Software Test Documentation: Test Results

for

A Software for Summer Internship

Prepared by Ali Fırat Özdemir, Mert Damburacı,

Efekan Uysal, Umutcan Çelik

Middle East Technical University Northern Cyprus Campus

Computer Engineering

Supervised by İdil Candan

25.05.2025

Contents

1 Introduction

This document presents the test results of the Internship Management Software developed during our senior project. The main purpose of the testing process was to ensure the functionality, performance, and usability of the system through different types of testing methods, including unit testing, integration testing, system testing, performance testing, and user testing. All planned test types were successfully conducted. Unit tests were applied to service classes using JUnit 5 and Mockito to verify that individual methods performed correctly. Integration tests checked the cooperation between related components, while system testing ensured that the entire system worked as expected under real-life conditions. Performance tests using Apache JMeter showed the system's ability to handle concurrent requests. Lastly, user testing was carried out with students, a coordinator, and student affairs staff to evaluate usability and satisfaction.

We successfully completed all the tests defined in our test plan, and the results confirmed that the system meets the expected requirements.

2 Test Procedures, Cases and Results

2.1 Unit Testing

2.1.1 Test Procedure

In our project, we performed unit testing on service classes to check if the methods work correctly. JUnit 5 and Mockito were used. With Mockito, we could mock repository classes, so we didn't need to use a real database. We mainly tested methods that save, get, or delete data. For example, in the test for `saveAnnouncement()`, we checked if the method returns the correct announcement object after saving. We used `@Mock`, `@InjectMocks`, and @ExtendWith(MockitoExtension.class)` to prepare the testing environment. All unit tests were written by the backend team and executed successfully

2.1.2 Test cases

ID	***Description***	***Steps***	***Test Data***	***Pre-condition***	***Expected Output***	***Passed /Failed***
TC1	Test saveAnnouncement() method	1. Create a mock Announcement object 2. Mock repository save method 3. Call saveAnnouncement() 4. Check returned result	Content: "Important Message"	Repository mocked with Mockito	Returned Announcement has correct ID and content	Passed

TC2	Test getApprovedTraineeInformationForms() method	1. Mock findAll() method 2. Call the service method 3. Check list size	Two mock form objects	Repository mocked with Mockito	Returns list of 2 forms	Passed
TC3	Test updateFormStatus() when form exists	1. Mock findById() with existing form 2. Call updateFormStatus() 3. Check returned boolean and form status	Status = "Approved"	Form found in repository	Returns true, status updated	Passed
TC4	Test approveInsurance() logic	1. Mock findById() and user repo 2. Call approveInsurance() 3. Check fields and verify calls	Form with ID, username	Valid internship form in DB	Sets approval true, logs it, sends mail	Passed
TC5	Test sendVerificationEmails() method	1. Create mock branches 2. Mock findAll() 3. Call method 4. Check email service	2 mock branches	Repository mocked	Email sent for each branch	Passed
TC6	Test saveCompanyBranch() method	1. Create a mock CompanyBranch 2. Mock save and email functions 3. Call saveCompanyBranch() 4. Verify result	CompanyBranch with email and username	Repository and EmailService mocked	CompanyBranch saved and email sent	Passed
TC7	Test createEvaluationForm() method	1. Mock findById 2. Call createEvaluationForm() 3. Verify save and email methods	Working Day: 20 Performance: "Excellent" Feedback: "Well done"	Trainee form mocked	Evaluation saved and email sent	Passed

TC8	Test addForm() method	1. Create mock AcademicStaff 2. Mock academicStaffRepository.findById() 3. Call addForm() 4. Mock formRepository.save() 5. Check return	File: file.pdf Content: "form content"	AcademicStaff exists in repository	Form is saved and returned with ID 1	Passed
TC9	Test updateInitialFormStatus() for company mail	1. Mock findById() for form 2. Mock last approved form 3. Mock token creation 4. Call method	Status: "Company Approval Waiting"	Trainee form and approved form exist	Status updated, token created, email sent to company	Passed
TC10	Test applyForInternshipOffer() method	1. Mock findByUserName() 2. Mock findById() for InternshipOffer 3. Call applyForInternshipOffer() 4. Verify save method	Username: "student1" Offer ID: 101	Student and internship offer exist	Application is saved successfully in repository	Passed
TC11	Test createAllEvaluations() method	1. Mock findById() for report 2. Prepare mock DTO with 11 scores & comments 3. Call createAllEvaluations() 4. Verify saves & email	Report ID: 1 Username: "student1"	Report exists in DB	11 evaluations saved and email sent to student	Passed
TC12	Test addReport() for second report upload	1. Mock findById() for form2. Mock existing report3. Prepare DTO with file4. Call addReport()5. Verify save and email call	Username: "student1 "Form ID: 1File: report.pdf	Form status is "Approved"1 report exists	Second report saved, email sent to instructor	Passed

TC13	Test addResume() method	1. Mock findByUserName() 2. Create a mock resume file 3. Call addResume() 4. Verify resumeRepository.save() method	Username: "student1" File: "This is the resume content."	Student exists in repository	Resume is saved and returned successfully	Passed
TC14	Test createPasswordResetToken() method	1. Call createPasswordResetToken() 2. Verify token is saved	Username: "testuser"	-	Token is generated and saved with correct data	Passed

2.1.3 Test Results

- TC1: The test for the saveAnnouncement() method passed successfully. No errors occurred during the execution. The output matched the expected result.

- TC2: The test for getApprovedTraineeInformationForms() passed without errors. It returned the correct list size.

- TC3: The test for updateFormStatus() returned true and the form status was updated as expected.

- TC4: The test for approveInsurance() updated the approval status, saved the log, and triggered the mail function.

- TC5: The test for sendVerificationEmails() passed successfully. The email service was called exactly two times as expected.

- TC6: The test for the saveCompanyBranch() method passed successfully. The method saved the branch and triggered the email service without throwing any exception. The output matched the expected behavior.

- TC7: The test for the createEvaluationForm() method passed successfully. The evaluation form was saved, and the email was sent without any errors.

- TC8: The test for the addForm() method passed successfully. It saved the form using the mocked repository and returned the saved form object.

- TC9: The test for the updateInitialFormStatus() method passed successfully. The method updated the form status, created a reset token, and sent the email to the company without any errors.

- TC10: The test for the applyForInternshipOffer() method passed successfully. The student and internship offer were mocked, and the application was saved using the repository without any errors.

- TC11: The test for the createAllEvaluations() method passed successfully. The method saved 11 evaluation items and sent the email without any errors.

- TC12: The test for addReport() passed successfully. When a second report was uploaded, it saved the report and triggered an email notification to the instructor.

- TC13: The test for the addResume() method passed successfully. The method retrieved the student, created a resume object, and saved it using the mocked repository. No exceptions were thrown, and the result matched the expected behavior.

- TC14: The test for the createPasswordResetToken() method passed successfully. The method generated a valid token and saved it to the repository with correct expiration time.

2.1.4 Test Log

All unit tests were run using IntelliJ IDEA. The tests were executed with JUnit 5 and Mockito. The console output and test result screen showed whether each test passed or failed. Screenshots for each test are given below.

- TC1: Tested the saveAnnouncement() method using a mocked repository. The service method was called and the console printed the correct ID and content. Test method name: testSaveAnnouncement_ReturnsSavedAnnouncement() (see Figure 1).

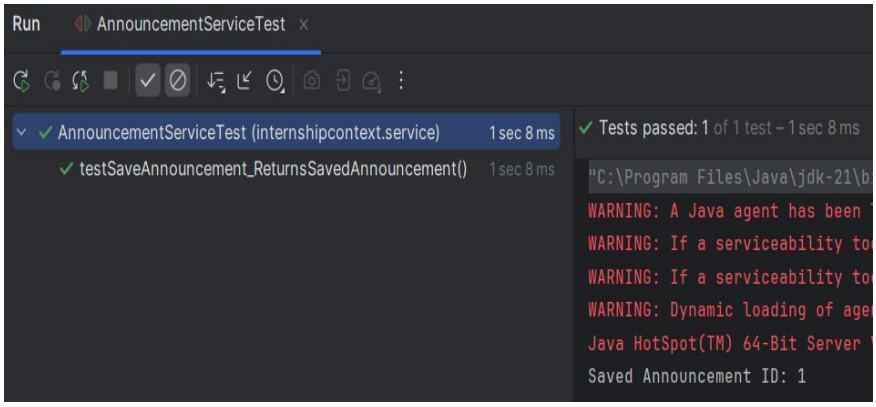


Figure 1: Unit test result for saveAnnouncement() method

The following test cases were executed and their results are shown in Figure 2:

• TC2: Tested getApprovedTraineeInformationForms() with a mocked repository. Confirmed that a list with 2 items was returned.

• TC3: Called updateFormStatus() after mocking findById(). Checked that it returned true and correctly updated the form status.

• TC4: Called approveInsurance() on a mocked internship. Verified that the approval flag was set, the log was saved, and an email was sent to the user.

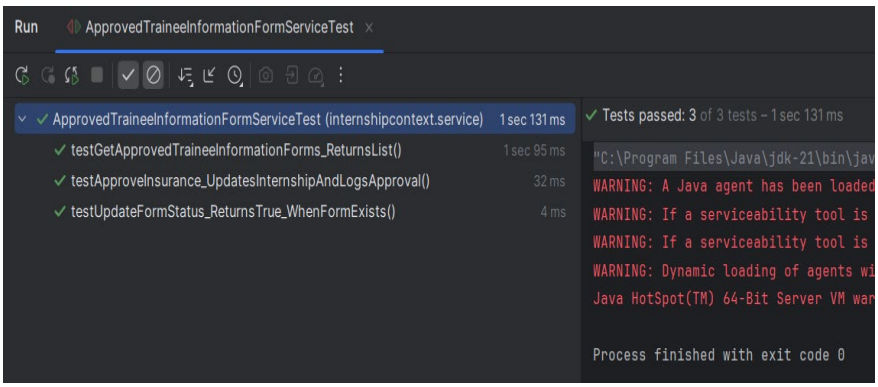


Figure 2: Test Results – ApprovedTraineeInformationFormService

- TC5: Tested sendVerificationEmails() method with two mock company branches. Verified that emailService.sendEmail() was called twice with correct subject and content (see Figure 3).

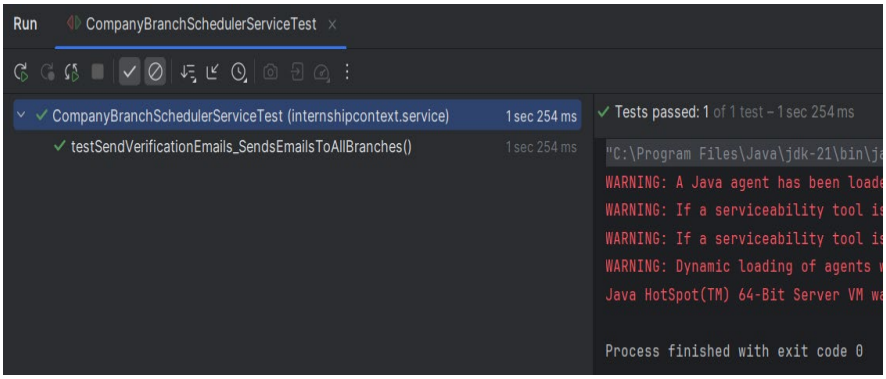


Figure 3: Test Result – CompanyBranchSchedulerService

- TC6: Tested the saveCompanyBranch() method. It saved a mock CompanyBranch and called the email sending function. Verified that the method worked as expected and did not throw an exception. Test method name: testSaveCompanyBranch_SavesAndSendsEmail() (see Figure 4).

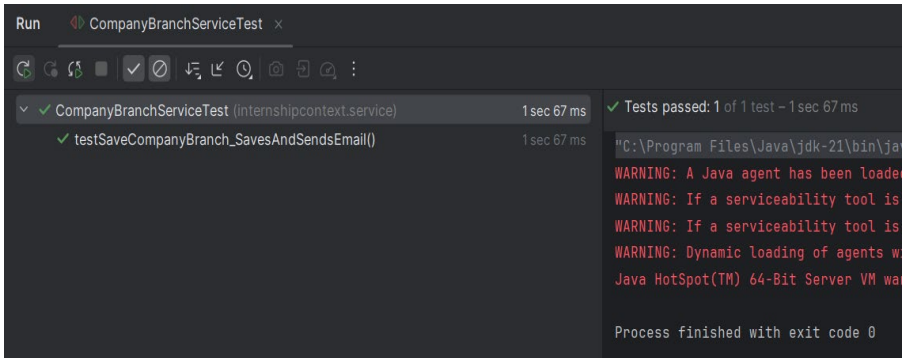


Figure 4: Result of saveCompanyBranch() unit test

- TC7: Tested the createEvaluationForm() method with a mocked trainee form and email service. The method saved the evaluation and triggered the email sending. Test method name: testCreateEvaluationForm_SavesEvaluationAndSendsEmail() (see Figure 5).

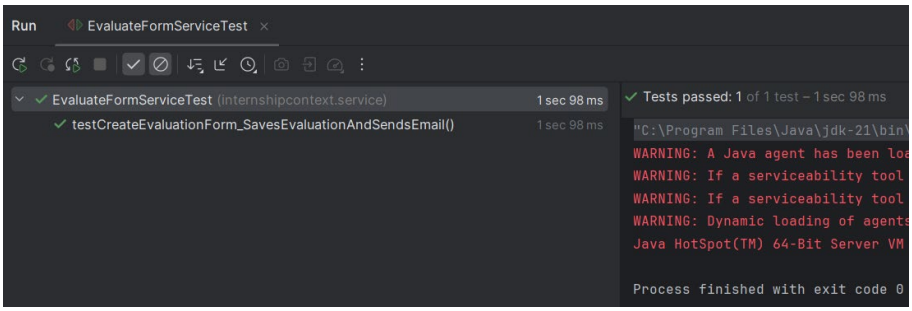


Figure 5: Unit test result for createEvaluationForm() method

- TC8**:** Tested the addForm() method using a mocked AcademicStaff and repository. Confirmed that the form was saved and the returned object had a valid ID. Test method name: testAddForm_SavesAndReturnsForm() (see Figure 6).

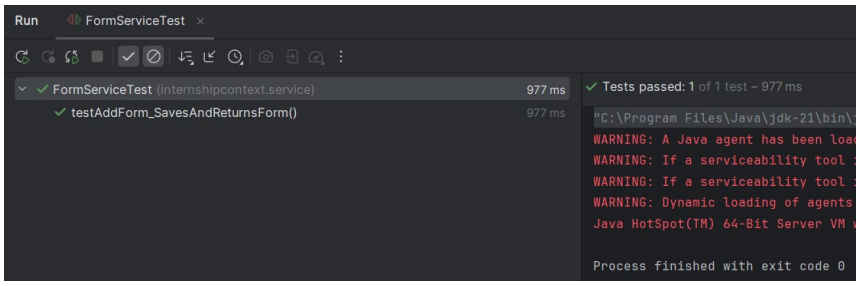


Figure 6: Unit test result for addForm() method

- TC9: Tested the updateInitialFormStatus() method with a mocked trainee form and company branch. Verified that the method updated the status, generated the token, and triggered the email service correctly. Test method name: testUpdateInitialFormStatus_SendsCompanyEmail() (see Figure 7).

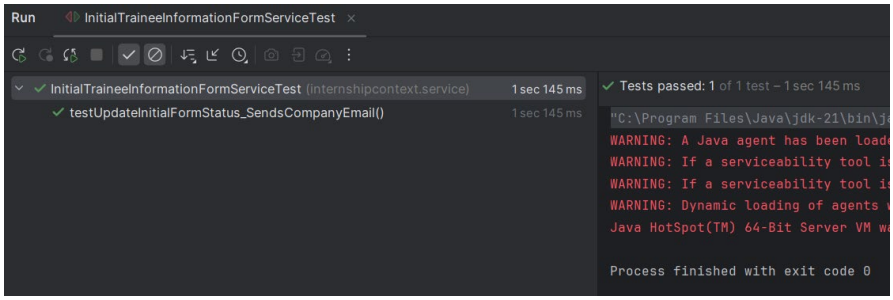


Figure 7: Unit test result for updateInitialFormStatus() method

- TC10: Tested applyForInternshipOffer() with mocked student and internship offer. Verified that the repository's save() method was called correctly. Test method name: testApplyForInternshipOffer_SavesApplication() (see Figure 8).

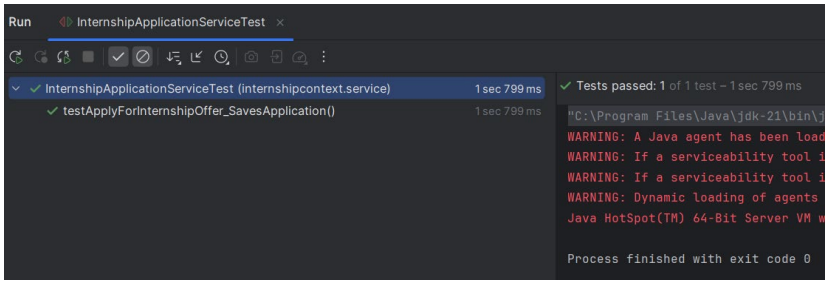


Figure 8: Unit test result for applyForInternshipOffer() method

- TC11: Tested the createAllEvaluations() method using a mocked report and evaluation repository. Verified that 11 evaluation entries were saved and the email service was called correctly. Test method name: testCreateAllEvaluations_SavesAndSendsEmail() (see Figure 9).

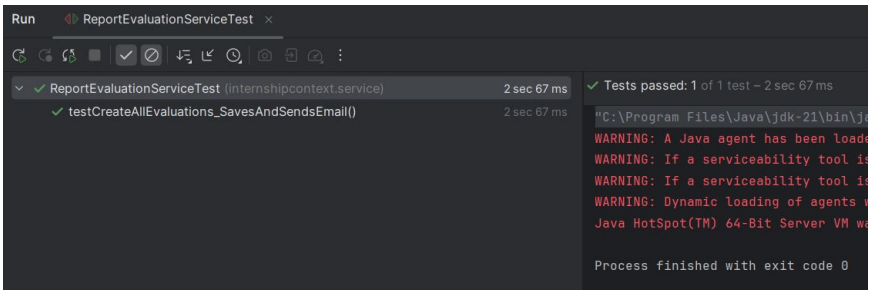


Figure 9: Unit test result for createAllEvaluations() method

- TC_{12}: Tested the addReport() method with mocked dependencies. Simulated a second report upload by a student. Verified that the report was saved and the instructor received a notification email. Test method name: testAddReport_SavesReportAndSendsEmailOnSecondUpload() (see Figure 10).

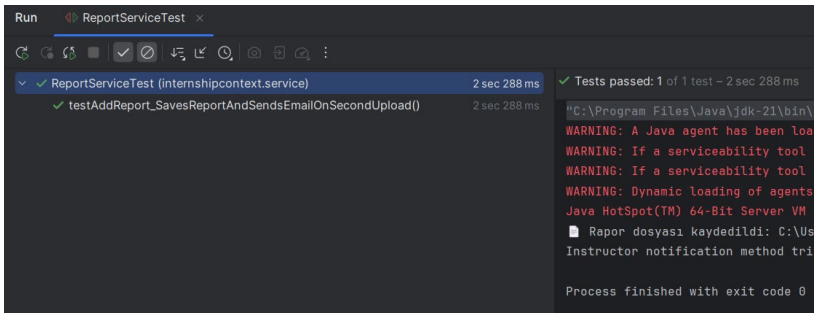


Figure 10: Unit test result for addReport() method

- TC13: Tested the addResume() method with a mocked student repository. Verified that the method fetched the student successfully, created the resume object, and saved it. Test method name: testAddResume_SuccessfullySavesResume() (see Figure 11).

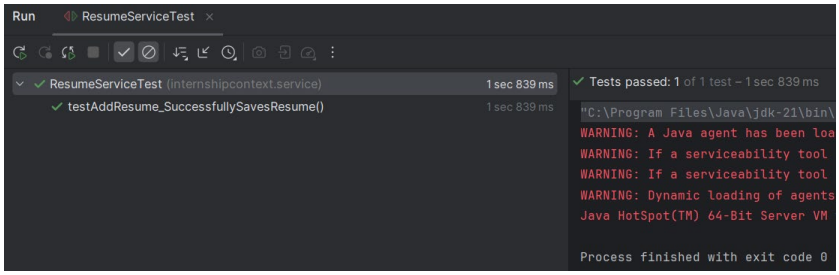


Figure 11: Unit test result for addResume() method

- TC14: Tested the createPasswordResetToken() method with a mock username. Verified that the generated token was saved using the mocked repository. Test method name: testCreatePasswordResetToken_SavesToken() (see Figure 12).

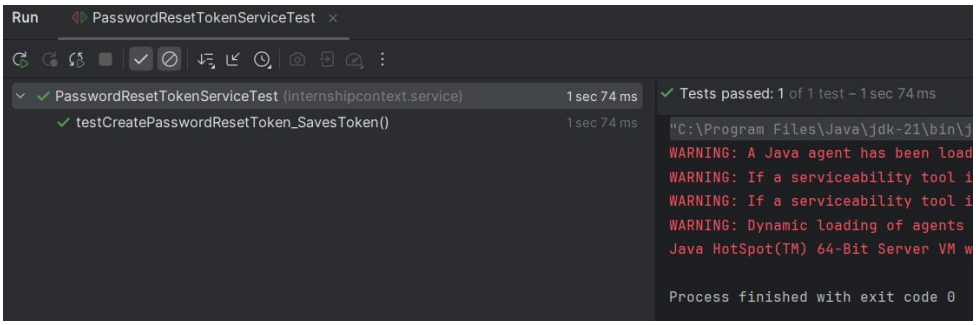


Figure 12: Unit test result for createPasswordResetToken() method

2.2 Integration Testing

2.2.1 Test Procedure

Unit and integration testing were conducted with a strong focus on ensuring the seamless interaction between frontend and backend components. The testing process primarily emphasized end-to-end (E2E) and integration scenarios to validate the system's overall behavior.

