class Task {
7
8        @Id
9        @GeneratedValue(strategy = GenerationType.IDENTITY)
10       private Long id;
11
12       private String title;
13       private String description;
14       private boolean completed;
15
16       // Getters and setters
17       public Long getId() {
18 1         return id;
19       }
20
21       public void setId(Long id) {
22           this.id = id;
23       }
24
25       public String getTitle() {
26 1         return title;
27       }
28
29       public void setTitle(String title) {
30           this.title = title;
31       }
32
33       public String getDescription() {
34 1         return description;
35       }
36
37       public void setDescription(String description) {
38           this.description = description;
39       }
40
41       public boolean isCompleted() {
42 2         return completed;
43       }
44
45       public void setCompleted(boolean completed) {
46           this.completed = completed;
47       }
48   }
```

Mutations

```
18  1. replaced Long return value with 0L for com/example/demo/model/Task::getId → KILLED
26  1. replaced return value with "" for com/example/demo/model/Task::getTitle → KILLED
34  1. replaced return value with "" for com/example/demo/model/Task::getDescription → KILLED
42  1. replaced boolean return with false for com/example/demo/model/Task::isCompleted → KILLED
    2. replaced boolean return with true for com/example/demo/model/Task::isCompleted → SURVIVED Covering tests
```



Mutations

```
18  1. replaced Long return value with 0L for com/example/demo/model/Task::getId → KILLED
26  1. replaced return value with "" for com/example/demo/model/Task::getTitle → KILLED
34  1. replaced return value with "" for com/example/demo/model/Task::getDescription → KILLED
42  1. replaced boolean return with false for com/example/demo/model/Task::isCompleted → KILLED
    2. replaced boolean return with true for com/example/demo/model/Task::isCompleted → SURVIVED Covering tests
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

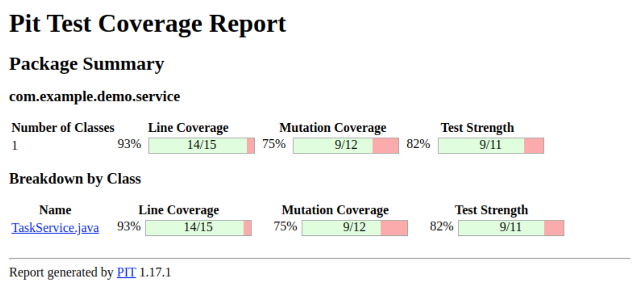
Tests examined

- com.example.demo.service.TaskServiceTest.[engine:junit-jupiter]/[class:com.example.demo.service.TaskServiceTest]/[method:updateTask_ShouldUpdateAndReturnTask()] (21 ms)
- com.example.demo.TaskTest.[engine:junit-jupiter]/[class:com.example.demo.TaskTest]/[method:testTaskGettersAndSetters()] (38 ms)
- com.example.demo.controller.TaskControllerTest.[engine:junit-jupiter]/[class:com.example.demo.controller.TaskControllerTest]/[method:createTask_ShouldReturnCreatedTask()] (6 ms)
- com.example.demo.service.TaskServiceTest.[engine:junit-jupiter]/[class:com.example.demo.service.TaskServiceTest]/[method:createTask_ShouldReturnSavedTask()] (9 ms)
- com.example.demo.service.TaskServiceTest.[engine:junit-jupiter]/[class:com.example.demo.service.TaskServiceTest]/[method:getTaskById_ShouldReturnTaskIfFound()] (7 ms)
- com.example.demo.controller.TaskControllerTest.[engine:junit-jupiter]/[class:com.example.demo.controller.TaskControllerTest]/[method:getTaskById_ShouldReturnTaskIfExists()] (9 ms)

Report generated by PIT 1.17.1

Service Layer

- **Line Coverage**: 93%
- **Mutation Coverage**: 75%
- **Test Strength**: 82%
- **Findings**:
  - Missed conditions in error handling and null cases.
- **Actions Taken**:
  - Enhanced `TaskServiceTest` with cases for exception handling and invalid operations.



Pit Test Coverage Report

Package Summary

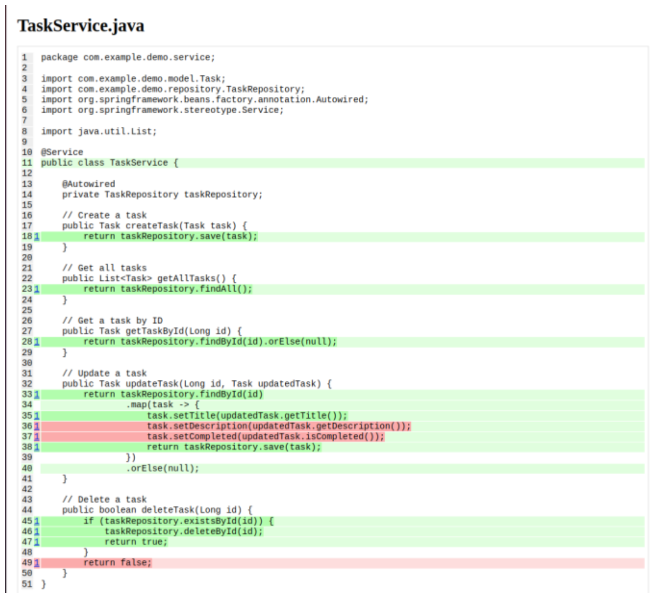
com.example.demo.service

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	93%	14/15	75%	9/12	82%	9/11

