

ProjectAllocationServiceImpl.java

```
40 import java.util.List;
20
21 @Service(value="projectService")
22 @Transactional
23 public class ProjectAllocationServiceImpl implements ProjectAllocationService {
24
25     @Autowired
26     private ProjectRepository projectRepository;
27     @Autowired
28     private MentorRepository mentorRepository;
29     @Override
30     public Integer allocateProject(ProjectDTO project) throws InfyInternException {
31         Optional<Mentor> optional = mentorRepository.findById(project.getMentorDTO().getMentorId());
32         Mentor mentor = optional.orElseThrow(() -> new InfyInternException("Service.MENTOR_NOT_FOUND"));
33         if(mentor.getNumberOfProjectsMentored()>=3)
34             throw new InfyInternException("Service.CANNOT_ALLOCATE_PROJECT");
35         Project project1 = new Project();
36         project1.setProjectId(project.getProjectId());
37         project1.setProjectName(project.getProjectName());
38         project1.setIdeaOwner(project.getIdeaOwner());
39         project1.setReleaseDate(project.getReleaseDate());
40         project1.setMentor(mentor);
41         mentor.setNumberOfProjectsMentored(mentor.getNumberOfProjectsMentored()+1);
42         Project project2 = projectRepository.save(project1);
43
44         return project2.getProjectId();
45     }
46
47
48     @Override
49     public List<MentorDTO> getMentors(Integer numberOfProjectsMentored) throws InfyInternException {
50         List<Mentor> list1 = mentorRepository.findByNumberOfProjectsMentored(numberOfProjectsMentored);
51         if(list1.isEmpty())
52             throw new InfyInternException("Service.MENTOR_NOT_FOUND");
53         List<MentorDTO> list2 = list1.stream().map(x -> new MentorDTO(x.getMentorId(),x.getMentorName(),x.getN
```


ProjectAllocationServiceImpl.java

Quick Access

```
40     project1.setMentor(mentor);
41     mentor.setNumberOfProjectsMentored(mentor.getNumberOfProjectsMentored()+1);
42     Project project2 = projectRepository.save(project1);
43
44     return project2.getProjectId();
45 }
46
47
48 @Override
49 public List<MentorDTO> getMentors(Integer numberOfProjectsMentored) throws InfyInternException {
50     List<Mentor> list1 = mentorRepository.findByNumberOfProjectsMentored(numberOfProjectsMentored);
51     if(list1.isEmpty())
52         throw new InfyInternException("Service.MENTOR_NOT_FOUND");
53     List<MentorDTO> list2 = list1.stream().map(x -> new MentorDTO(x.getMentorId(),x.getMentorName(),x.getNumberOfProjectsMentored()))
54         .collect(Collectors.toList());
55
56     return list2;
57 }
58
59
60 @Override
61 public void updateProjectMentor(Integer projectId, Integer mentorId) throws InfyInternException {
62     Optional<Mentor> optional = mentorRepository.findById(mentorId);
63     Mentor mentor = optional.orElseThrow(() -> new InfyInternException("Service.MENTOR_NOT_FOUND"));
64     if(mentor.getNumberOfProjectsMentored()>=3)
65         throw new InfyInternException("Service.CANNOT_ALLOCATE_PROJECT");
66     Optional<Project> optional1 = projectRepository.findById(projectId);
67     Project project = optional1.orElseThrow(() -> new InfyInternException("Service.PROJECT_NOT_FOUND"));
68     project.setMentor(mentor);
69     mentor.setNumberOfProjectsMentored(mentor.getNumberOfProjectsMentored()+1);
70
71 }
72
73
74 @Override
75 public void deleteProject(Integer projectId) throws InfyInternException {
76     Optional<Project> optional1 = projectRepository.findById(projectId);
77     Project project = optional1.orElseThrow(() -> new InfyInternException("Service.PROJECT_NOT_FOUND"));
78     if(project.getMentor()==null)
79         projectRepository.delete(project);
80     else
81     {
82         Mentor mentor = project.getMentor();
83         mentor.setNumberOfProjectsMentored(mentor.getNumberOfProjectsMentored()-1);
84         project.setMentor(null);
85     }
86 }
```

Writable Smart Insert 1:1:0


```

55     return list2;
56 }
57
58
59
60 @Override
61 public void updateProjectMentor(Integer projectId, Integer mentorId) throws InfyInternException {
62     Optional<Mentor> optional = mentorRepository.findById(mentorId);
63     Mentor mentor = optional.orElseThrow(() -> new InfyInternException("Service.MENTOR_NOT_FOUND"));
64     if(mentor.getNumberOfProjectsMentored()>=3)
65         throw new InfyInternException("Service.CANNOT_ALLOCATE_PROJECT");
66     Optional<Project> optional1 = projectRepository.findById(projectId);
67     Project project = optional1.orElseThrow(() -> new InfyInternException("Service.PROJECT_NOT_FOUND"));
68     project.setMentor(mentor);
69     mentor.setNumberOfProjectsMentored(mentor.getNumberOfProjectsMentored()+1);
70
71 }
72
73
74 @Override
75 public void deleteProject(Integer projectId) throws InfyInternException {
76     Optional<Project> optional1 = projectRepository.findById(projectId);
77     Project project = optional1.orElseThrow(() -> new InfyInternException("Service.PROJECT_NOT_FOUND"));
78     if(project.getMentor()==null)
79         projectRepository.delete(project);
80     else
81     {
82         Mentor mentor = project.getMentor();
83         mentor.setNumberOfProjectsMentored(mentor.getNumberOfProjectsMentored()-1);
84         project.setMentor(null);
85         projectRepository.delete(project);
86     }
87 }
88 }