All tests were developed and executed within the IntelliJ IDE, using Playwright as the primary testing framework. Because we had tens of cases that needs integration between frontend and backend, we selected a total of 20 test cases were implemented in spec.ts files, covering key workflows and user interactions. These test cases simulated real user behavior and verified the correct functioning of various application modules in an integrated environment. Tests of more complicated and combined actions are performed within the system testing part.

Playwright provided robust capabilities for automating browser-based tests, enabling efficient validation of UI elements, API responses, and client-server communication. Each test case was carefully and specially crafted to ensure that both individual components and their interactions met the expected outcomes for each unique use case.

Additionally, during the testing process, it was also tested whether some functions that needed to send e-mails sent e-mail notifications.

Efekan performed all the tests in the integration testing.

2.2.2 Test cases

ID	Description	Steps	Test Data	Pre-condition	Expected Output	Passed/Failed
TC1	Test login integration: Successful login for different type of users	1) Navigates to localhost:4200 before test 2) Fills username/password fields and clicks login button for each test user 3) Frontend sends credentials to backend (localhost:8080/auth/verify) 4) Validates if it is successful login by checking if the redirect URL matches user type.	username: 'cidil', password: 'cidil2', username: 'e258546', password: 'efekan1', username: 'eever', password: 'eever1', username: 's_affairs', password: 'student_affairs1', username: 'Google-Branch', password: 'Google-Branch1',	Frontend and Backend are both should be running.	Url to change as http://localhost:4200${user.expectedRoute} Where expected route is different for every user for example: for student /student and for instructor /instructor etc.	Passed
TC2	Test login integration: Invalid credentials show error message	1) Navigates to localhost:4200 before test 2) Fills username/password fields and clicks login button for invalid credentials. 3) Frontend sends credentials to backend (localhost:8080/auth/verify) 4) Verifies error message displays	Username: 'invaliduser' Password: wrongpassword'	Frontend and Backend are both should be running.	Error message that includes 'You have entered wrong username or password' message	Passed
TC3	Test login integration: Empty form submission shows validation errors	1) Navigates to localhost:4200 before test 2) Frontend sends null credentials to backend (localhost:8080/auth/verify) 3) Verifies error message displays	Username: Password:	Frontend and Backend are both should be running.	Error message that includes 'You have entered wrong username or password' message	Passed
TC4	Test login integration: Company login link redirects and allows company login	1) Navigates to localhost:4200/company-login before test 2) Fills username/password fields and clicks login button 3) Frontend sends credentials to backend (localhost:8080/auth/verify) 4) Validates if it is successful login by checking if the redirect URL matches user type.	Username: 'Google-Branch', Password: 'Google-Branch1',	Frontend and Backend are both should be running.	Url to change as 'http://localhost:4200/company-branch/my-offers'	Passed

TC5	*Announcement loading*	*1) Navigates to localhost:4200/company-login before test* *2) Fills username/password fields and clicks login button* *3) Frontend sends get request to backend (localhost:8080/api/announcements)* *4) Validates by checking if the announcements were fetched and loaded correctly*	*None*	Frontend and Backend are both should be running.	*List of announcements*	*Passed*
TC6	*Announcement adding*	*1) Navigates to localhost:4200/login before test* *2) Fills username/password fields and clicks login button* *3) Frontend sends post to backend (localhost:8080/api/announcements)* *4) Validates by checking if the new announcement was added correctly.*	*Title:Test Announcement3 ' + Date.now()* *Content: 'This is a test announcement content'*	*Frontend and Backend are both should be running.*	*Announcement fetched and appeared in the list*	*Passed*
TC7	*Loading existing forms*	*1) Logs in as a coordinator or student* *2) Navigates to localhost:4200/coordinator/forms* *3) Sends get request to backend (localhost:8080/api/forms)* *4) Verifies that forms table is correctly filled*	*None*	*Frontend and Backend are both should be running.*	*Forms table is not empty and correctly filled after fetching data from the backend.*	*Passed*
TC8	*Adding form*	*1) Logs in as a coordinator* *2) Navigates to (localhost:4200/coordinator/forms)* *3) Sends cv.pdf with a post request to backend (localhost:8080/api/forms)* *4) Verifies that number of forms in the table increased by one.*	*cv2.pdf*	*Frontend and Backend are both should be running.*	*Number of forms in the table increased by one.*	*Passed*
TC9	*Adding non-Pdf form*	*1) Logs in as a coordinator* *2) Navigates to (localhost:4200/coordinator/forms)* *3) Sends a non pdf input with post request to backend (localhost:8080/api/forms)*	*testfile.txt*	*Frontend and Backend are both should be running.*	*"Only PDF files are allowed" message is displayed and the number of forms hasn't changed*	*Passed*

		4) Verifies that correct error message is displayed and the number of forms hasn't changed				
TC10	*Assigning an instructor to a student*	*1) Logs in as a coordinator* *2) Navigates to (localhost:4200/coordinator/assign-instructors)* *3) Verifies that forms loaded correctly from the backend* *4) For the first internship selects the second instructor in the menu and assigns them.* *5) Verifies the assignment.*	*None*	*Frontend and Backend are both should be running.*	*Success message is showed up and instructor assigned successfully is visible after an automatic table reload.*	*Passed*
TC11	*Update a deadline*	*1) Logs in as a coordinator* *2) Navigates to (localhost:4200/coordinator/set-deadlines)* *3) Sets the report deadline as 2024-12-31.* *4) Submits deadlines to backend (localhost:8080/api/deadline/add)* *5) Verifies the change in deadline.*	*2024-12-31*	*Frontend and Backend are both should be running.*	*Report Deadline is updated as 2024-12-31 in deadlines page.*	*Passed*
TC12	*Reset a deadline*	*1) Logs in as a coordinator* *2) Navigates to (localhost:4200/coordinator/set-deadlines)* *3) Removes the report deadline and send delete http request to backend (localhost:8080/api/deadline/delete/internship-deadline)* *5) Verifies the change in deadline.*	*None*	*Frontend and Backend are both should be running.*	*Report Deadline is changed as Not Set in deadline page.*	*Passed*
TC13	*Trainee form evaluation (coordinator): Load trainee forms*	*1) Logs in as a coordinator* *2) Navigates to (localhost:4200/coordinator/evaluate-forms)* *3) Fetches all the forms for coordinator. (localhost:8080/api/internships)* *4) Shows all forms in a table.*	*None*	*Frontend and Backend are both should be running.*	*Trainee Information forms table is not empty and correctly filled after fetching data from the backend.*	*Passed*

TC14	*Trainee form evaluation (coordinator): Approve trainee forms*	*1) Logs in as a coordinator* *2) Navigates to (localhost:4200/coordinator/evaluate-forms)* *3) Fetches all the forms for coordinator. (localhost:8080/api/internships)* *4) Finds a non-approved form in the table and approve it.* *5) Sends the id of the form we want to approve to backend with a post http request. (localhost:8080/api/coordinatorApprove/approve/${id}?username=${encodeURIComponent(username)})* *6) Verifies the success notification has popped up and form's coordinator approval changed as approved in frontend.*	*None*	*Frontend and Backend are both should be running.*	*Success notification has popped up and form's coordinator approval changed as approved.*	*Passed*
TC15	*Instructor View: browsing reports*	*1) Logs in as an instructor* *2) Navigates to (localhost:4200/instructor/evaluate-assigned-reports)* *3) Fetches all the forms for that instructor.* *4) Checks the reports of the 5th report in the table by fetching it from the backend. (localhost:8080/reports/trainee/ + formId+ /reports)* *5) Verifies that the report information is fetched and displayed correctly.*	*5th form*	*Frontend and Backend are both should be running.*	*Report information is fetched and displayed correctly.*	*Passed*
TC16	*Instructor View: Viewing Company Evaluation*	*1) Logs in as an instructor* *2) Navigates to (localhost:4200/instructor/evaluate-assigned-reports)* *3) Fetches all the forms for that instructor.* *4) Finds a form that has its company evaluation filled.* *5) Verifies that the company evaluation is loaded and being displayed.*	*None*	*Frontend and Backend are both should be running.*	*Company evaluation is loaded and being displayed.*	*Passed*
TC17	*Student Affairs Insurance Approval*	*1) Logs in as student affairs* *2) Navigates to (localhost:4200/student-affairs/approved-internships)*	*None*	*Frontend and Backend are both*	*Approved text is displayed, and it persisted after*	*Failed*

		3) Fetches all the forms for student affairs. *4) Finds a form that hasn't approved.* *5) Approves the form by sending post http request to backend. (localhost:8080/api/studentAffairs/approveInsurance)* *6) Verifies that it is displayed as approved and it persists after reloading the page and fetching data again from the backend.*		*should be running.*	*reloading the page.*	
TC18	*Uploading Resume*	*1) Logs in as a student* *2) Navigates to (localhost:4200/student /my-resume)* *3) Clicks to browse and upload cv2.pdf.* *4) Confirms upload and reload the page to verifies that resume is accessible from the backend correctly.*	*cv2.pdf*	*Frontend and Backend are both should be running.*	*Uploaded resume is displayed, and it persisted after reloading the page.*	*Passed*
TC19	*Approving Internship Application*	*1) Logs in as a company-branch* *2) Navigates to (localhost:4200/company-branch/evaluate-intern-student)* *3) Approves a non-approved form by sending a post request to backend. (localhost:8080/api/traineeFormCompany/approveInternship)* *4) Confirms that "Internship successfully approved!" message is shown and status changed to Approved.*	*None*	*Frontend and Backend are both should be running.*	*"Internship successfully approved!" message is shown and status changed to Approved.*	*Passed*
TC20	*Assigning Supervisor*	*1) Logs in as a company-branch* *2) Navigates to (localhost:4200/company-branch/evaluate-intern-student)* *3) Clicks on Set Instructor button for 2nd form in the list.*	*Supervisor Name: "Ka Ryo"* *Supervisor Surname: "Ten"*	*Frontend and Backend are both should be running.*	*Supervisor is changed as Ka Ryo Ten in the table.*	*Passed*

		4) Writes Ka Ryo Ten to menu and submits it to backend. (localhost:8080/api/internships/updateSupervisor) *5) After window is being closed, verifies that supervisor is changed as Ka Ryo Ten in the table.*				

2.2.3 Test Results

Since we usually test the integration of each feature we add to our system, only one of our test scenarios failed.

- TC1: All test users logged in successfully, each being redirected to the correct page based on their user role.
- TC2: The system correctly handled login failure and displayed the appropriate error message.
- TC3: The system displayed successfully validation errors when trying to submit an empty form.
- TC4: Successfully redirected to the correct company login dashboard after successful login as a company branch.
- TC5: Fetched announcements successfully, confirming GET request functionality.
- TC6: Added a new announcement and confirmed its appearance in the announcement list. Additionally confirmed that email notification is being sent to selected user types.
- TC7: Verified forms were correctly fetched and displayed for coordinator/student.
- TC8: PDF file successfully uploaded, and form count incremented, verifying POST functionality.
- TC9: System correctly prevented non-pdf form file upload and showed the appropriate error message.
- TC10: Instructor assignment was successful and reflected immediately in the UI with a success message.
- TC11: Deadline was set to 2024-12-31 and is being displayed correctly verifying POST functionality.
- TC12: Deadline was successfully removed and shown as "Not Set". This proved that our DELETE request functions as intended.
- TC13: Trainee Information Forms for coordinator were loaded and populated in the table without an issue.
- TC14: As a coordinator system successfully approved a form and updated the status of the form. The email notification sent to the student was also approved successfully.
- TC15: Instructor accessed details of reports uploaded by students correctly.
- TC16: Instructors can view the company evaluations of students successfully.
- TC17: The test case failed as the insurance status was not changed to "approved". However, after a page reload the status changes correctly. Although we don't know the issue yet it is possible that this indicates a possible issue in backend where backend returns an error message to

frontend even after successfully changing the status. It was also noted that during this process, an email notification was successfully sent to the student stating that their insurance had been approved.

- TC18: Resume was uploaded and displayed correctly. The preservation of the resume proves that the POST request works correctly.
- TC19: Approval of internship by company was successful, status updated, and success message appeared.
- TC20: Supervisor details updated correctly and reflected in UI which shows a correctly working POST request functionality.

2.2.4 Test Log

All integration and E2E tests were run using IntelliJ IDEA. The tests were automatically executed with Playwright. The console output and test result screen showed whether each test passed or failed. Screenshots for each test are given below.

TC1-TC4:

Tested login integration by logging in as different type of users and trying to login with invalid credentials. The system gave the expected output in all cases. To check the tests performed, please check the figure below.

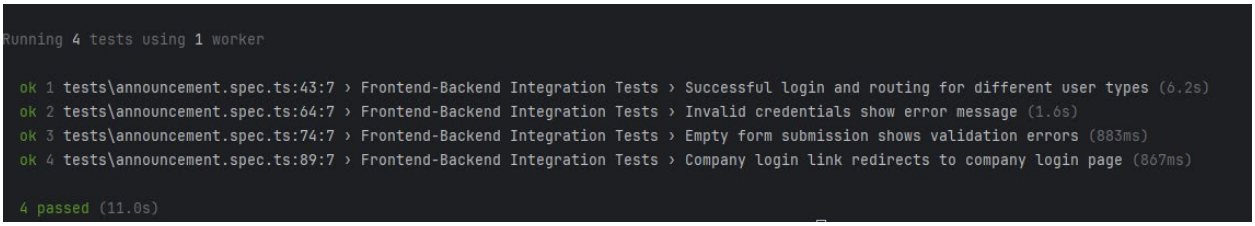


Figure 13: Login test

TC5-TC6:

Tested announcement integration with 3 cases. First test loads the existing announcements correctly from the backend. Second case verifies that frontend shows the announcement form when add button is clicked and third case confirms that the form that is shown actually sends http request that adds a new announcement. The system gave the expected output in all cases. You can check the figure below to see test results.

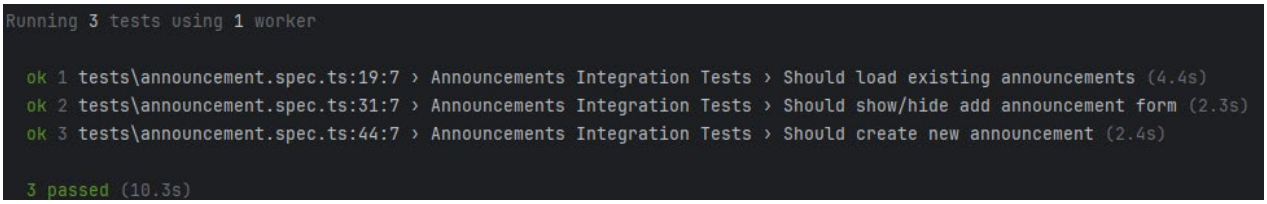


Figure 14: Announcement integration test

TC7-TC9:

Tested form integration with 4 cases. First three cases check TC7, TC8 and TC9 and the last case checks if the page reloads after uploading. It is not considered a case because it is not directly related with integration. As you can see the figure below the system gave the expected output in all cases.

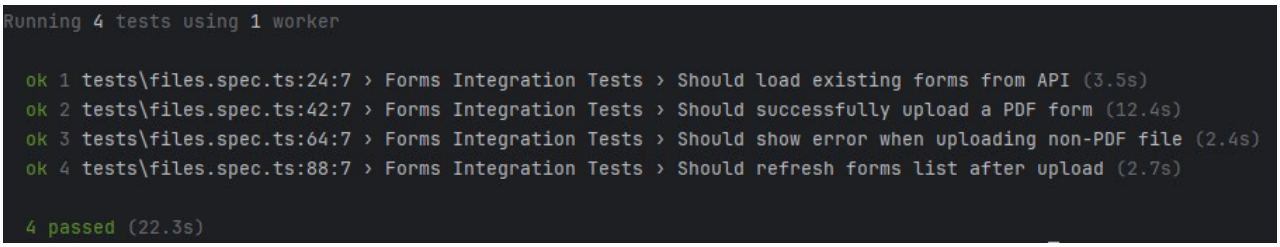


Figure 15: Login test

TC10:

Tested instructor assignment integration. Verified that instructor assigned as selected and success message showed up. Check the figure below.

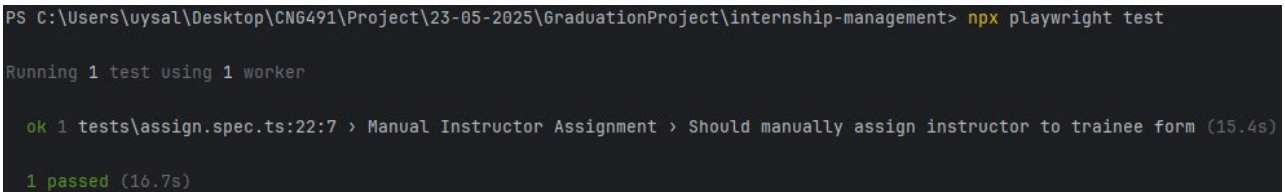


Figure 16: Instructor Assignment test

TC11-TC12:

Tested deadline setting integration. Verified that deadlines can be set and reset correctly as selected in the frontend. You can check the figure below to see the test performed.

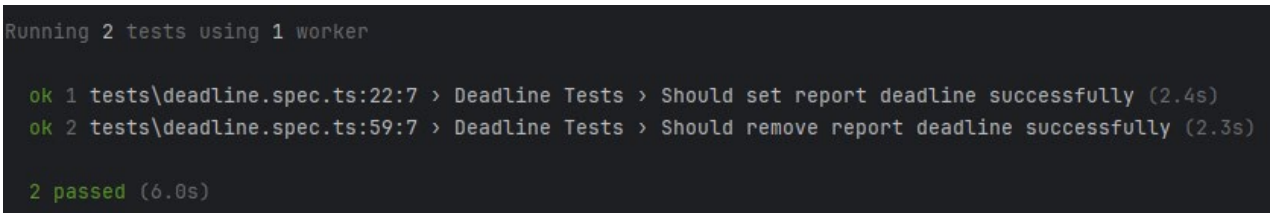


Figure 17: Deadline test

T13-T14:

Tested that the coordinator can view and approve the internship forms of the students in the frontend and that this works in an integrated way with the backend. You can see the figure below to see the console screen of the tests.

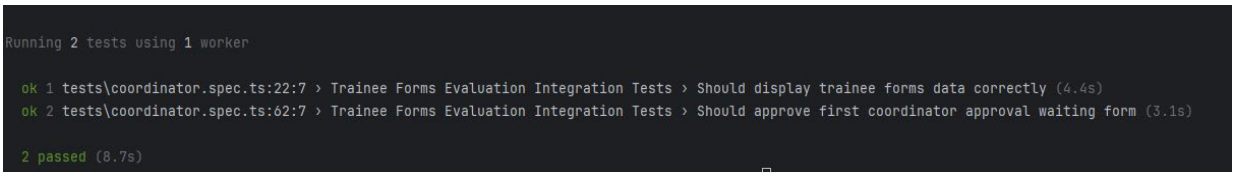


Figure 18: Coordinator Form Approval test

T15-T16:

Tested that instructors can view reports of the students and company evaluation of their internship in the frontend. You can see the tests performed below.

Figure 19: Instructor Report View test

T17:

Tested student affairs to approve the insurance of an internship and further checked if the change persists after reloading the page. After running the tests shown below, the approvement is being displayed in the frontend only after reloading the page.

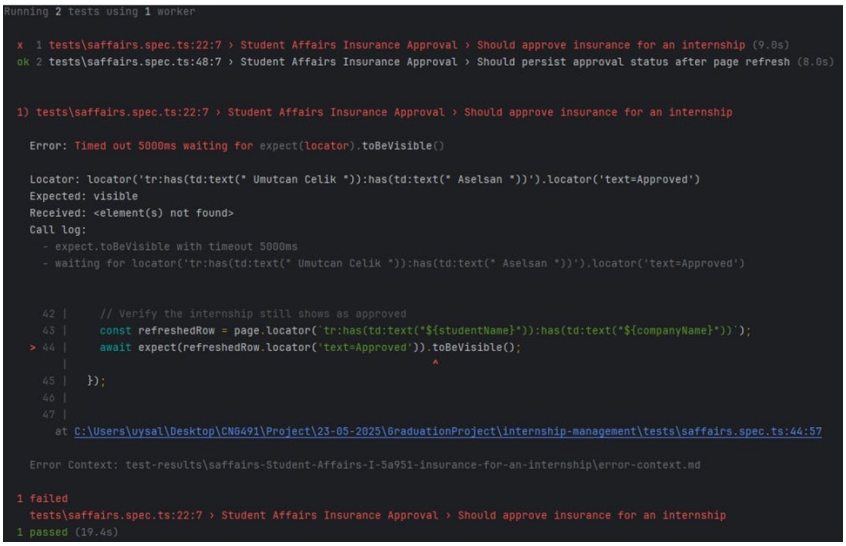


Figure 20: Student Affairs Insurance Approval test

T18:

Tested student resume upload to the system and confirmed that the uploaded resume can be fetched again from the backend after reloading the page. Below is the test we performed.