Breakdown by Class

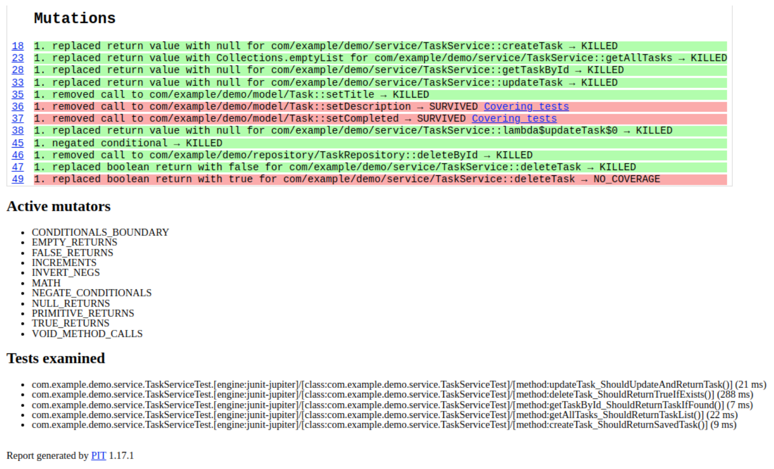
Name	Line Coverage		Mutation Coverage		Test Strength	
TaskService.java	93%	14/15	75%	9/12	82%	9/11

Report generated by PIT 1.17.1



TaskService.java

```java
package com.example.demo.service;

import com.example.demo.model.Task;
import com.example.demo.repository.TaskRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class TaskService {

    @Autowired
    private TaskRepository taskRepository;

    // Create a task
    public Task createTask(Task task) {
        return taskRepository.save(task);
    }

    // Get all tasks
    public List<Task> getAllTasks() {
        return taskRepository.findAll();
    }

    // Get a task by ID
    public Task getTaskById(Long id) {
        return taskRepository.findById(id).orElse(null);
    }

    // Update a task
    public Task updateTask(Long id, Task updatedTask) {
        return taskRepository.findById(id)
                .map(task -> {
                    task.setTitle(updatedTask.getTitle());
                    task.setDescription(updatedTask.getDescription());
                    task.setCompleted(updatedTask.isCompleted());
                    return taskRepository.save(task);
                })
                .orElse(null);
    }

    // Delete a task
    public boolean deleteTask(Long id) {
        if (taskRepository.existsById(id)) {
            taskRepository.deleteById(id);
            return true;
        }
        return false;
    }
}
```



Mutations

```
18 1. replaced return value with null for com/example/demo/service/TaskService::createTask → KILLED
23 1. replaced return value with Collections.emptyList for com/example/demo/service/TaskService::getAllTasks → KILLED
28 1. replaced return value with null for com/example/demo/service/TaskService::getTaskById → KILLED
33 1. replaced return value with null for com/example/demo/service/TaskService::updateTask → KILLED
35 1. removed call to com/example/demo/model/Task::setTitle → KILLED
36 1. removed call to com/example/demo/model/Task::setDescription → SURVIVED Covering tests
37 1. removed call to com/example/demo/model/Task::setCompleted → SURVIVED Covering tests
38 1. replaced return value with null for com/example/demo/service/TaskService::lambda$updateTask$0 → KILLED
45 1. negated conditional → KILLED
46 1. removed call to com/example/demo/repository/TaskRepository::deleteById → KILLED
47 1. replaced boolean return with false for com/example/demo/service/TaskService::deleteTask → KILLED
49 1. replaced boolean return with true for com/example/demo/service/TaskService::deleteTask → NO_COVERAGE
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- com.example.demo.service.TaskServiceTest.[engine:junit-jupiter]/[class:com.example.demo.service.TaskServiceTest]/[method:updateTask_ShouldUpdateAndReturnTask()] (21 ms)
- com.example.demo.service.TaskServiceTest.[engine:junit-jupiter]/[class:com.example.demo.service.TaskServiceTest]/[method:deleteTask_ShouldReturnTrueIfExists()] (288 ms)
- com.example.demo.service.TaskServiceTest.[engine:junit-jupiter]/[class:com.example.demo.service.TaskServiceTest]/[method:getTaskById_ShouldReturnTaskIfFound()] (7 ms)
- com.example.demo.service.TaskServiceTest.[engine:junit-jupiter]/[class:com.example.demo.service.TaskServiceTest]/[method:getAllTasks_ShouldReturnTaskList()] (22 ms)
- com.example.demo.service.TaskServiceTest.[engine:junit-jupiter]/[class:com.example.demo.service.TaskServiceTest]/[method:createTask_ShouldReturnSavedTask()] (9 ms)

Report generated by PIT 1.17.1

Challenges and Solutions

1. **Low Mutation Coverage**:
   - Identified gaps using PIT reports (e.g., untested branches in controller methods).
   - Solution: Added targeted tests to kill uncovered mutants.
2. **Testing Spring Boot-Specific Logic**:
   - Spring's boilerplate code creates non-functional mutants.
   - Solution: Focused on testing business logic and ignoring framework-generated code.

Tools and Technologies

- **Spring Boot**: Framework for backend development.
- **JUnit**: Testing framework for writing and executing test cases.
- **Mockito**: For mocking dependencies during unit testing.
- **PIT (Pitest)**: Mutation testing tool to evaluate test quality.

Conclusion

Testing and mutation analysis have improved the quality of the backend:

- **Line Coverage**: 84% overall ensures broad testing.
- **Mutation Coverage**: 68% reflects good test depth, with room for improvement.
- **Test Strength**: 86% indicates that most mutants are detected and killed.

Future Steps:

1. Improve mutation coverage to over 80% by addressing remaining gaps.
2. Add integration tests for multi-layer validation.
3. Explore tools like **Arcmutate** to enhance Spring-specific testing.

The backend is now robust, with well-tested functionality that ensures reliable task management.