```

Writable

Smart Insert

1:1:0



ASUS VivoBook


```
1 package com.infy.infyinterns.utility;
2
3 import java.util.stream.Collectors;
4
5 @RestControllerAdvice
6 public class ExceptionControllerAdvice
7 {
8
9     private static final Log LOGGER = LogFactory.getLog(ExceptionControllerAdvice.class);
10
11     @Autowired
12     private Environment environment;
13
14     // add appropriate annotation
15     @ExceptionHandler(InfyInternException.class)
16     public ResponseEntity<ErrorInfo> infyInternExceptionHandler(InfyInternException exception)
17     {
18         LOGGER.error(exception.getMessage(), exception);
19         ErrorInfo errorInfo = new ErrorInfo();
20         errorInfo.setErrorCode(HttpStatus.BAD_REQUEST.value());
21         errorInfo.setErrorMessage(environment.getProperty(exception.getMessage()));
22         return new ResponseEntity<>(errorInfo, HttpStatus.BAD_REQUEST);
23     }
24
25     // add appropriate annotation
26     @ExceptionHandler(Exception.class)
27     public ResponseEntity<ErrorInfo> generalExceptionHandler(Exception exception)
28     {
29         LOGGER.error(exception.getMessage(), exception);
30         ErrorInfo errorInfo = new ErrorInfo();
31         errorInfo.setErrorCode(HttpStatus.INTERNAL_SERVER_ERROR.value());
32         errorInfo.setErrorMessage(environment.getProperty("General.EXCEPTION_MESSAGE"));
33         return new ResponseEntity<>(errorInfo,
34                                     HttpStatus.INTERNAL_SERVER_ERROR);
35     }
36 }
```



```

28 // add appropriate annotation
29 @ExceptionHandler(InfyInternException.class)
30 public ResponseEntity<ErrorInfo> infyInternExceptionHandler(InfyInternException exception)
31 {
32     LOGGER.error(exception.getMessage(), exception);
33     ErrorInfo errorInfo = new ErrorInfo();
34     errorInfo.setErrorCode(HttpStatus.BAD_REQUEST.value());
35     errorInfo.setErrorMessage(environment.getProperty(exception.getMessage()));
36     return new ResponseEntity<>(errorInfo, HttpStatus.BAD_REQUEST);
37 }
38
39 // add appropriate annotation
40 @ExceptionHandler(Exception.class)
41 public ResponseEntity<ErrorInfo> generalExceptionHandler(Exception exception)
42 {
43     LOGGER.error(exception.getMessage(), exception);
44     ErrorInfo errorInfo = new ErrorInfo();
45     errorInfo.setErrorCode(HttpStatus.INTERNAL_SERVER_ERROR.value());
46     errorInfo.setErrorMessage(environment.getProperty("General.EXCEPTION_MESSAGE"));
47     return new ResponseEntity<>(errorInfo,
48                                HttpStatus.INTERNAL_SERVER_ERROR);
49 }
50
51 // add appropriate annotation
52 @ExceptionHandler({MethodArgumentNotValidException.class, ConstraintViolationException.class})
53 public ResponseEntity<ErrorInfo> validatorExceptionHandler(Exception exception)
54 {
55     LOGGER.error(exception.getMessage(), exception);
56     String errorMsg;
57     if (exception instanceof MethodArgumentNotValidException)
58     {
59         MethodArgumentNotValidException manvException = (MethodArgumentNotValidException) exception;
60         errorMsg = manvException.getBindingResult()
61                                .getAllErrors()