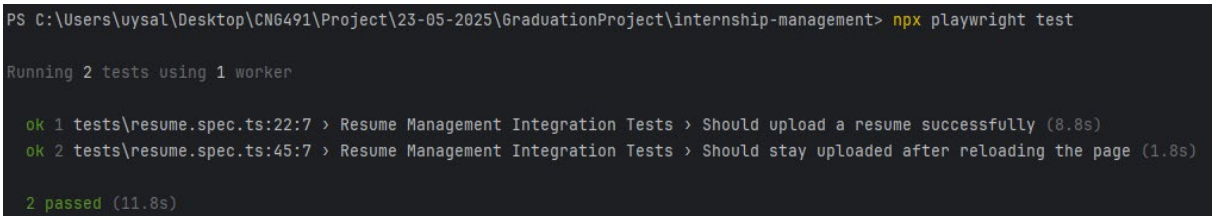


Figure 21: Resume Upload test

T19-T20:

Tested company branch to approve student internships and assigning supervisors to these internships. Verified that approvements and supervisor assignments are done correctly. You can examine the following figure to see the tests.

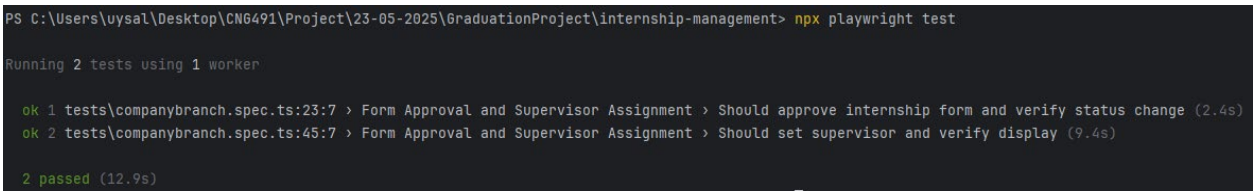


Figure 22: Company Internship Approval test

2.3 System Testing

2.3.1 Test Procedure

System testing was performed in real-time using a scenario-based approach to validate the end-to-end workflow of the system. The primary objective was to ensure that the integrated components of the system function correctly together and meet the functional requirements. The test procedure followed a structured approach based on the system's actual usage scenarios that simulates real-world usages and ensuring comprehensive coverage of all critical functionalities. All system tests were performed by Efekan.

2.3.2 Test Scenarios

Scenario 1:

a) As the new internship period approaches, the internship coordinator determines the deadlines for filling out the internship application form and submitting the report.

b) A student who finds an internship enters the trainee information form into the system.

c) The student can make changes on this form before it is approved.

d) The internship application entered by the student into the system is approved by the internship coordinator if deemed appropriate. The student gets an email notification about this.

e) The internship application approved by the coordinator is now visible to the company. The company approves the student's internship from the system. The student is notified by an e-mail.

f) The company assigns a supervisor to the student.

g) Student affairs approves the insurance. And can export to excel. The student is notified by an e-mail.

h) The student uploads the internship report to the system after the internship is completed.

i) The company evaluates the student's internship. The student is notified by an e-mail.

j) The coordinator assigns a faculty member to evaluate the student.

k) The assigned instructor sees the reports uploaded by the student and the evaluation form uploaded by the company through the system.

l) Instructor grades the last uploaded report and requests re-submission. The student is notified by an e-mail.

m) The student submits the report again.

n) Instructor approves by giving grade again. The student is notified by an e-mail.

o) The student and the coordinator can see this grade.

Scenario 2:

a) The student uploads his/her resume to the system.

b) Student uses browse Internships page to see the companies where previous students have done internship. They can also see which internships match with their skills in their resume. Student applies to a company here.

c) The company can see and approve this application and the student's resumé on the Applicants page.

Scenario 3:

a) The student uploads his/her resume to the system.

b) The company creates an internship advertisement on the open internships page.

c) Student uses Internships Offers page to applies to an internship advertisement.

d) The company can see the applicants from the open internships page and reject the student.

Scenario 4:

a) Coordinator adds an announcement. All selected user types get notified by an e-mail.
b) Students can view the announcement from Announcements page.

Scenario 5:

a) Coordinator adds a summer training form.
b) Students can view and download the summer training form.

2.3.3 Test Results

Scenario 1: Passed

Scenario 1 is a scenario that fictionalizes the main purpose of the internship system, and the current system was able to successfully fulfil all steps. The internship coordinator was able to set the application and report deadlines, and students could enter and edit their trainee information. Approved internship applications triggered email notifications to students and became visible to companies, who could then approve them and assign supervisors. Student affairs successfully processed and approved insurance and exported records to Excel, with appropriate notifications sent to students. After completing the internship, students uploaded their reports, and companies submitted evaluations, both accessible to the assigned instructors. Instructors could review the documents, assign grades, and request revisions with students notified via email. Final grades were submitted by instructors and became visible to both the student and the coordinator.

However, during testing, we noticed a small usability issue: students were unable to view the revision decision directly within the system interface. Although this doesn't affect the functionality and even though the students were notified by an email, the lack of visibility in the system can create confusion. This issue will be addressed to ensure a better user experience.

Scenario 2: Passed

The student was able to upload their resume to the system without issues. Upon navigating to the "Browse Internships" page, the student could view a list of companies where previous students had interned, along with personalized internship suggestions that matched the skills listed in their resume. The student was able to apply to a selected company directly through the system. On the company side, the application appeared correctly on the "Applicants" page, and the company was able to view and approve the student's application and resume. All functionalities worked as expected.

Scenario 3: Passed

As we already tested in second scenario the system successfully allows the student to upload their resume. The company is able to create an internship advertisement, which is then displayed on the open internships page, indicating that advertisement creation feature is functional. The student is able to view and apply to the internship via the Internships Offers page, demonstrating that application is correctly implemented. Finally, the company can view the list of applicants and reject a student through the open internships page, confirming that rejection functionality operates as expected.

Scenario 4: Passed

When the coordinator adds an announcement, all selected user types receive a notification via email. Additionally, students can access and view the newly added announcement correctly on the Announcements page which confirms that both the notification and display functionalities are working as intended.

Scenario 5: Passed

After the coordinator adds a summer training form, students can access and download the newly added summer training form correctly on the Forms page. This shows that all functionalities are working as expected.

2.3.4 Test Log

Scenario 1:

a) Coordinator sets the deadlines for filling out the internship application form and submitting the report.

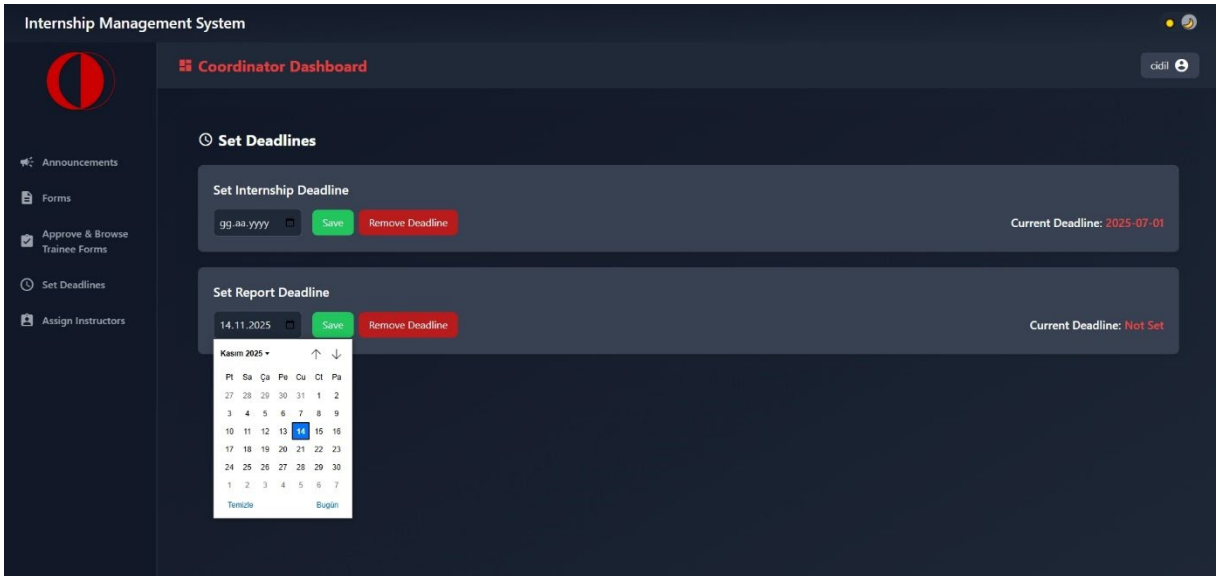


Figure 23: Coordinator Set Deadline page

b) A student who finds an internship enters the trainee information form into the system.

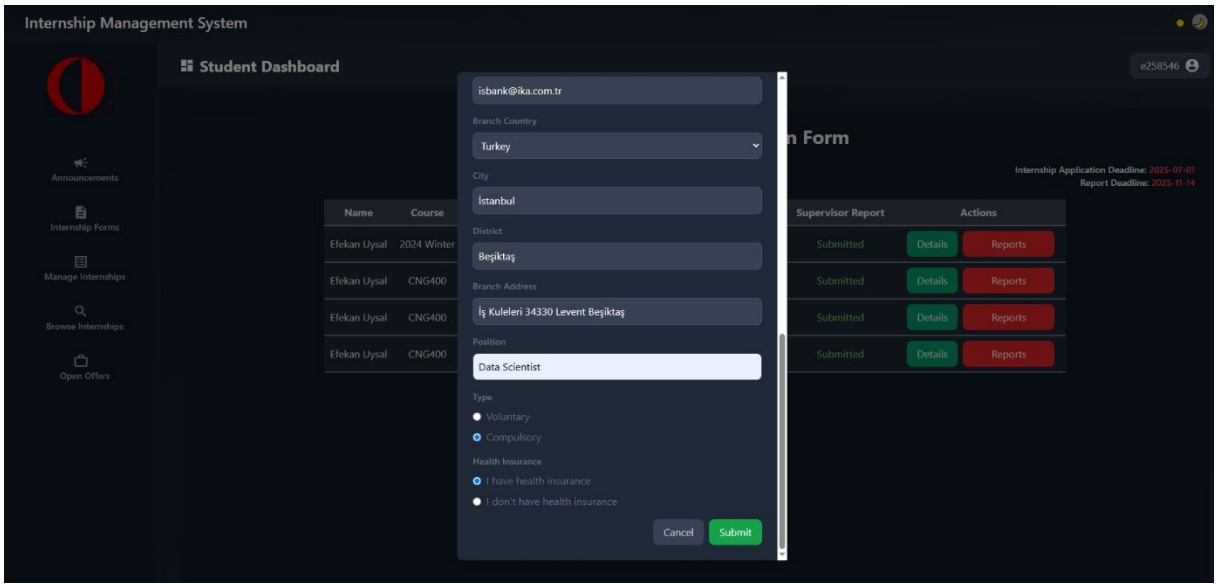


Figure 24: Student's Trainee Form page

c) The student can make changes on this form before it is approved by the company.

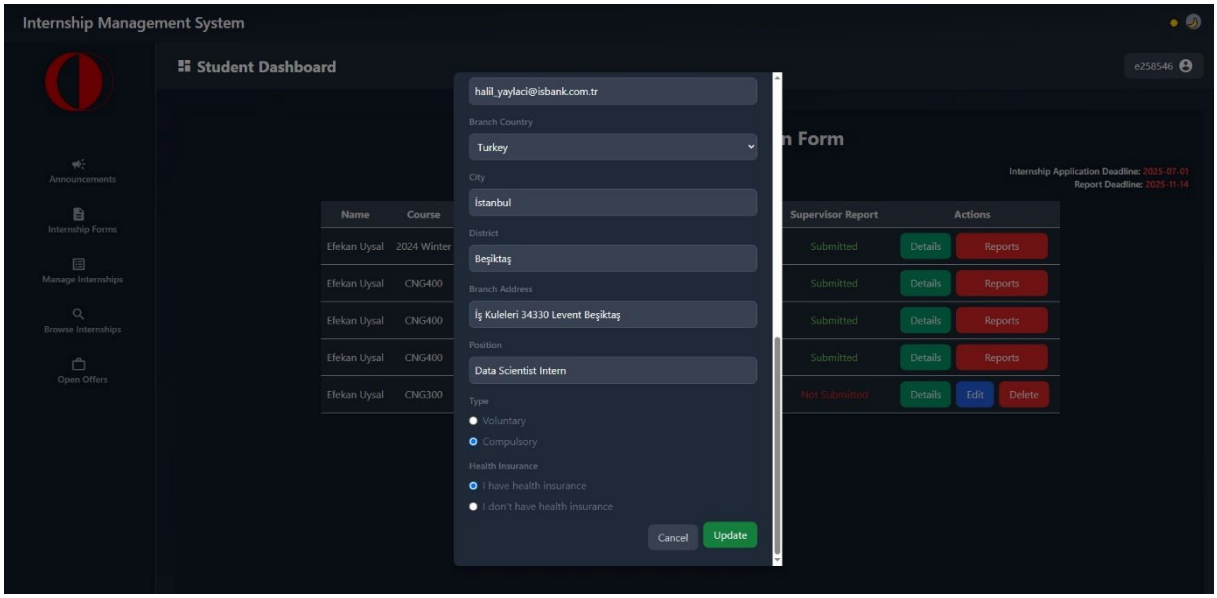


Figure 25: Student Form edit

d) The internship application is approved by the internship coordinator. Student gets an email notification.

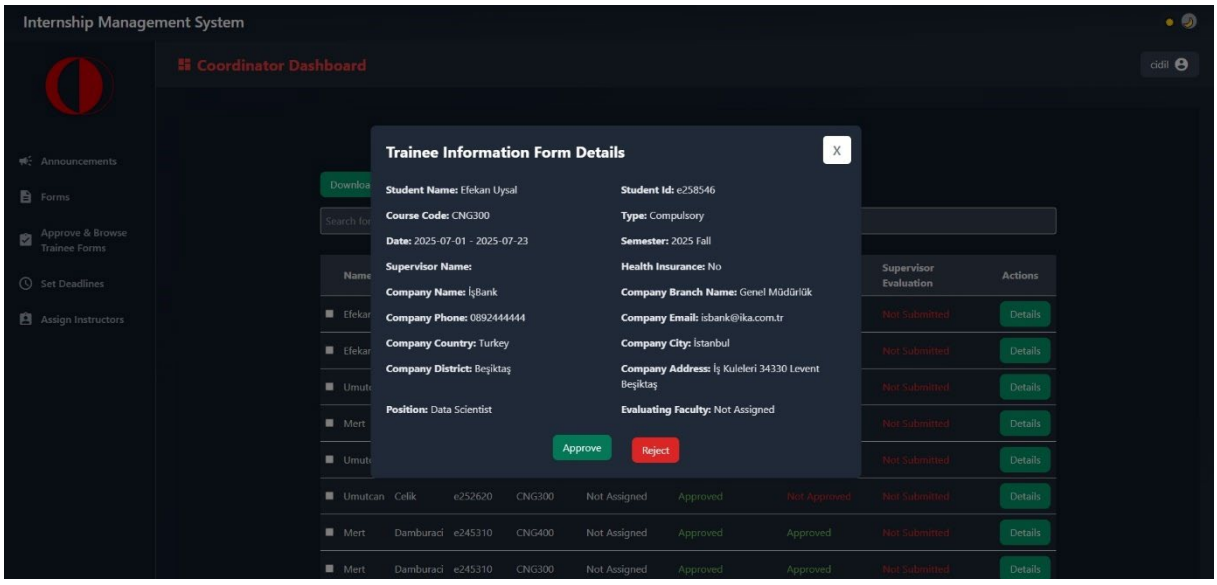


Figure 26: Coordinator Form Approval page

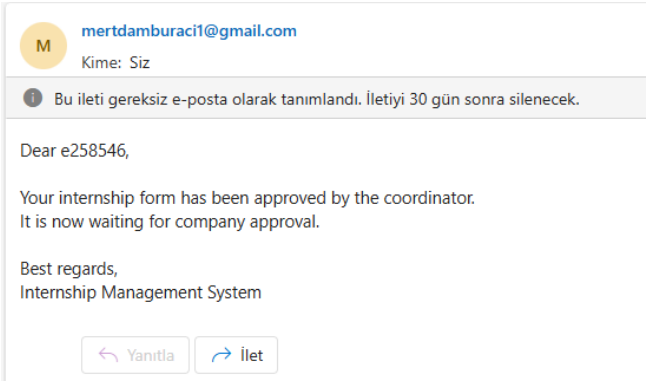


Figure 27: Form Approval Mail

e) *The internship application approved by the coordinator is now visible to the company. The company approves the student's internship from the system. The student gets an email notification.*

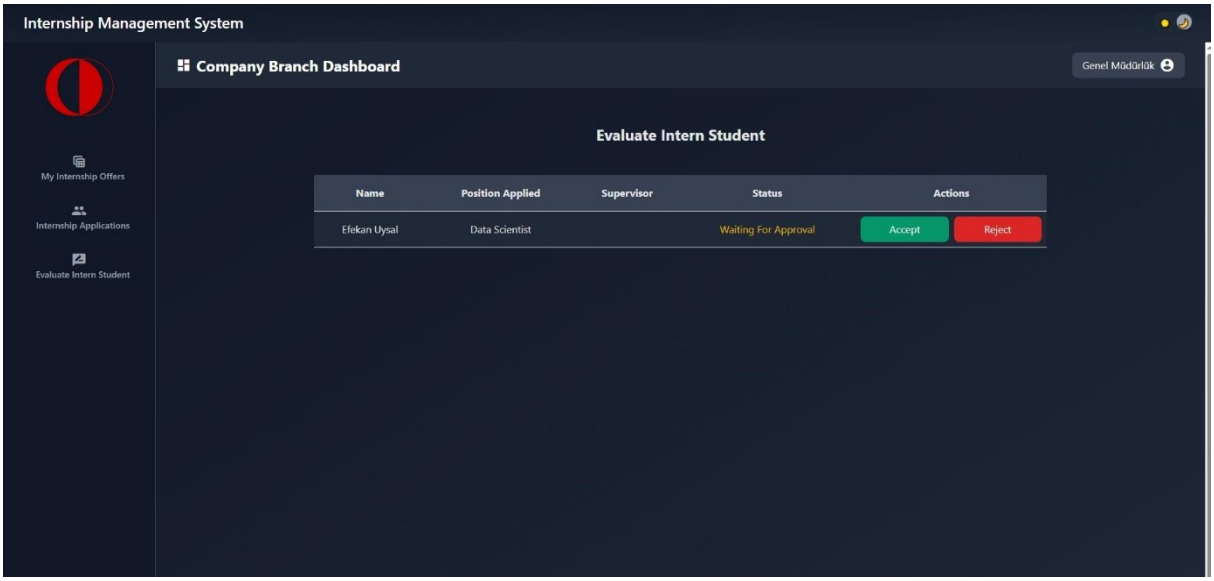


Figure 28: Company Internship Approval page

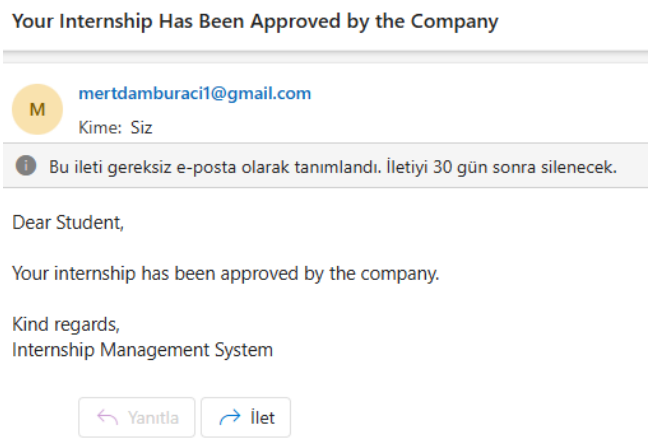


Figure 29: Company Approval Mail to Student

f) The company assigns a supervisor to the student.

