Testing of the software solution is divided into two parts: component testing and system testing.

Component testing focuses on testing individual backend components in isolation to ensure they function as expected. This is achieved using the JUnit framework, which allows developers to write and execute automated unit tests for methods or classes. Mocking frameworks, such as Mockito, are often used alongside JUnit to simulate dependencies and isolate the component being tested. For example, a service class can be tested independently of its repository or external services by mocking their behavior, ensuring that the tests are focused solely on the logic within the component.

System testing, on the other hand, evaluates the software as a whole, ensuring that all components work together as intended. This type of testing validates end-to-end workflows, user interfaces, and system interactions. To achieve this, tools like Selenium are used for automated testing of the user interface (UI) in web applications. Selenium enables the simulation of user interactions, such as clicking buttons, entering text, and navigating pages, ensuring that the frontend and backend work seamlessly together.

Component Testing

Functionality	Test Case	Input Data	Expected Results	Actual Results	Testing Procedure
Retrieving all departments	Retrieving all departments and validating employee counts	- Two departments: 1. ID = 1, Name = "Finance" 2. ID = 2, Name = "HR" - Employee count for each = 5 (mocked)	- List of two departments: "Finance" and "HR" - Employee count = 5 for each department	Passed	 Mock departmentRepository to return a list of departments. Mock userService.countUsersByDepartment to return employee counts. Validate method output.
Retrieving a department by ID	Retrieving an existing department by ID	- Department: ID = 1, Name = "Finance"	- Department returned: ID = 1, Name = "Finance"	Passed	1. Mock departmentRepository to return the department. 2. Call getDepartmentById. 3. Validate returned department.
	Handling non-existent department retrieval	- Query department with ID = 1 (does not exist)	- Throws NoSuchElementException with message: "Department not found with id 1"	Passed	1. Mock departmentRepository to return an empty result. 2. Call getDepartmentById. 3. Check exception and message.
Creating a department	Successfully creating a department	- Department name: "IT"	- Department created: ID = 1, Name = "IT"	Passed	1. Mock departmentRepository to save and return the department. 2. Call createDepartment. 3. Validate returned department.
Deleting a department	Successfully deleting a department without employees	- Department: ID = 1, Users = empty list	- Department is deleted without issues	Passed	1. Mock departmentRepository to return a department with no users. 2. Call deleteDepartment. 3. Verify that delete method was invoked.
	Failing to delete a department with employees	- Department: ID = 1, Users = List containing one user	- Throws IllegalStateException with message: "Cannot delete a department with employees"	Passed	1. Mock departmentRepository to return a department with users. 2. Call deleteDepartment. 3. Check exception and message.

Functionality	Test Case	Input Data	Expected Results	Actual Results	Testing Procedure
Updating a department name	Successfully updating a department's name	Department: ID = 1, Current Name = "Old Name" - New Name: "New Name"	- Department updated: Name = "New Name"	Passed	 Mock departmentRepository to find and save the department. Call updateDepartmentName. Validate returned department.

✓ ✓ DepartmentServiceTest (backend)	1 sec 336 ms
testDeleteDepartmentWithEmployees()	1 sec 271 ms
testDeleteDepartmentWithoutEmployees()	17 ms
testGetDepartmentById_NotFound()	4 ms
testGetDepartmentById_Success()	5 ms
testUpdateDepartmentName()	17 ms
testGetAllDepartments()	19 ms
✓ testCreateDepartment()	3 ms

System Testing

Approve trip request

- 1. Inputs: -
- 2. Testing steps:
 - a) Open the application
 - b) Click "Department approval requests"
 - c) Click "Review request"
 - d) Click "Approve"
- 3. Expected values:

The user is redirected to the main page

Popup "Trip request {id} approved successfully"

Create new trip request

1. Inputs:

Destination = "Ljubljana, Slovenia"

Duration = 2025-02-04 04:07:00, 2025-02-28 00:00:00

Purpose = "Trip"

- 2. Testing steps:
 - a) Open the application
 - b) Click "Create trip request"

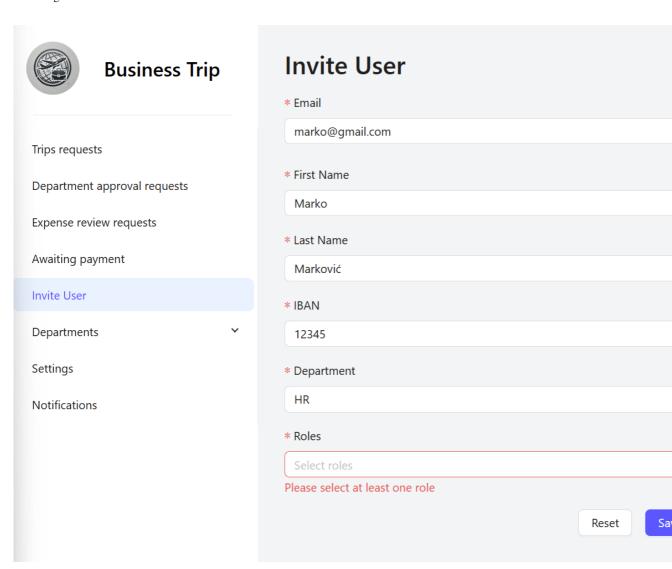
c) Input data d) Click "Save" 3. Expected values: The user is redirected to the main page Popup "Trip created successfully" Change department 1. Inputs: -2. Testing steps: a) Open the application b) Click "Departments" c) Click "Accounting" d) Click "Edit" e) Expand roles f) Click "DepartmentHead" g) Collapse roles h) Click "Save" 3. Expected values: The user is redirected to the departments page Popup "User updated successfully" Invite user with an empty role 1. Inputs: Email = "marko@gmail.com" First Name = "Marko" Last Name = "Marković" IBAN = 12345 Department = "HR" Roles = NULL 2. Testing steps: a) Open the application b) Click "Invite User"

Warning "Please select at least one role"

c) Input data

d) Click Save

3. Expected values:



Change countries daily wage

1. Inputs:

Wage = 55

- 2. Testing steps:
 - a) Click "Settings"
 - b) Expand "ASIA"
 - c) Click "AFGHANISTAN"
 - d) Input data
 - e) Click "Save"
- 3. Expected Values:

Popup "Settings updated successfully"

Project: Selenium_tests