```

Writable

Smart Insert

1:1:0

ExceptionControllerAdvice.java LoggingAspect.java

```
1 package com.infy.infyinterns.utility;
2
3 import org.apache.commons.logging.Log;
4
5 @Component
6 @Aspect
7 public class LoggingAspect
8 {
9
10     private static final Log LOGGER = LogFactory.getLog(LoggingAspect.class);
11     @AfterThrowing(pointcut="execution(* com.infy.infyinterns.service.ProjectAllocationServiceImpl.*(..))", throwing="exception")
12     public void logServiceException(InfyInternException exception)
13     {
14         // code
15         LOGGER.error(exception.getMessage(),exception);
16     }
17 }
18
19
20
21
22
23
24
```



```

InfyInternsApplicationTests.java
1 package com.infy.infyinterns;
2
3 import java.time.LocalDate;
4
5
6
7
8
9
10
11 @SpringBootTest
12 public class InfyInternsApplicationTests {
13
14     @Mock
15     private MentorRepository mentorRepository;
16
17
18     @InjectMocks
19     private ProjectAllocationService projectAllocationService = new ProjectAllocationServiceImpl();
20
21
22     @Test
23     public void allocateProjectCannotAllocateTest() throws Exception {
24         MentorDTO mentorDTO = new MentorDTO(100, "Warner", 6);
25         ProjectDTO projectDTO = new ProjectDTO(1, "Shoe Cart", 10012, LocalDate.of(2021, 06, 02), mentorDTO);
26         Mentor mentor = new Mentor();
27         mentor.setMentorId(mentorDTO.getMentorId());
28         mentor.setMentorName(mentorDTO.getMentorName());
29         mentor.setNumberOfProjectsMentored(mentorDTO.getNumberOfProjectsMentored());
30         Mockito.when(mentorRepository.findById(mentorDTO.getMentorId())).thenReturn(Optional.of(mentor));
31         InfyInternException exception = Assertions.assertThrows(InfyInternException.class, () -> projectAllocationService.allocateProject(projectDTO));
32         Assertions.assertEquals("Service.CANNOT_ALLOCATE_PROJECT", exception.getMessage());
33     }
34
35     @Test
36     public void allocateProjectMentorNotFoundTest() throws Exception {
37         MentorDTO mentorDTO = new MentorDTO(1019, "Bethany", 2);
38         ProjectDTO projectDTO = new ProjectDTO(1, "Shoe Cart", 10012, LocalDate.of(2021, 06, 02), mentorDTO);
39
40         Mockito.when(mentorRepository.findById(projectDTO.getMentorDTO().getMentorId())).thenReturn(Optional.empty());
41         InfyInternException exception = Assertions.assertThrows(InfyInternException.class, () -> projectAllocationService.allocateProject(projectDTO));
42         Assertions.assertEquals("Service.MENTOR_NOT_FOUND", exception.getMessage());
43     }
44 }

```



```

sts {

repository;

projectAllocationService = new ProjectAllocationServiceImpl();

allocateTest() throws Exception {
    mentorDTO(100, "Warner", 6);
    projectDTO(1, "Shoe Cart", 10012, LocalDate.of(2021, 06, 02), mentorDTO);

    getMentorId();
    getMentorName();
    assertEquals(mentorDTO.getNumberOfProjectsMentored());
    findById(mentorDTO.getMentorId()).thenReturn(Optional.of(mentor));
    Assertions.assertThrows(InfyInternException.class, () -> projectAllocationService.allocateProject(projectDTO));
    assertEquals("CANNOT_ALLOCATE_PROJECT", exception.getMessage());

}

notFoundTest() throws Exception {
    mentorDTO(1019, "Bethany", 2);
    projectDTO(1, "Shoe Cart", 10012, LocalDate.of(2021, 06, 02), mentorDTO);

    findById(projectDTO.getMentorDTO().getMentorId()).thenReturn(Optional.empty());
    Assertions.assertThrows(InfyInternException.class, () -> projectAllocationService.allocateProject(projectDTO));
    assertEquals("MENTOR_NOT_FOUND", exception.getMessage());
}

```



```

22 public class InfyInternsApplicationTests {
23
24     @Mock
25     private MentorRepository mentorRepository;
26
27     @InjectMocks
28     private ProjectAllocationService projectAllocationService = new ProjectAllocationServiceImpl();
29
30     @Test
31     public void allocateProjectCannotAllocateTest() throws Exception {
32         MentorDTO mentorDTO = new MentorDTO(100, "Warner", 6);
33         ProjectDTO projectDTO = new ProjectDTO(1, "Shoe Cart", 10012, LocalDate.of(2021, 06, 02), mentorDTO);
34         Mentor mentor = new Mentor();
35         mentor.setMentorId(mentorDTO.getMentorId());
36         mentor.setMentorName(mentorDTO.getMentorName());
37         mentor.setNumberOfProjectsMentored(mentorDTO.getNumberOfProjectsMentored());
38         Mockito.when(mentorRepository.findById(mentorDTO.getMentorId())).thenReturn(Optional.of(mentor));
39         InfyInternException exception = Assertions.assertThrows(InfyInternException.class, () -> projectAllocationService.allocateProject(projectDTO));
40         Assertions.assertEquals("Service.CANNOT_ALLOCATE_PROJECT", exception.getMessage());
41
42     }
43
44
45     @Test
46     public void allocateProjectMentorNotFoundTest() throws Exception {
47         MentorDTO mentorDTO = new MentorDTO(1019, "Bethany", 2);
48         ProjectDTO projectDTO = new ProjectDTO(1, "Shoe Cart", 10012, LocalDate.of(2021, 06, 02), mentorDTO);
49
50         Mockito.when(mentorRepository.findById(projectDTO.getMentorDTO().getMentorId())).thenReturn(Optional.empty());
51         InfyInternException exception = Assertions.assertThrows(InfyInternException.class, () -> projectAllocationService.allocateProject(projectDTO));
52         Assertions.assertEquals("Service.MENTOR_NOT_FOUND", exception.getMessage());
53     }
54 }

```

Writable

Smart Insert

1:1:0

ENG 16:29
IN 23-04-2021

ASUS VivoBook