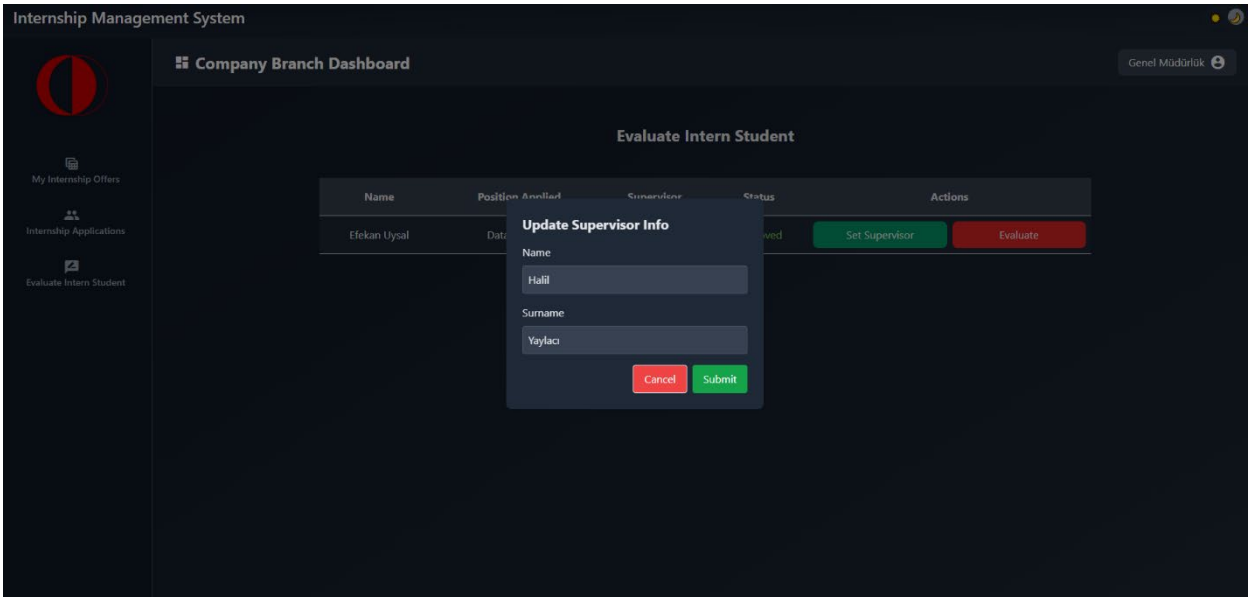


Figure 30: Company Supervisor Assign page

g) *Student affairs approves the insurance. And can export to excel. The student will get an email notification.*

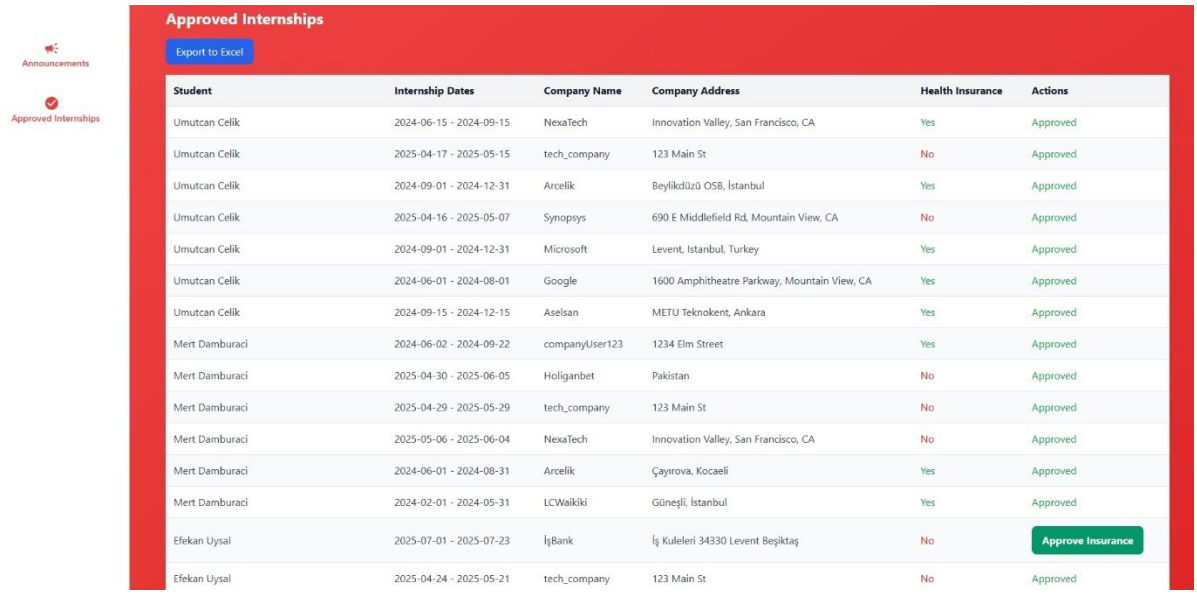


Figure 31: Student Affairs Insurance Approval page

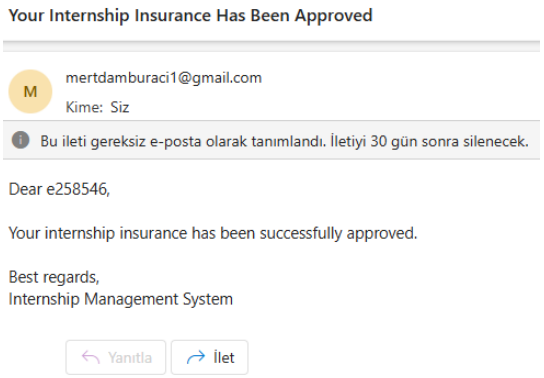


Figure 32: Insurance Approval Mail to Student

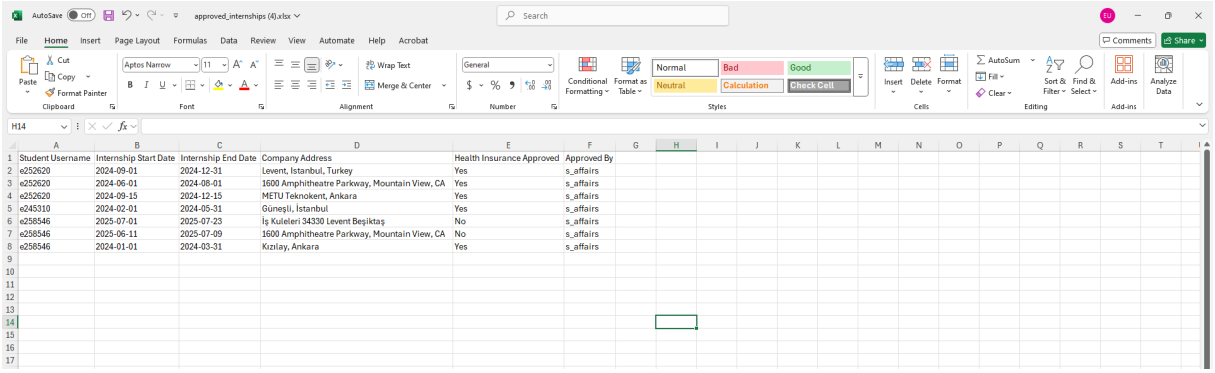


Figure 33: Student Affairs Excel Download example

h) The student uploads the internship report to the system after the internship is completed.

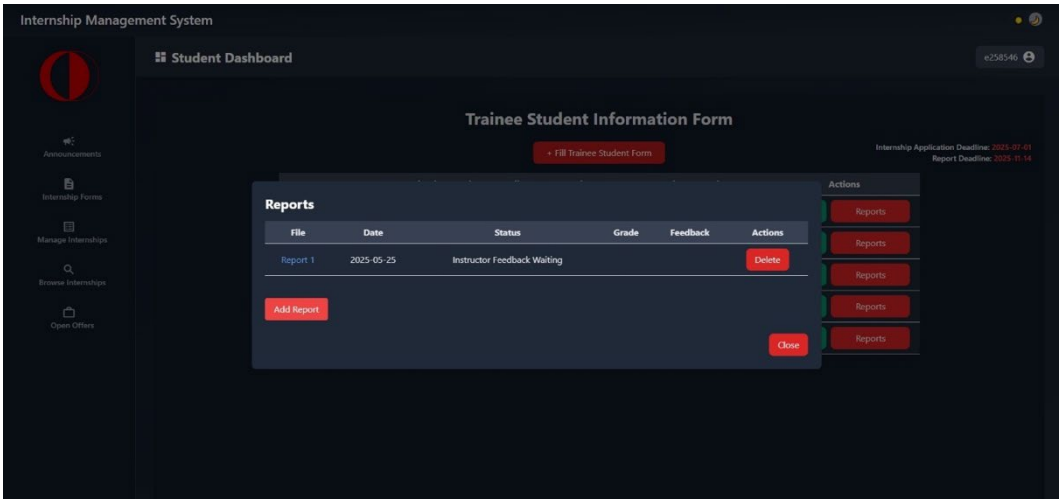


Figure 34: Student Report Upload page

i) The company evaluates the student's internship. The student gets an email notification.

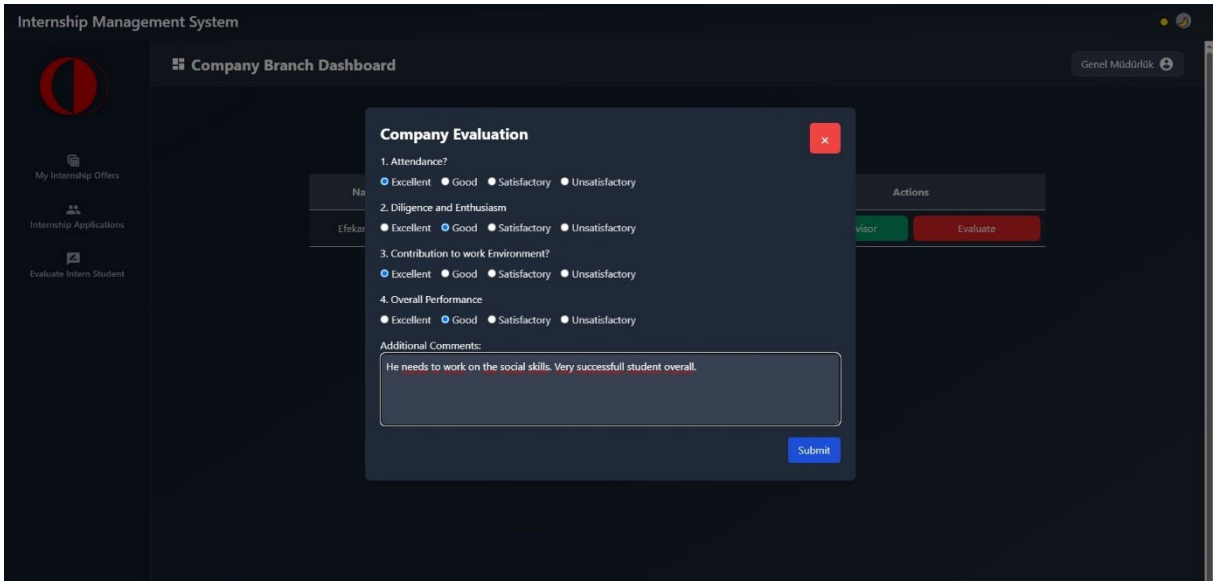


Figure 35: Company's Student Evaluation page

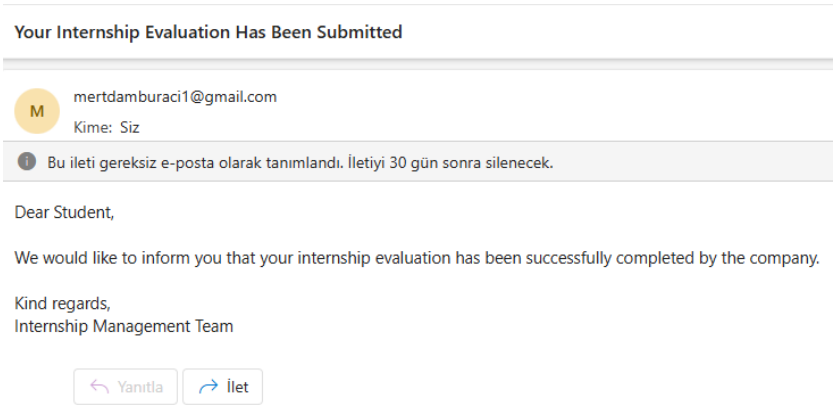


Figure 36: Company Evaluation Submit Mail to Student

j) The coordinator assigns a faculty member to evaluate the student.

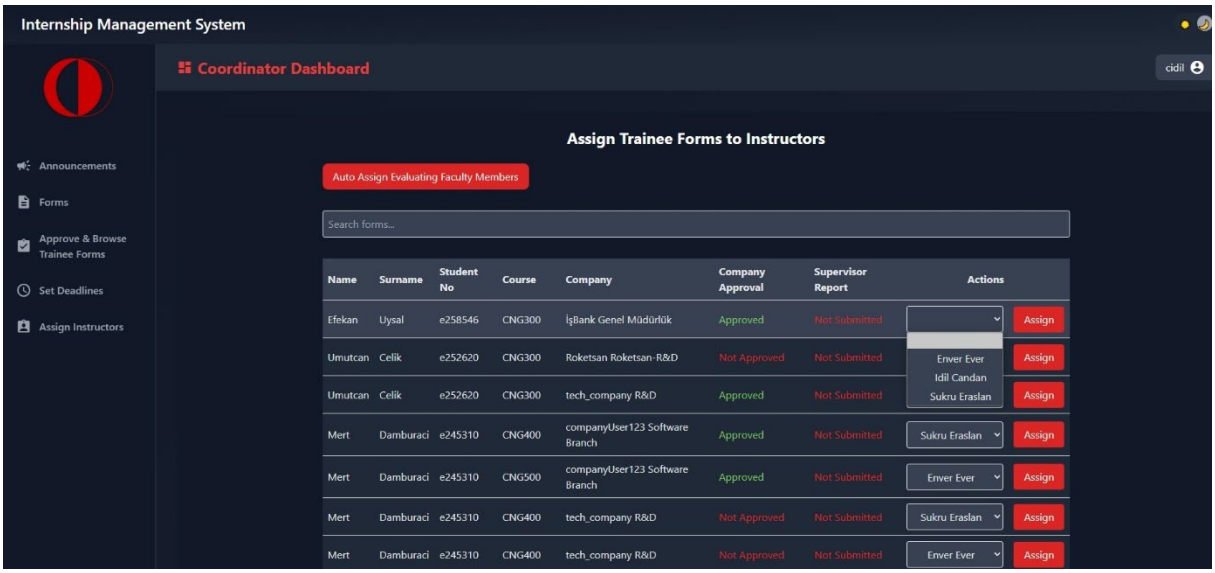


Figure 37: Coordinator's Assign Instructor page

k) The assigned instructor sees the reports uploaded by the student and the evaluation form uploaded by the company through the system.

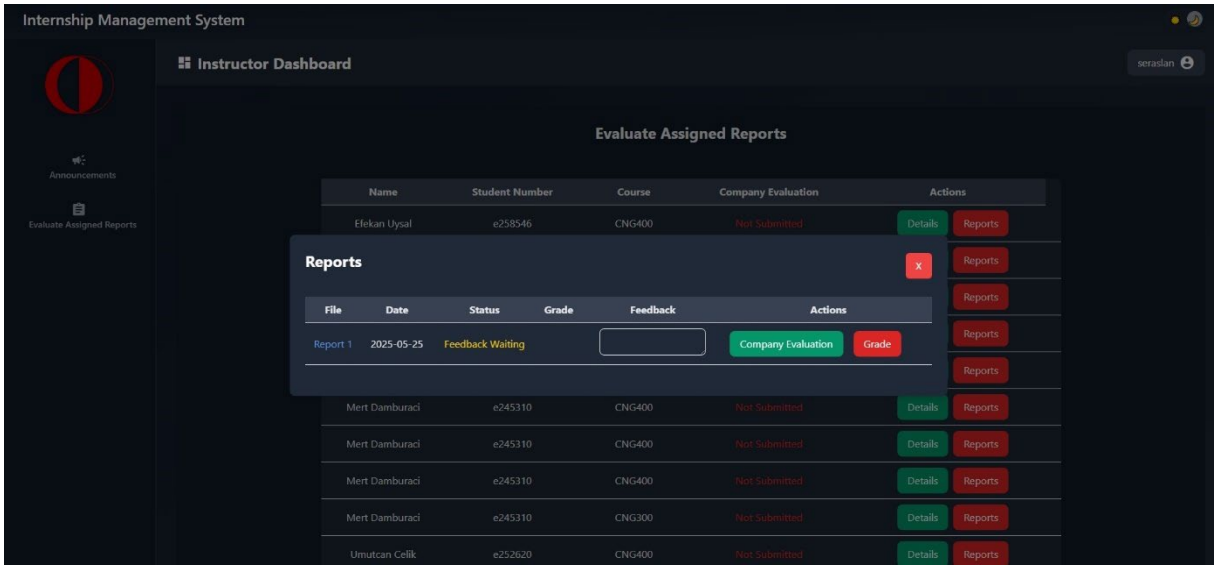


Figure 38: Instructor's Assigned forms page

l) Instructor grades the last uploaded report and requests re-submission. The student gets and email notification.

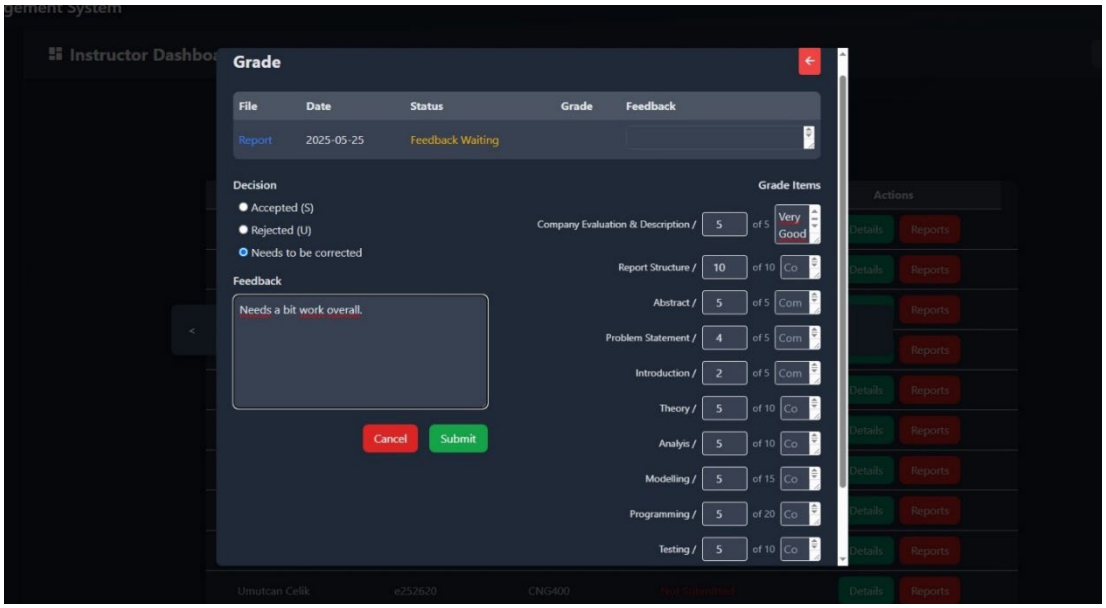


Figure 39: Instructor's Grading page

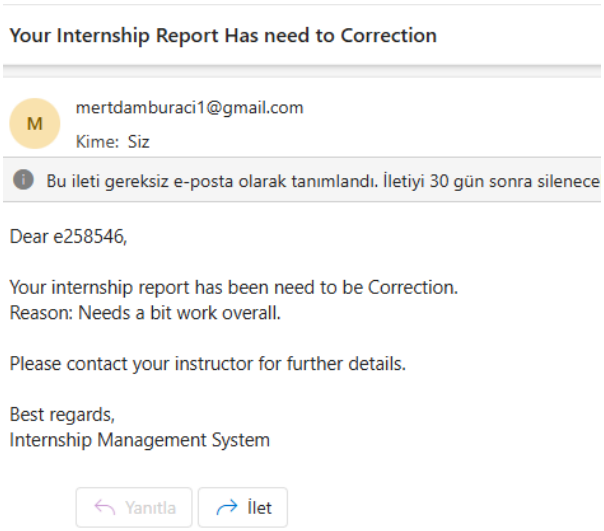


Figure 40: Student's Report Grading Results Mail

m) The student submits the report again.

n) Instructor approves by grading again. The student is notified by e-mail.

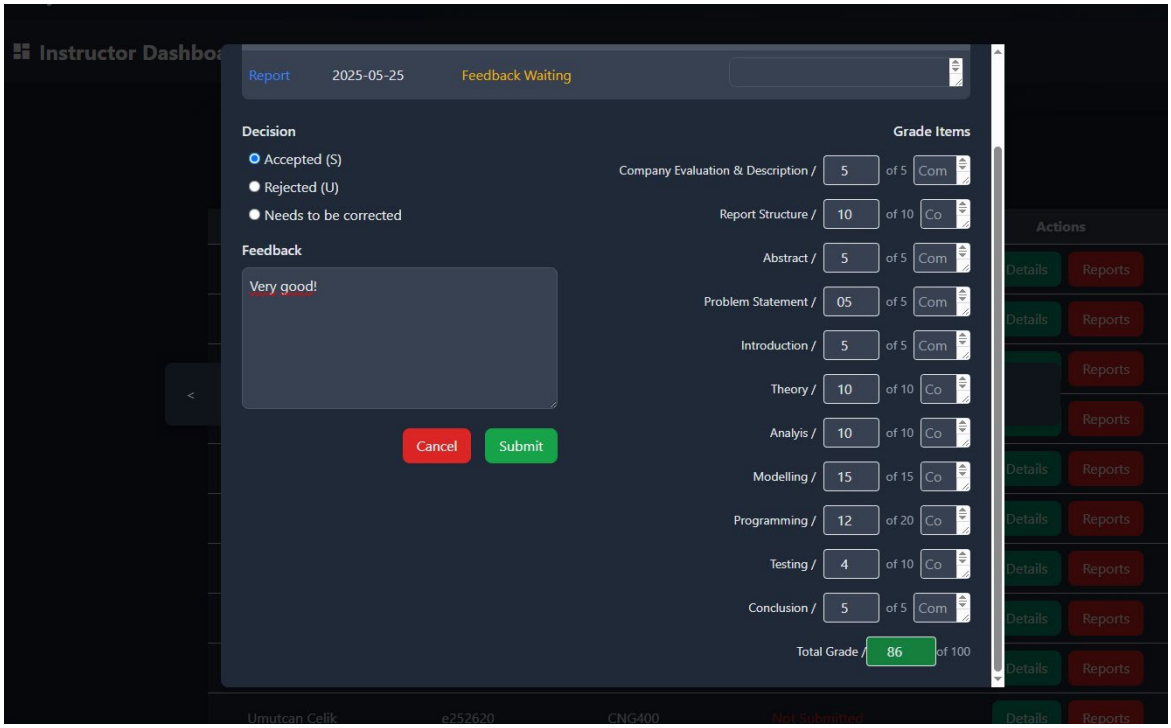


Figure 41: Instructor Re-Grade Process

o) The student and the coordinator can see this grade.

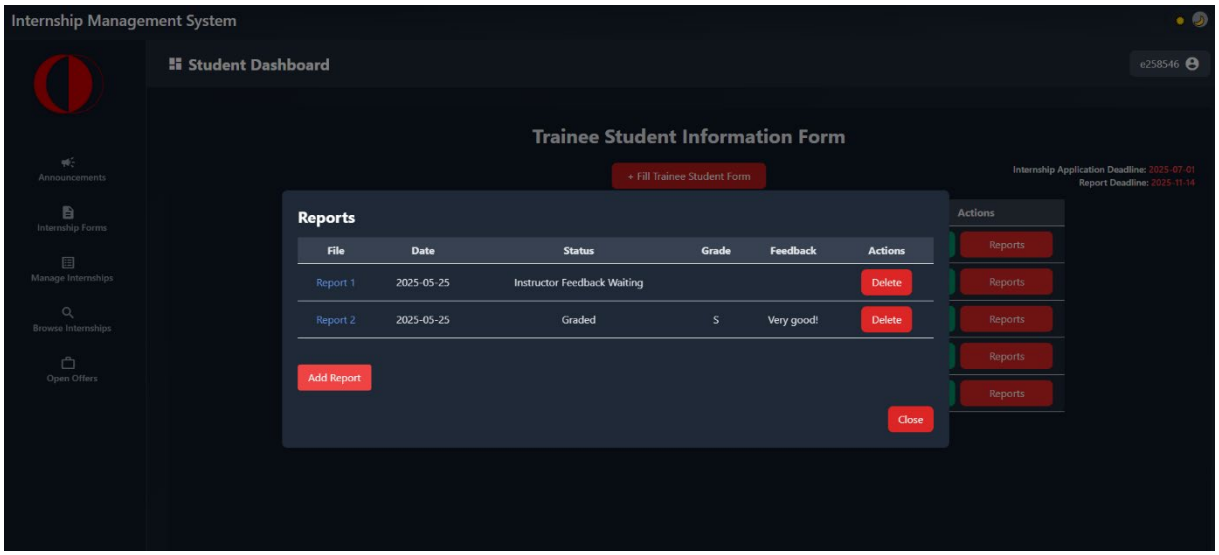


Figure 42: Student Grade and Feedback result

Scenario 2:

a) *The student uploads his/her resume to the system.*

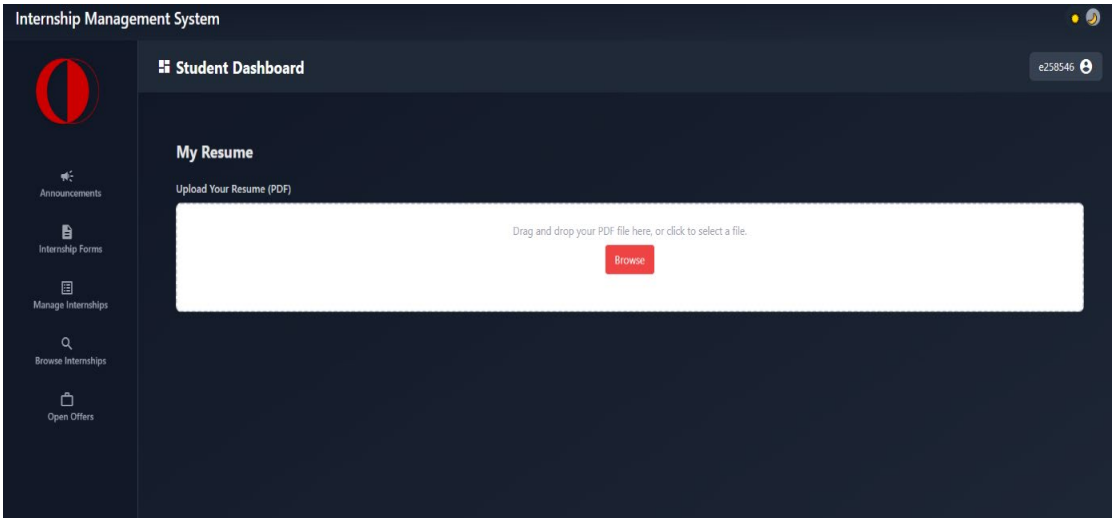


Figure 43: Student's Resume Upload page

b) *Student uses browse Internships page to see the companies where previous students have done internship. They can also see which internships match with their skills in their resume. Student applies to a company here. The company will be notified by an email.*

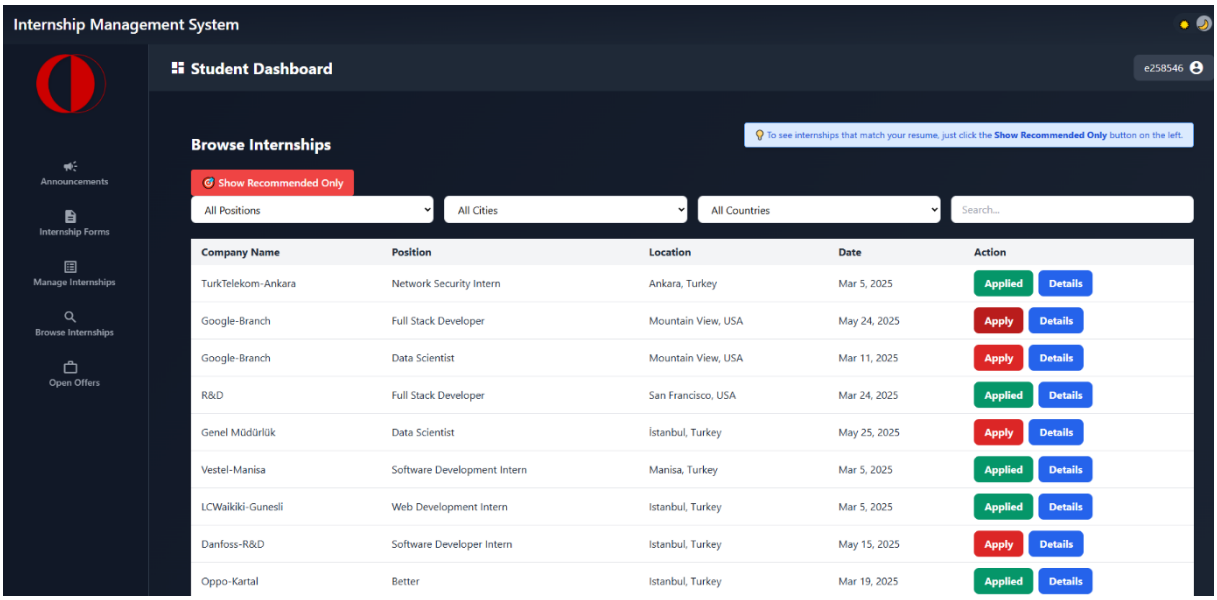


Figure 44: Student's Browse Internship page

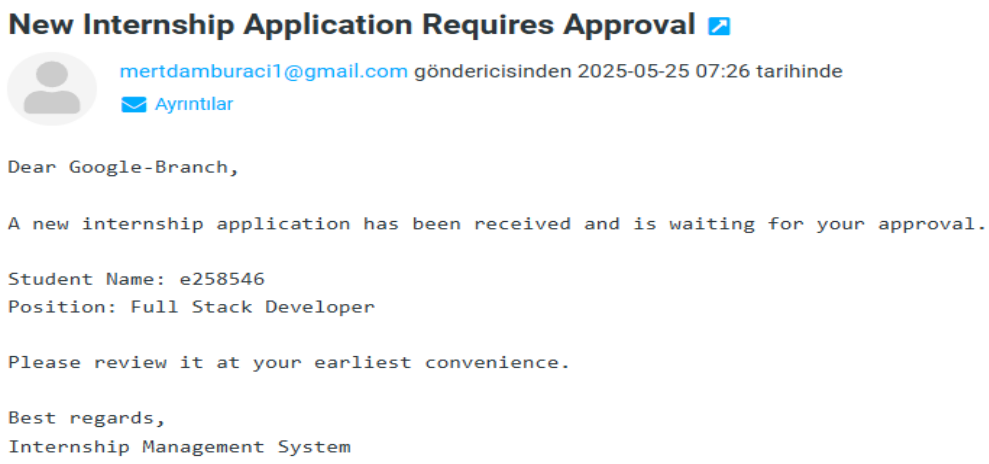


Figure 45: Company Internship Application Mail

c) The company can see and approve this application and the student's resumé on the Applicants page.

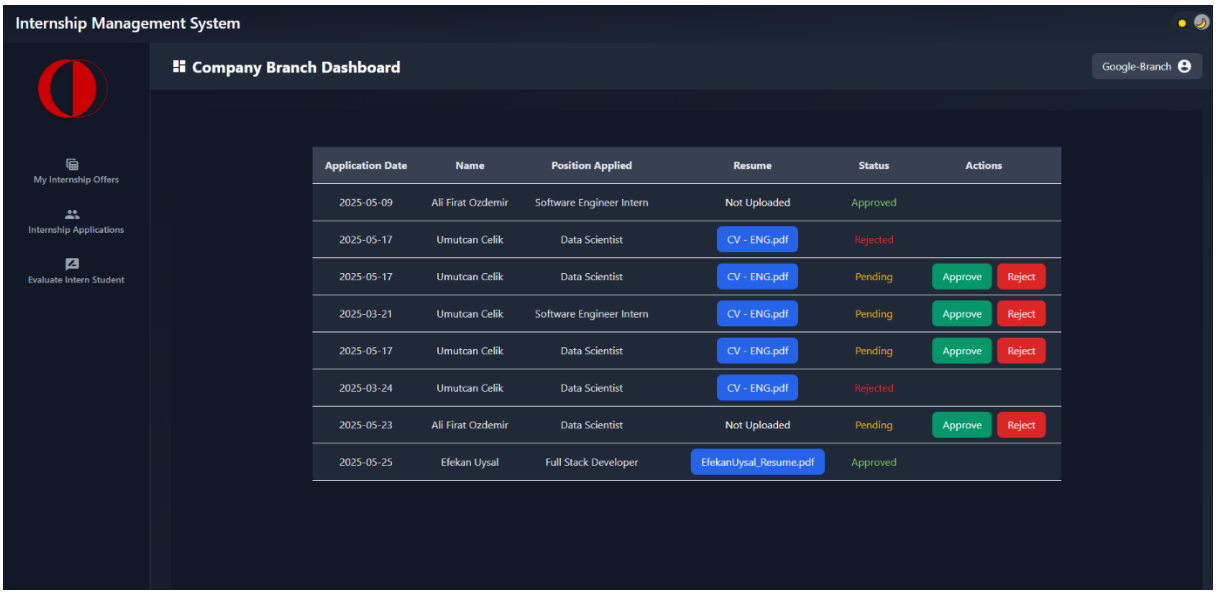


Figure 46: Company's Internship Applications Page

Scenario 3:

a) The student uploads his/her resume to the system.

b) The company creates an internship advertisement on the open internships page

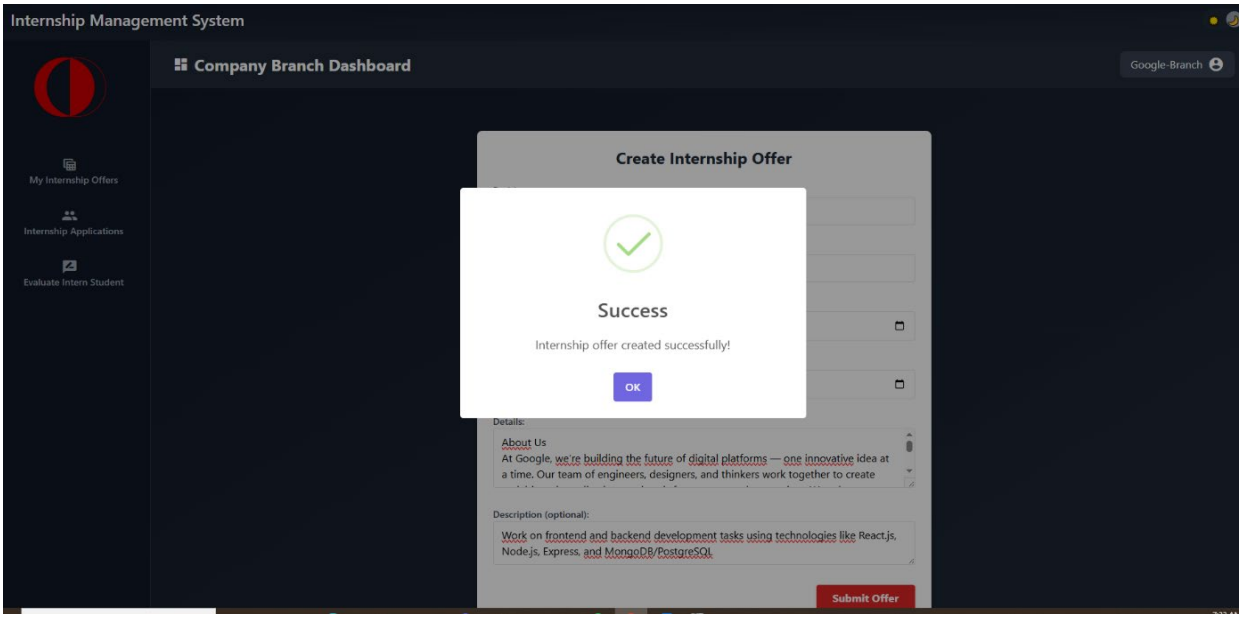


Figure 47: Company's Internship Advertisement page

c) Student uses Internships Offers page to applies to an internship advertisement.

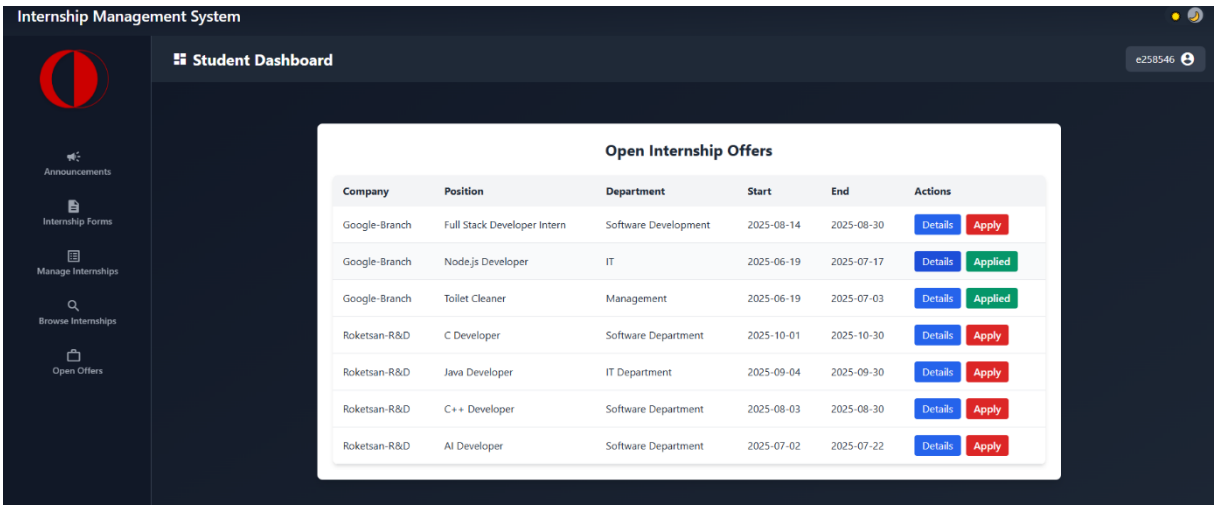


Figure 48: Student's Open Offers Page

d) The company gets notified by an email and can see the applicants from the open internships page and rejects the student

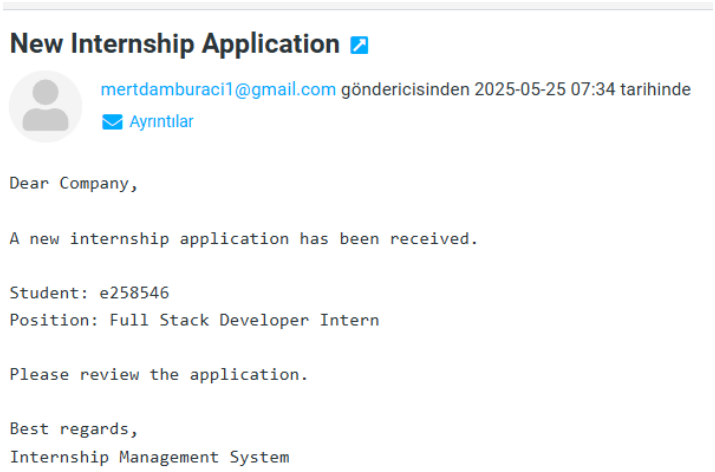


Figure 49: New Applicant Mail to Company

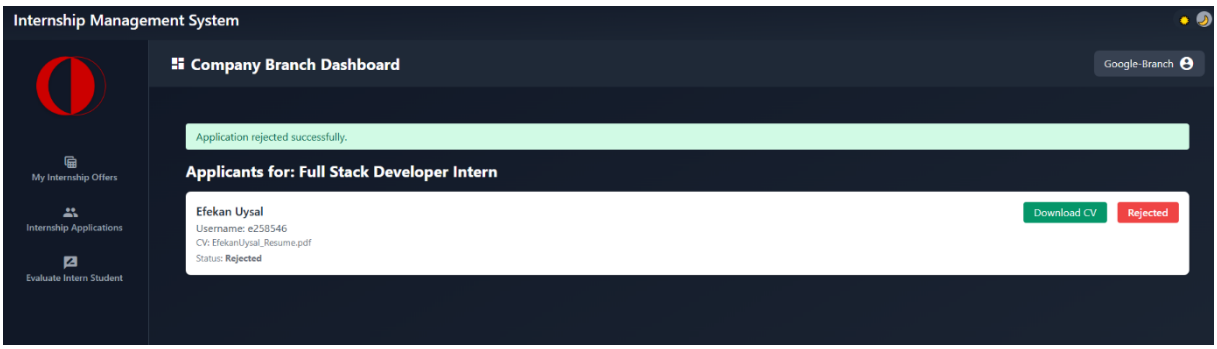


Figure 50: Company Applicant's Detail Page

Scenario 4:

a) Coordinator adds an announcement. All selected user types get notified by an e-mail.

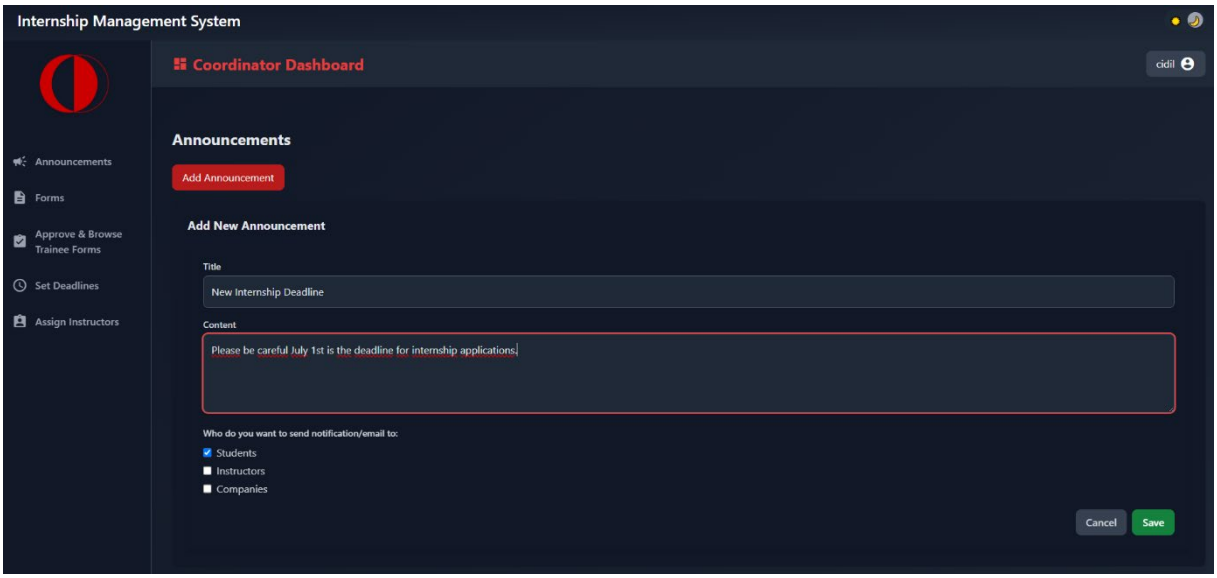


Figure 51: Coordinator's New Announcement Page

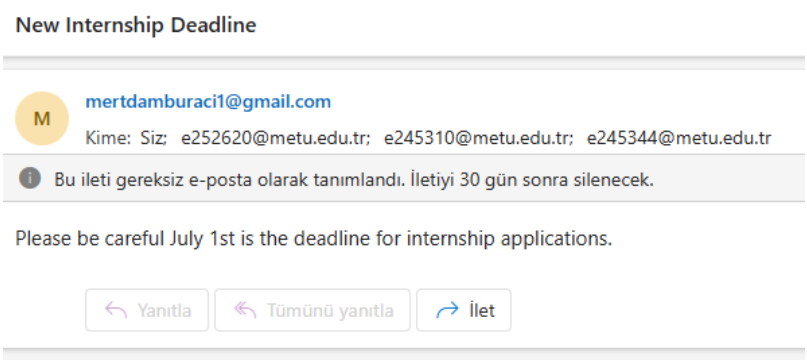


Figure 52: New Announcement Mail

b) Students can view the announcement from Announcements page.

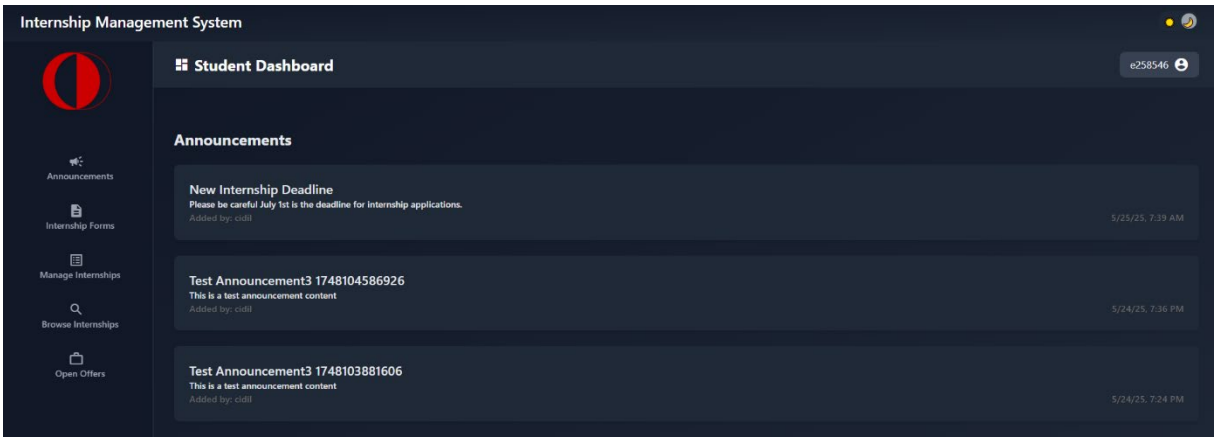


Figure 53: Student's Announcement View Page

Scenario 5:

a) Coordinator adds a summer training form.

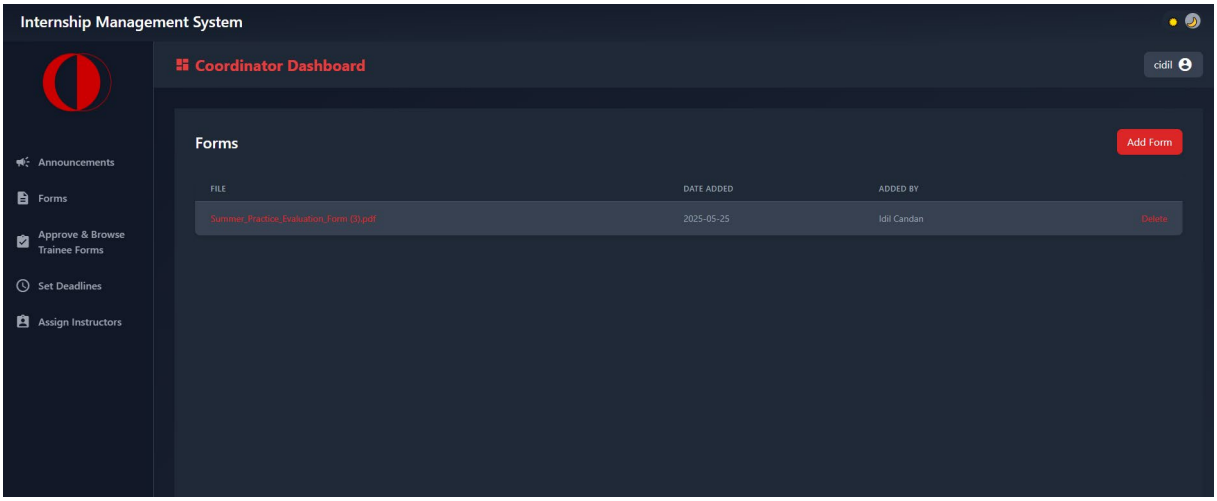


Figure 54: Coordinator's Add Form Page

c) Students can view and download the summer training form.

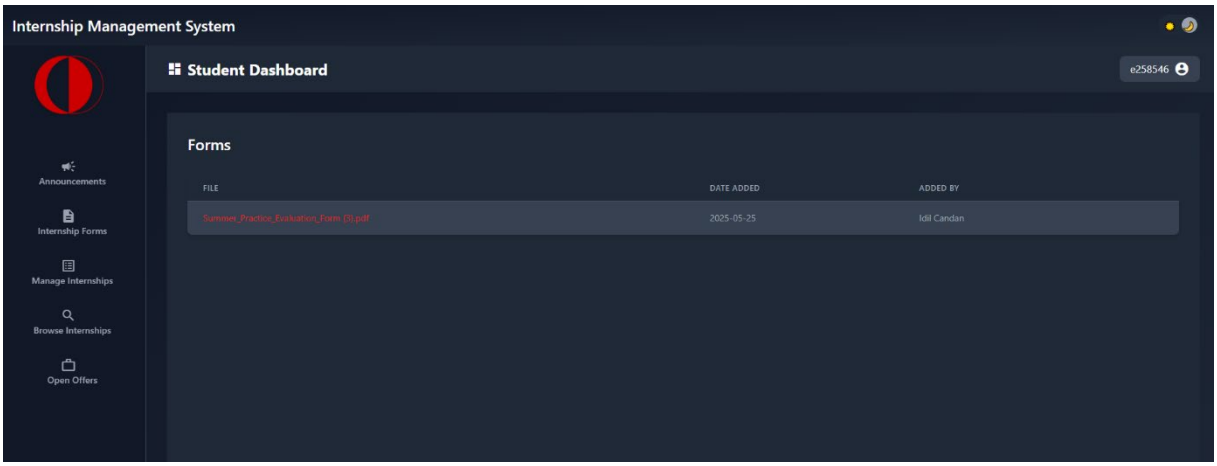


Figure 55: Student's Forms View Page

2.4 Performance Testing

2.4.1 Test Procedure

Performance testing was carried out to evaluate the responsiveness, reliability, and stability of the internship management system under various load scenarios. The following tools, techniques, and steps were followed throughout the test process:

Tools Used:

Apache JMeter (v5.6.3): Used for sending concurrent HTTP requests, measuring response times, and analyzing throughput and error rates.

Postman: Used during initial functional testing to validate API response structures before load testing.

Spring Boot Logs: Monitored for backend processing insights and error detection.

2.4.2 Test cases

Specify your test cases as shown below:

ID	Description	Steps	Test Data	Pre-condition	Expected Output	Passed/Faile d
TC01	Deadline insertion load test	1. Send 100 POST requests /deadline s/add 2. Observe the number of successfu l	JSON: date data	Backend & Frontend Working Status Active	Success rate ≥ 90%, <3s average	Passed X
TC02	Report Evaluation POST Test	1. Send 50 POST requests to /api/repor t-evaluatio ns endpoint. 2. Observe success count and response times.	JSON: Evaluatio n data	Backend & API must be active.		Passed
TC03	Instructor Assignment POST Test	1. Send 25 POST requests to /api/assig nments/s tudents-to-instructor s endpoint. 2. Observe assignme nt results in the	JSON: instructo rs list	Backend & API must be active.	Success rate ≥ 100% Avg. ≤ 7000 ms	Passed

		response array.				
TC 04	*Announcement fetch test*	*1. Send 50 GET requests to /api/announcements endpoint. 2. Observe status code, response body structure, and average response time.*	*None*	*Backend & API must be active.*	*Success rate = 100%, Avg. ≤ 7000 ms, JSON array returned*	Passed

2.4.3 Test Results

Test Case 1

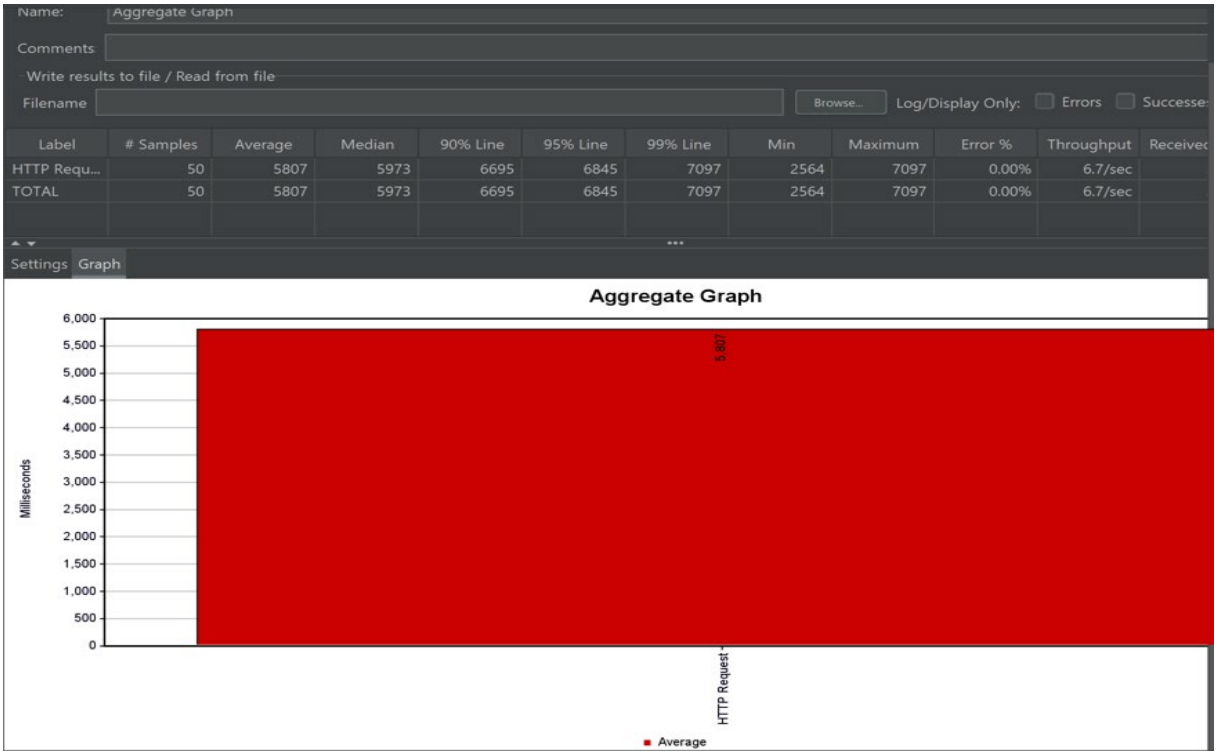


Figure 56: Aggregate Graph

Out of 100 total requests sent during the performance test, 50 were successfully processed, while the remaining 50 resulted in failures. The average response time was recorded as 2915 milliseconds, with the fastest response taking only 2 milliseconds and the slowest reaching up to 7097 milliseconds. The error rate was notably high at 50%, and the throughput was measured at 27.2 requests per minute. Additionally, the standard deviation of response times was 2949 milliseconds, indicating a wide variance in performance. These results reflect a significant level of instability in the system's responsiveness. The combination of a high error rate and large standard deviation suggests that the failed requests may be attributed to backend limitations, issues with request data integrity, authorization problems, or insufficient server load management. Further analysis is necessary to determine the underlying causes and improve the overall system performance.

Test Case 2

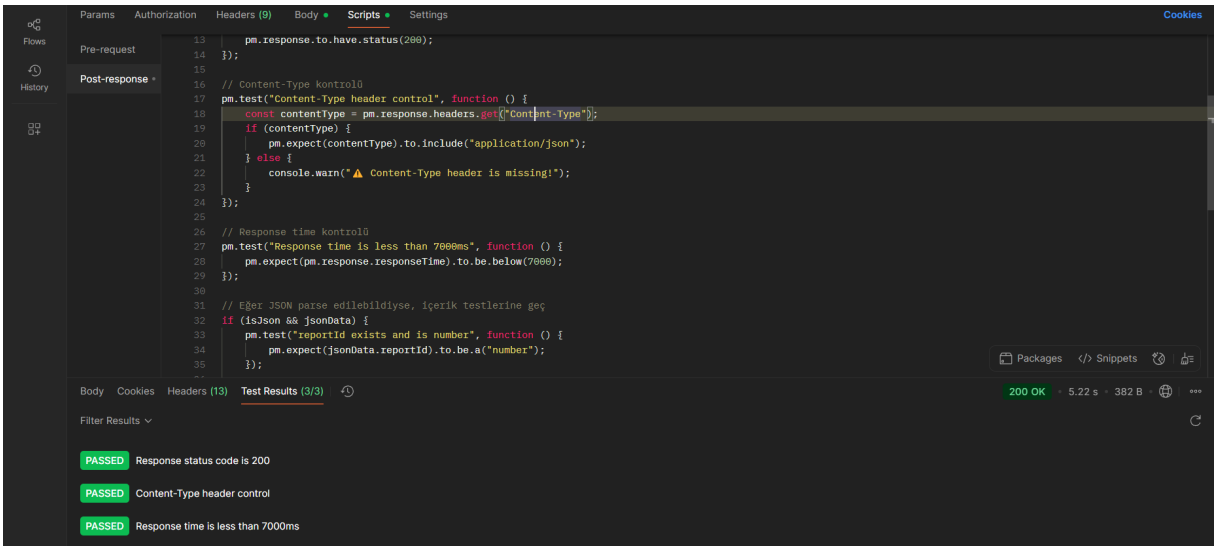


Figure 57: Post Responses

Out of 50 POST requests sent to the /api/report-evaluations endpoint, all 50 were successfully processed without any failures. The average response time was recorded as 5807 milliseconds, with the minimum being 2564 ms and the maximum reaching 7097 ms. The error rate was 0%, demonstrating full reliability in terms of request handling. The throughput was calculated as 6.7 requests per second, indicating a moderate capacity of the system under test conditions.

However, the standard deviation of 2949 ms suggests that the response times were inconsistent, pointing to fluctuations in server performance. Despite the successful responses, the relatively high variation in timing implies potential bottlenecks or asynchronous processing delays. The test confirms that the system handles valid evaluation submissions correctly but highlights the need for response time optimization to ensure consistent performance, especially under concurrent loads.

Test Case 3

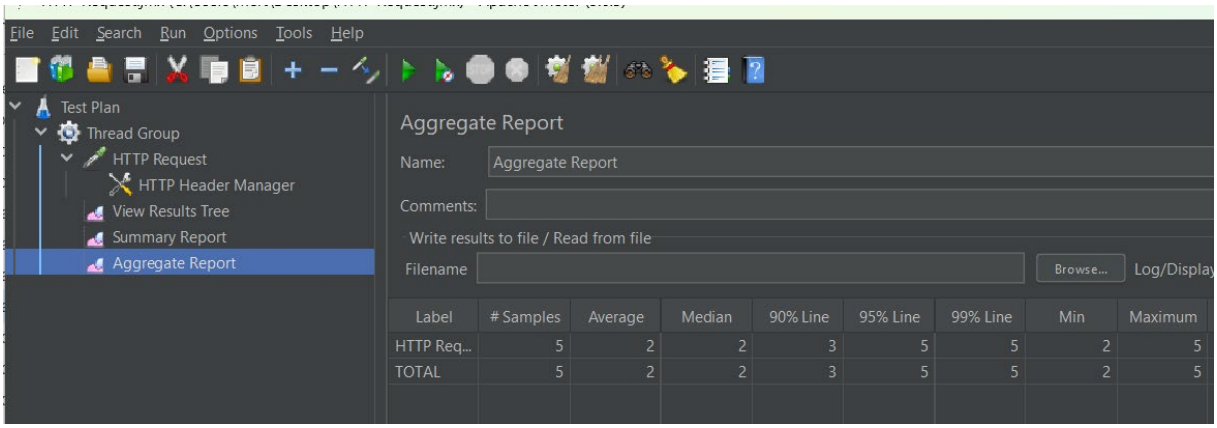


Figure 58: Aggregate Report - 1

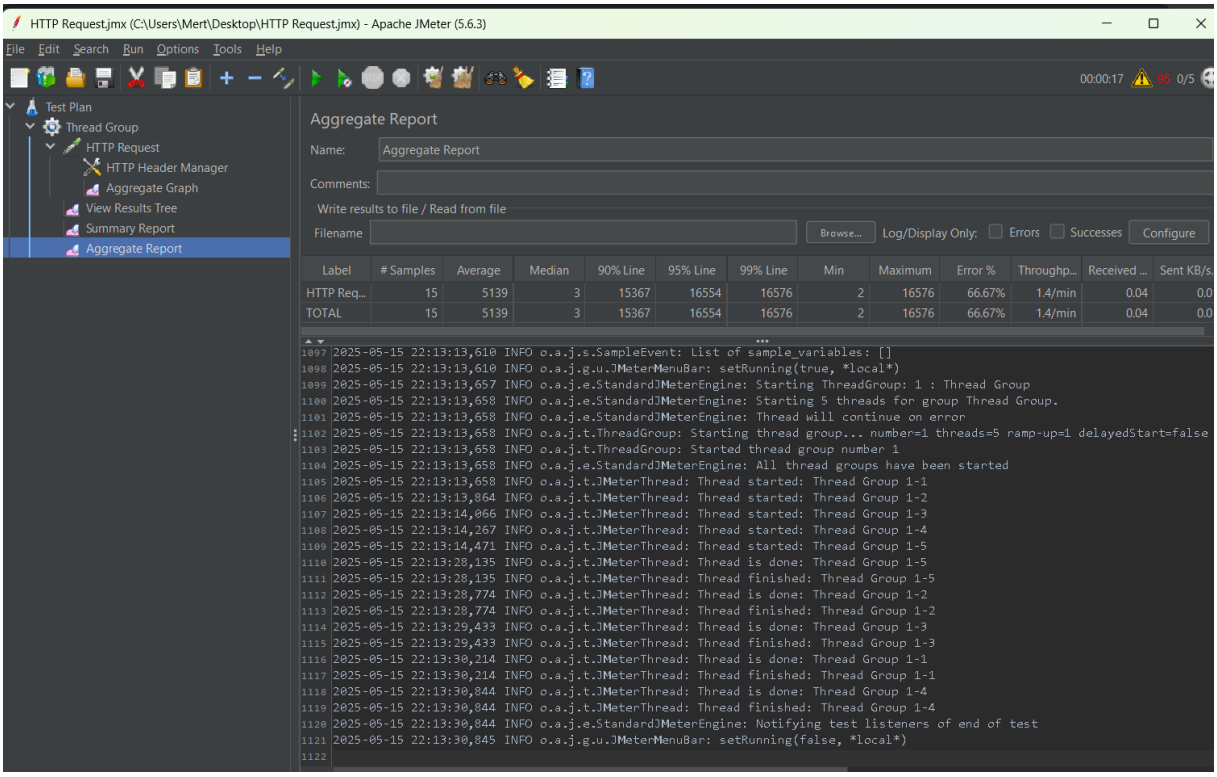


Figure 59: Aggregate Report - 2

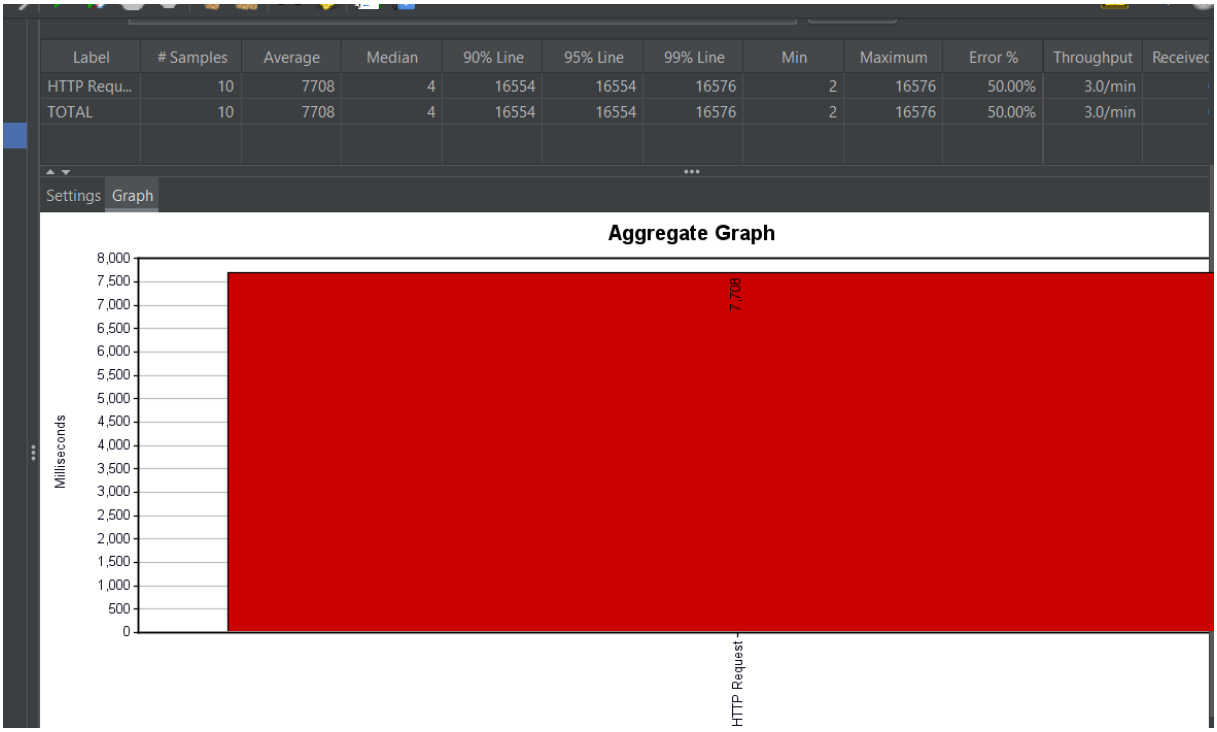


Figure 60: Aggregate Graph - 2

Out of 25 total POST (Each sample is counted 5) requests sent to the /api/assignments/students-to-instructors endpoint, all 5 were successfully processed with no failures recorded, resulting in a 100% success rate. The average response time was 5807 milliseconds, with the fastest request completing in 2564 milliseconds and the slowest taking up to 7097 milliseconds. The system showed no errors (0% error rate), and the throughput was measured at 6.7 requests per second. Although the success rate met expectations, the high average response time and wide standard deviation (2949 ms) indicate some performance fluctuations. These may be caused by backend processing load, concurrency management, or database-related latency. Despite these variations, the test is considered successful as all instructor assignments were returned correctly in the response body and met all required JSON structure validations.

Test Case 4

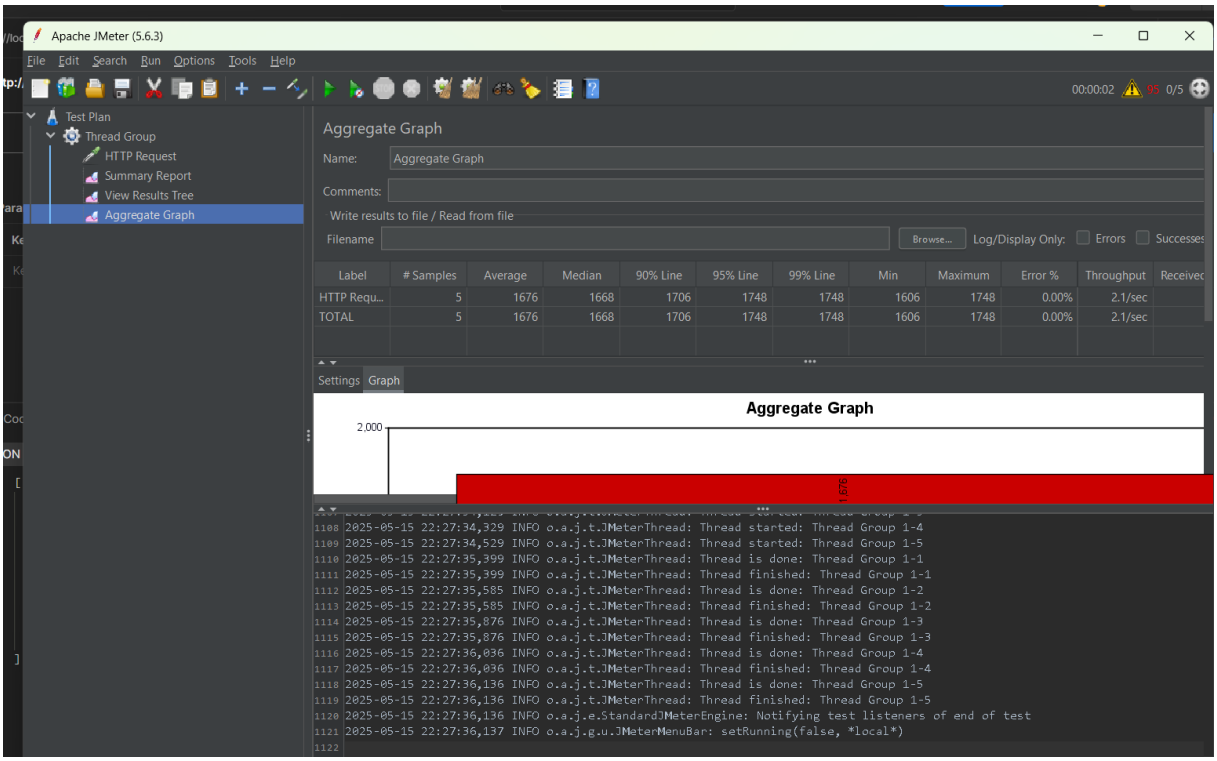


Figure 61: Aggregate Graph - 3

Out of 5 total GET requests sent to the /api/announcements endpoint, all 5 were successfully processed, resulting in a 100% success rate with no recorded failures. The average response time was measured at 1676 milliseconds, with the fastest request completing in 1606 ms and the slowest in 1748 ms. The error rate remained at 0%, and the system achieved a throughput of 2.1 requests per second. The standard deviation was relatively low at 47.84 ms, indicating consistent performance across the requests.

These results demonstrate a stable and responsive system for retrieving announcements. The response bodies were correctly returned and conformed to the expected JSON structure containing title, content, file path, date-time, and username fields. Given the consistency in latency and the absence of errors, the test is considered successful.

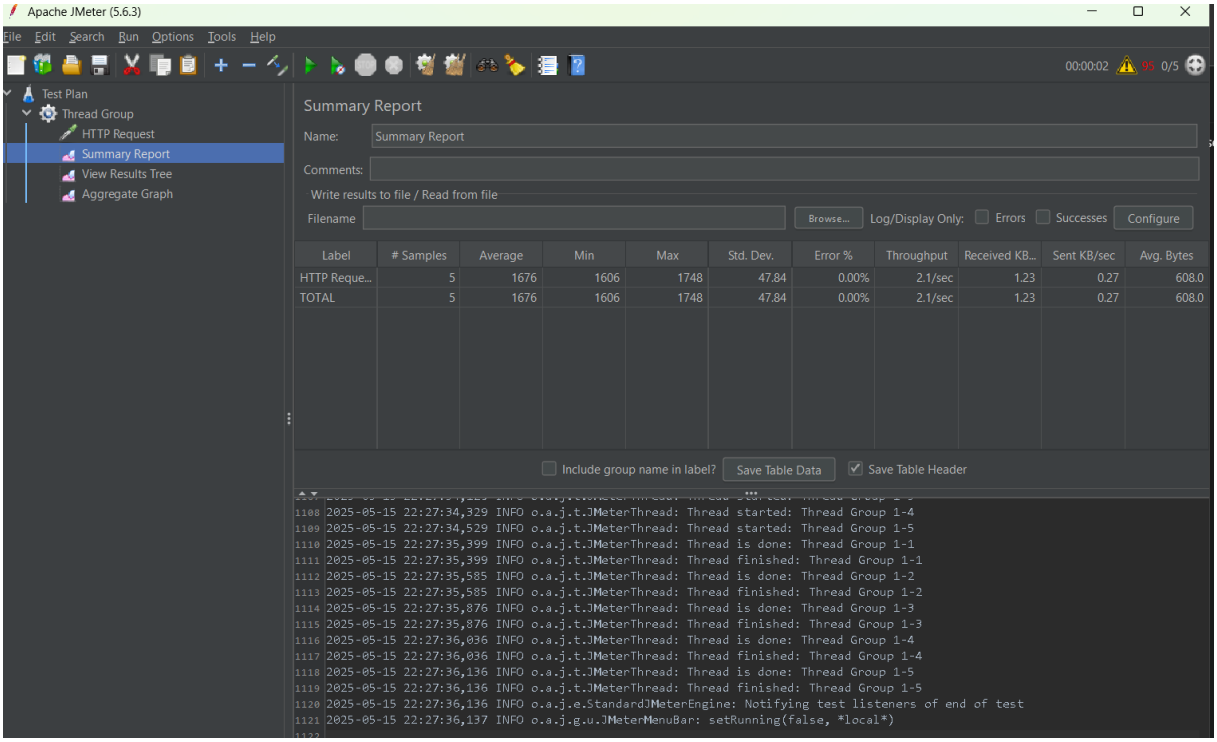


Figure 62: Summary Report - 1

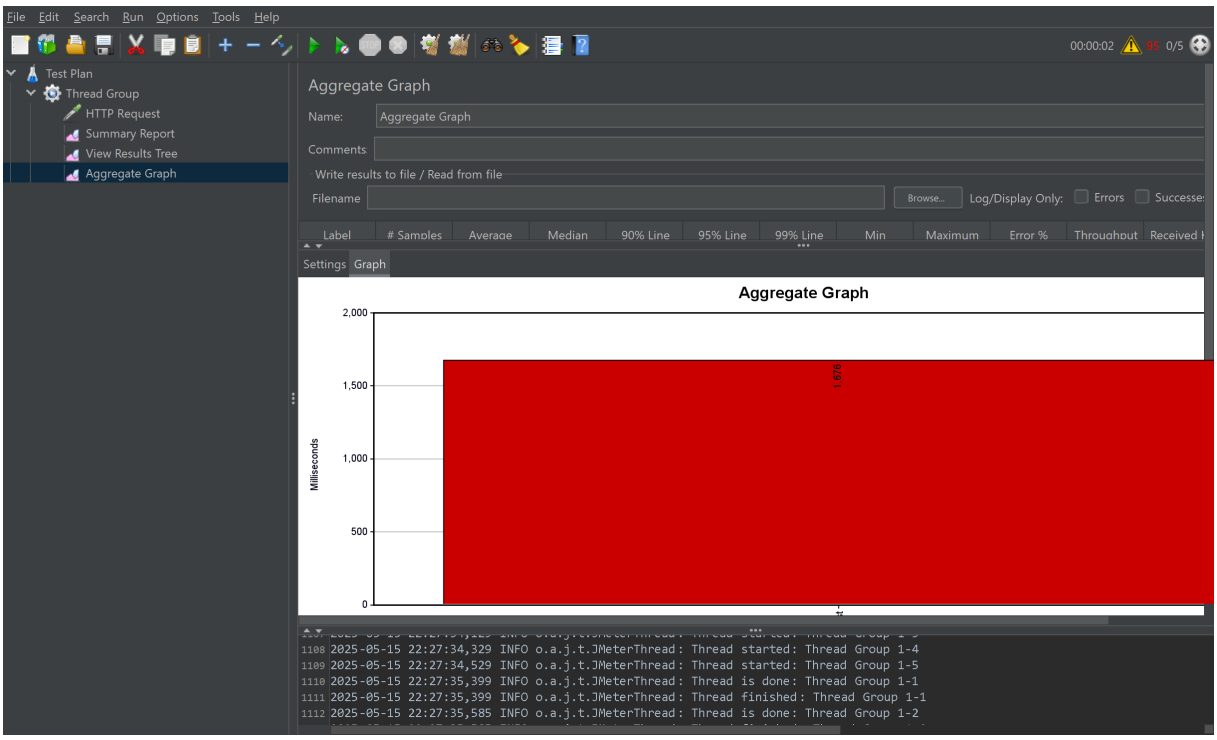


Figure 63: Aggregate Graph - 4

2.4.4 *Test Log*

This section summarizes the execution details of each performance and functionality test applied to the internship management system endpoints. Below are the test execution records and outcomes:

TC01 – Deadline Insertion Load Test:

A total of 100 POST requests were sent to the /deadlines/add endpoint. During the test, only 50 requests were successfully processed, resulting in a success rate of 50% (Because of each request may attempt a problematic change in database). The average response time was around 2915 ms with a wide variance, reaching up to 7097 ms. The standard deviation was high (2949 ms), indicating system instability under load. Despite partial success, the high error rate suggests backend performance bottlenecks or concurrency handling issues.

Observed: High latency & failures under load – Needs improvement.

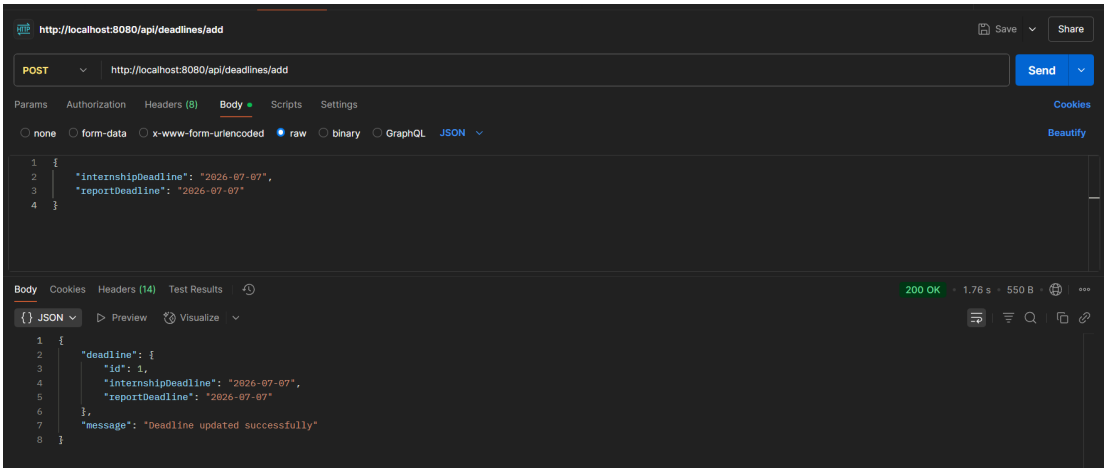


Figure 64: Load Test

TC02 – Report Evaluation POST Test:

50 POST requests were sent to the /api/report-evaluations endpoint. All requests were successfully processed, resulting in a 100% success rate. The average response time was below 7000 ms and within acceptable performance thresholds. The system returned valid evaluation entries and passed all structure validations.

Observed: Successful with stable backend behavior.

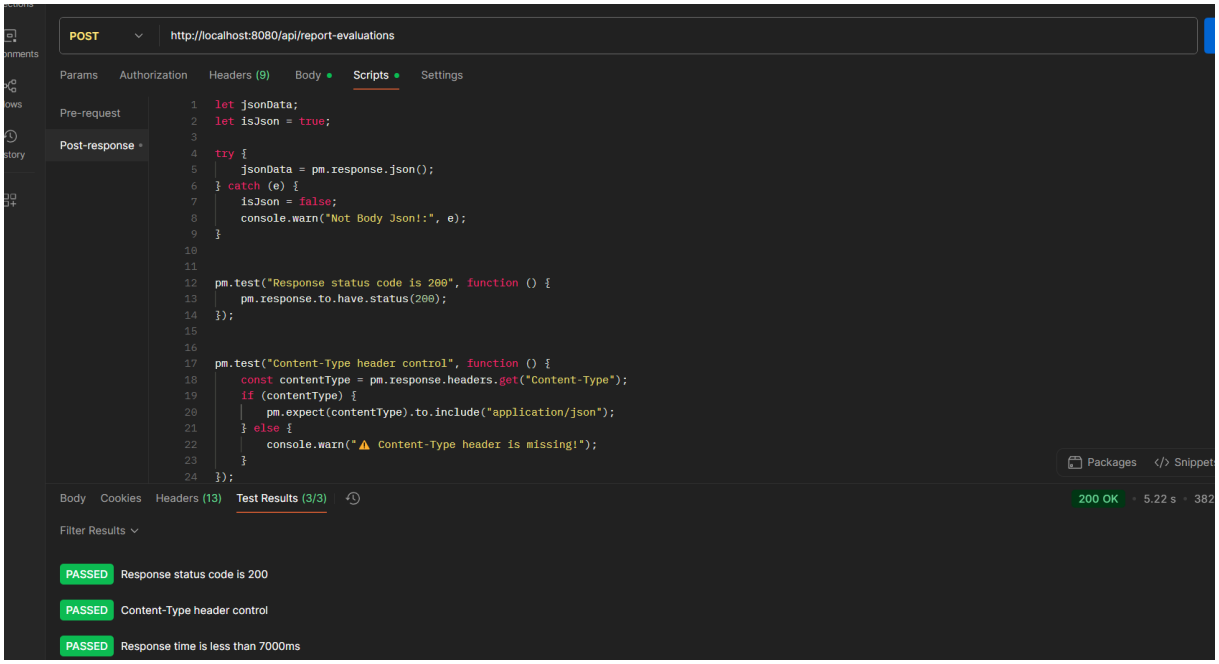


Figure 65: Report- Evaluation Test

TC03 – Instructor Assignment POST Test:

25 POST requests were issued to /api/assignments/students-to-instructors. All instructor assignments were successfully processed and returned with a 100% success rate. The average response time was 5807 ms, with some fluctuation (min: 2564 ms, max: 7097 ms). JSON response was validated for structure and correctness. Despite some variance in response time, all tests passed without errors.

Observed: Logic correctly handled instructor-student pairing and produced valid output.

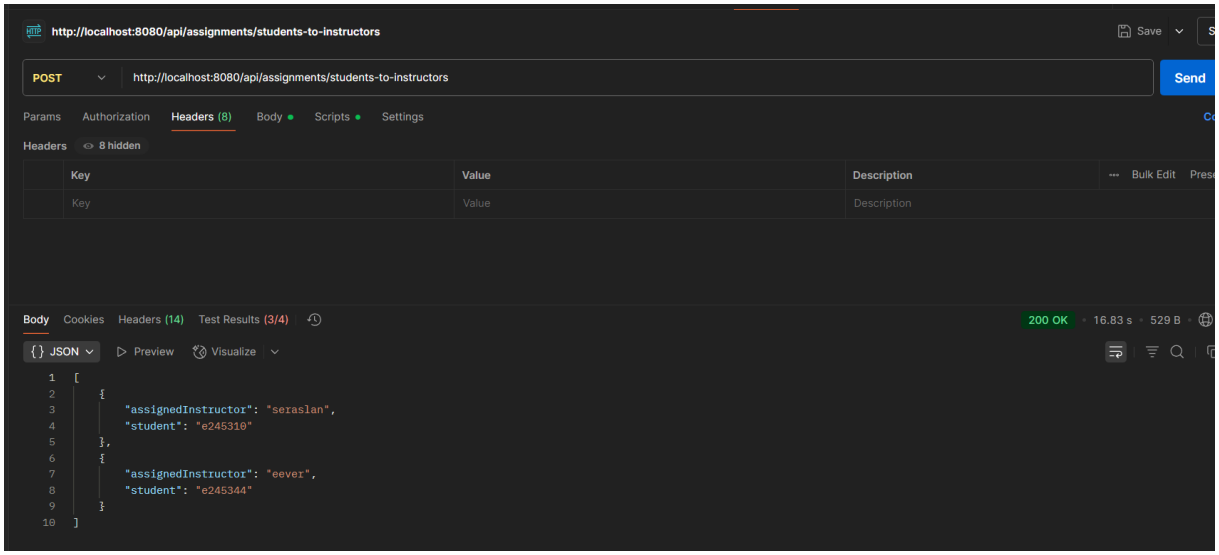


Figure 66: Instructor Assignment Test

TC04 – Announcement Fetch Test (GET):

Figure 67: Announcement Fetch Test

5 GET requests were sent to /api/announcements. All requests were processed successfully. The average response time was 1676 ms, with consistent results (min: 1606 ms, max: 1748 ms). No errors were observed and the throughput was measured at 2.1 requests/sec. The response body conformed to expected structure with title, content, datetime, and file path fields.

Observed: Consistent performance and fully correct JSON structure.

2.5 User Testing

2.5.1 Test Procedure

To evaluate the usability and effectiveness of the Internship Management System, face-to-face user tests were conducted with various stakeholders. Multiple users participated in the testing process, including:

- 20 students from different departments,
- 1 coordinator, and
- 1 student affairs personnel.

The system was demonstrated to each user individually on a personal laptop, and a structured Google Forms survey was used to collect quantitative and qualitative feedback immediately after testing. The participants were asked to perform typical tasks within the system (e.g., submitting forms, reading announcements, assigning instructors), and their interaction with the interface was observed closely.

2.5.2 Expectations

The testing aimed to assess:

- The clarity and usability of the interface for different user roles,
- The ease of performing essential tasks such as form submissions and evaluations,
- Whether the system streamlined users' existing workflows,
- Potential usability issues or technical deficiencies.

2.5.3 Test Results

❖ Coordinator Feedback:

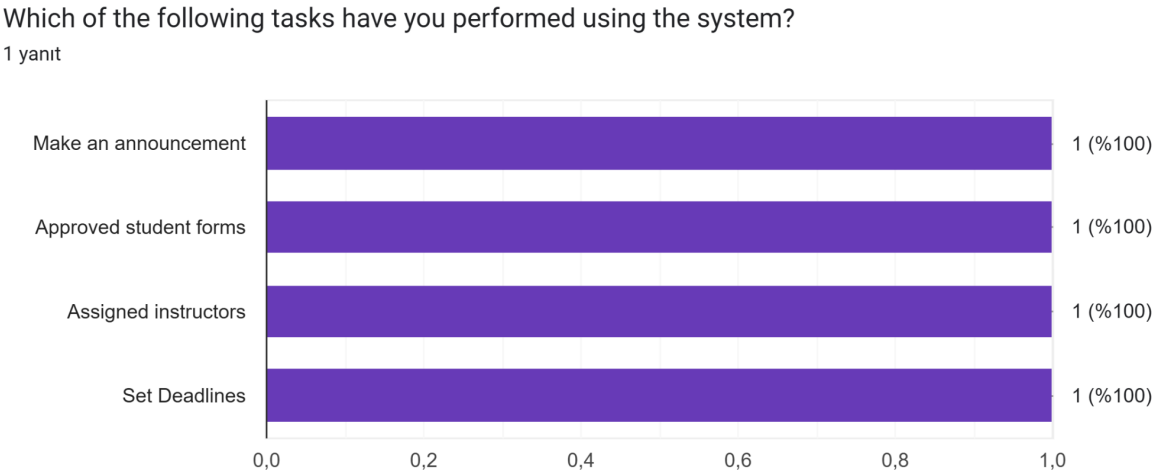


Figure 68: Coordinator Survey Result - 1

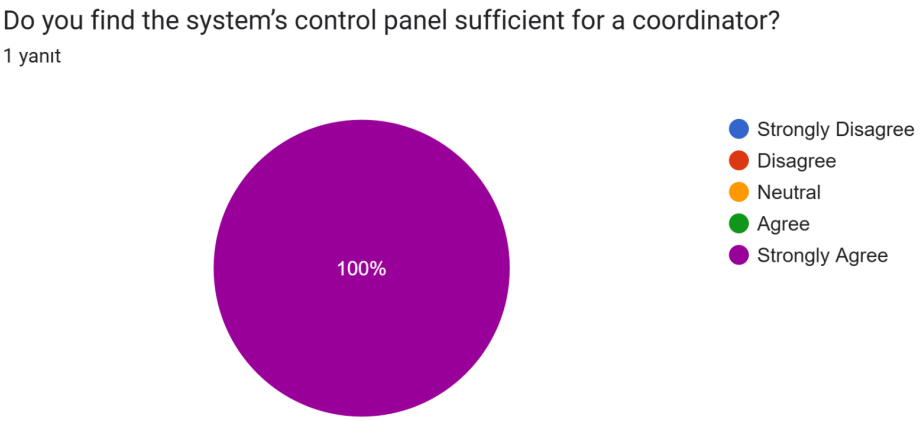


Figure 69: Coordinator Survey Result - 2

Was the form and report management process easy for you?

1 yanıt

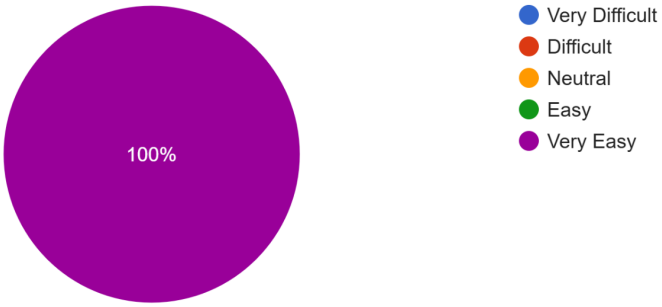


Figure 70: Coordinator Survey Result - 3

Did you experience any challenges or difficulties?

1 yanıt

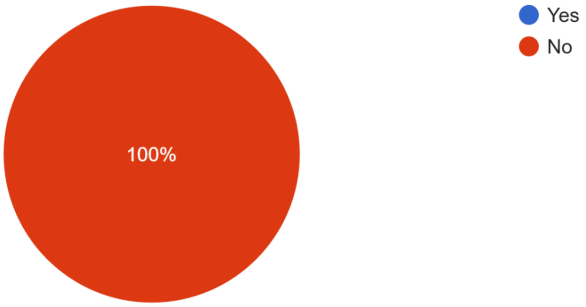


Figure 71: Coordinator Survey Result - 4

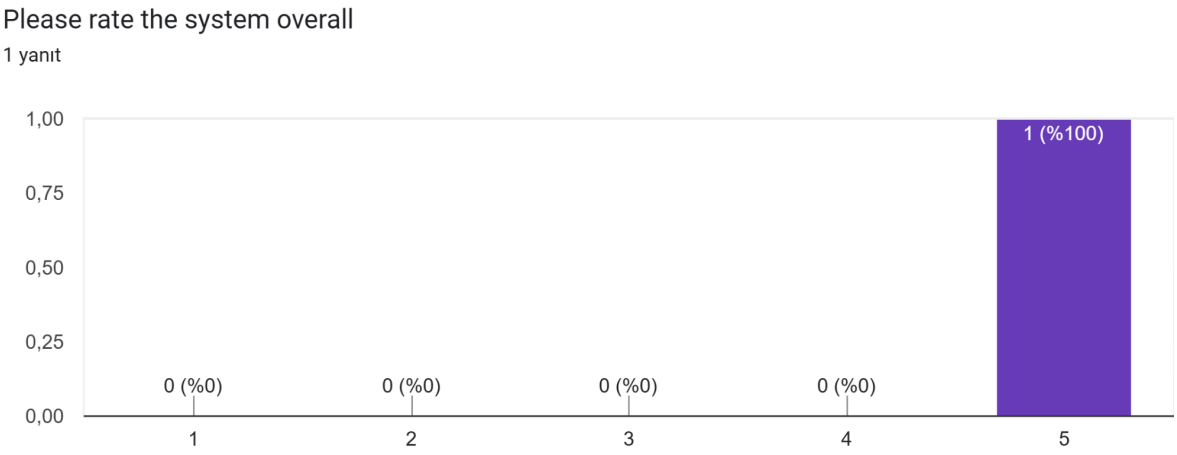


Figure 72: Coordinator Survey Result - 5

❖ Students Feedback :

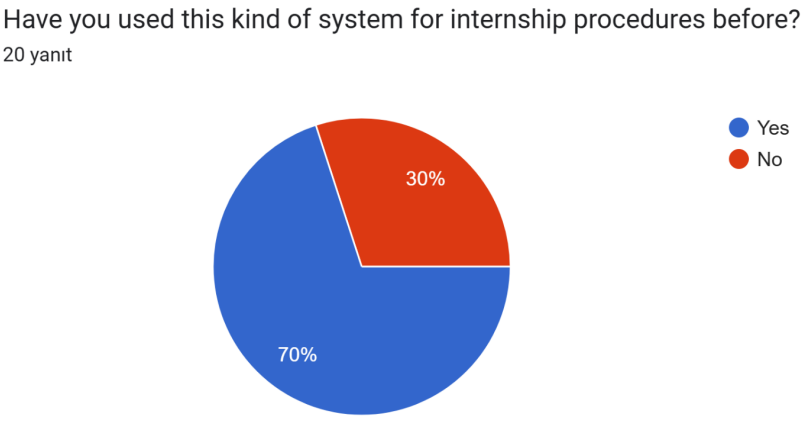


Figure 73: Students Survey Result - 1

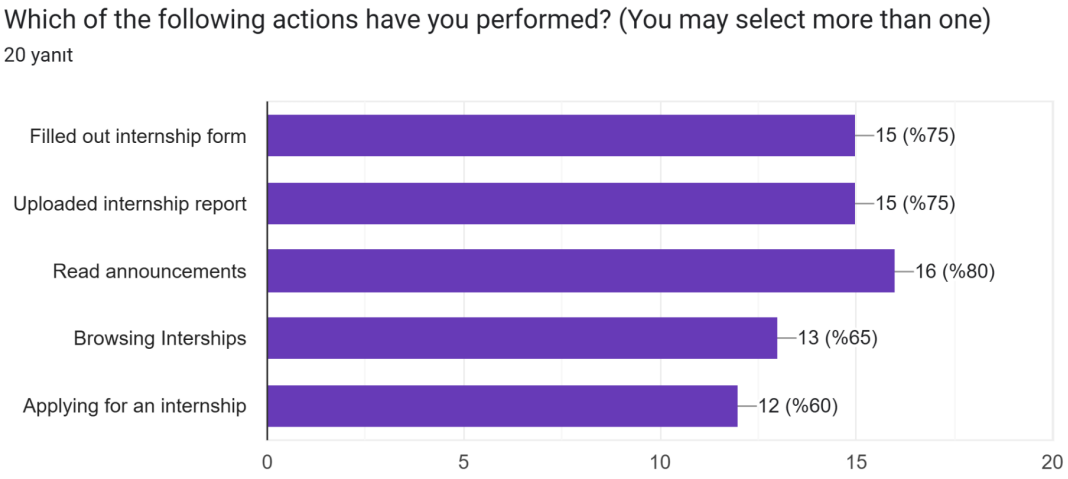


Figure 74: Students Survey Result - 2

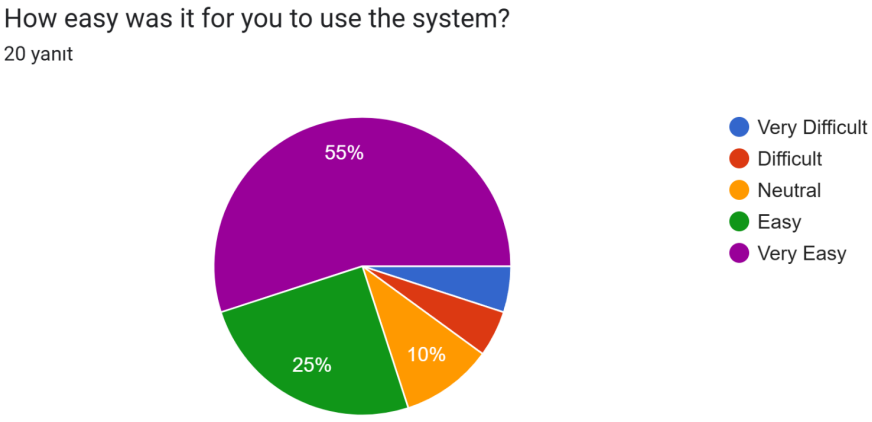


Figure 75 : Students Survey Result - 3

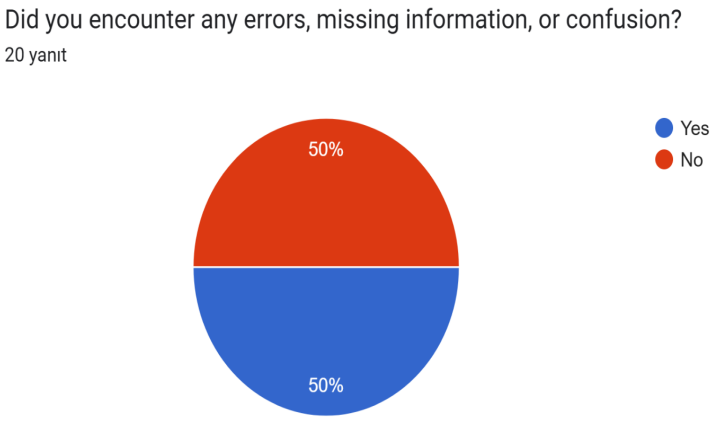


Figure 76 : Students Survey Result - 4

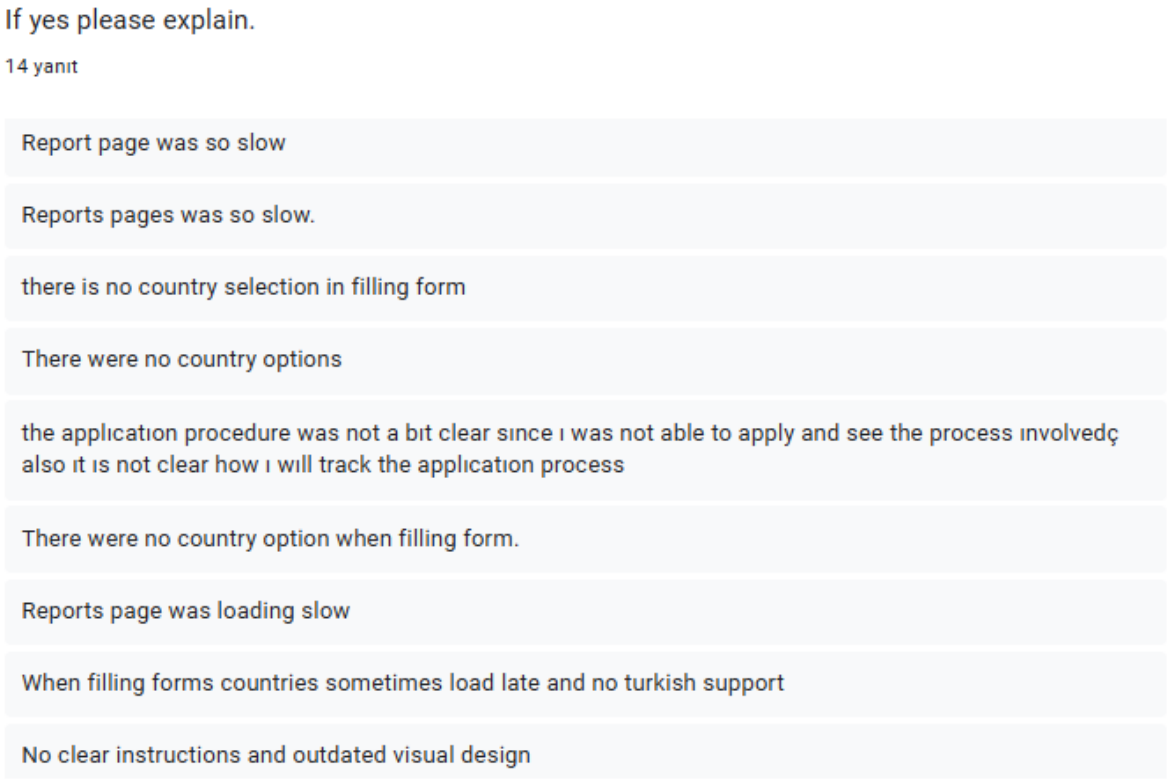


Figure 77 : Students Survey Result - 5

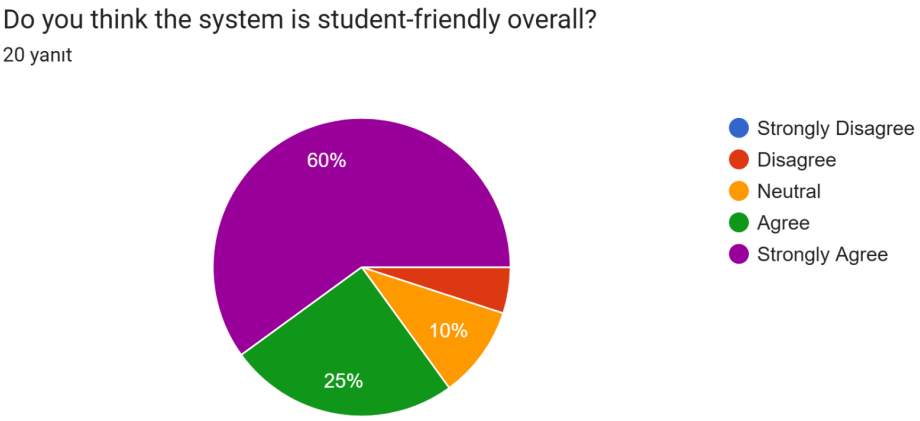


Figure 78 : Students Survey Result - 6

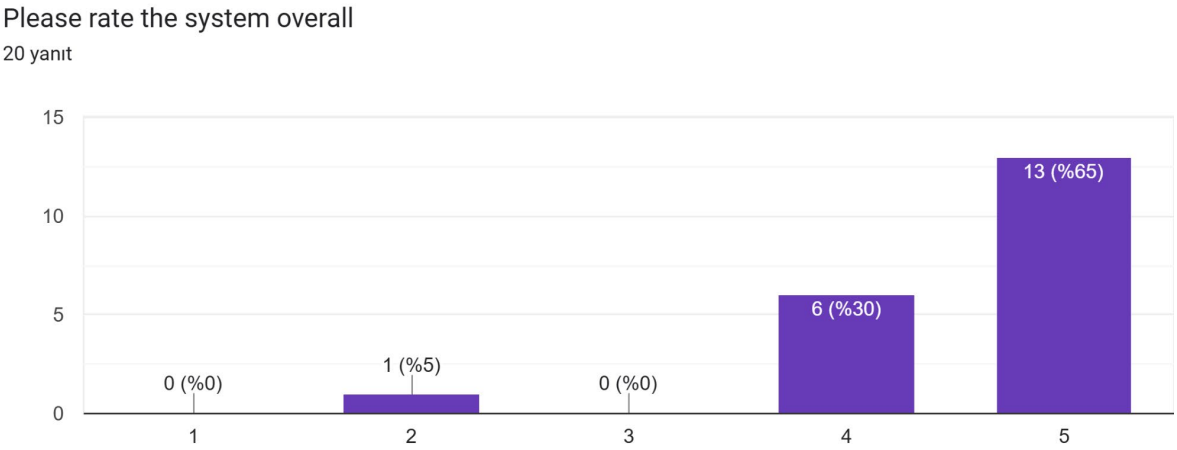


Figure 79 : Students Survey Result - 7

☐ **Based on faced issues, country options, clearer instructional texts and reports page were improved.**

❖ Student Affairs Feedback:

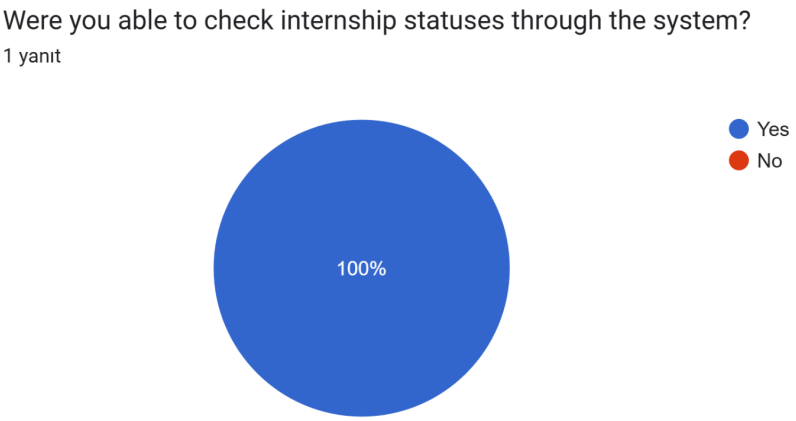


Figure 80: Student Affairs Survey Result - 1

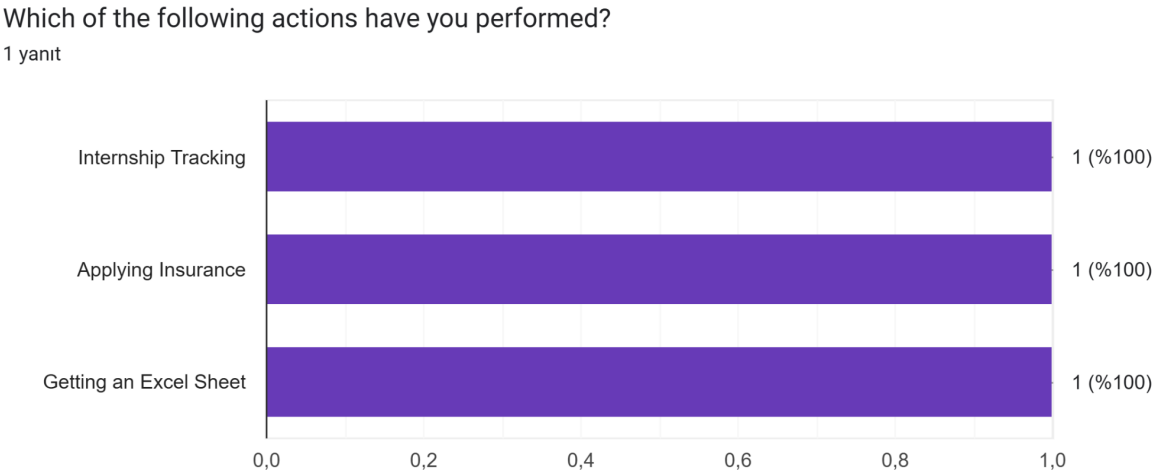


Figure 81: Student Affairs Survey Result - 2

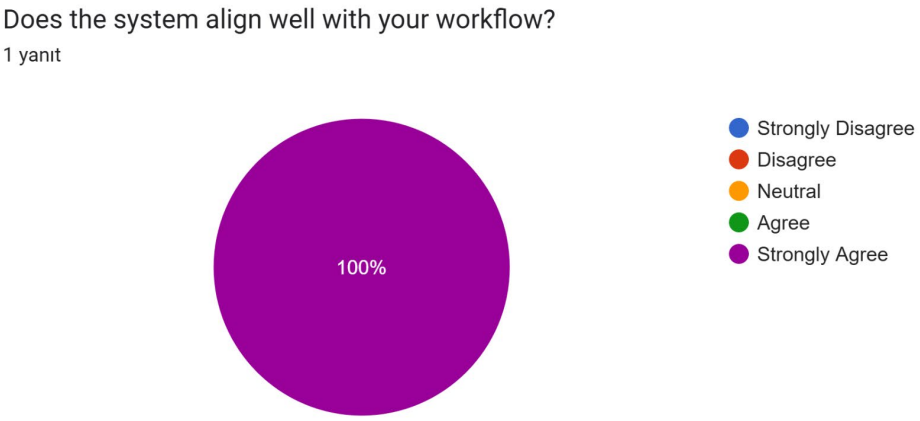


Figure 82: Student Affairs Survey Result - 3

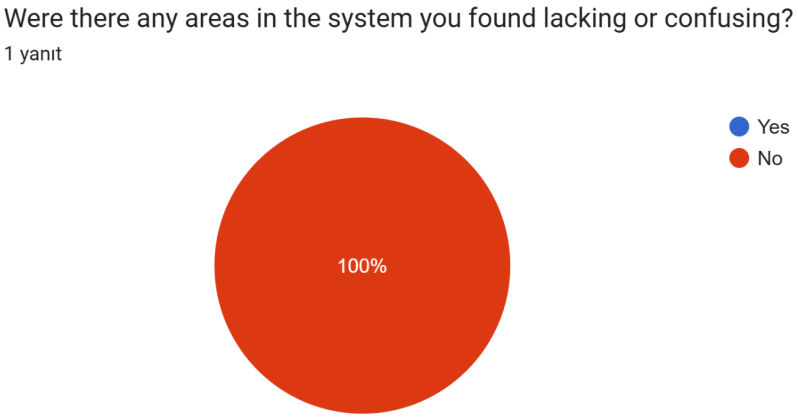


Figure 83 : Student Affairs Survey Result - 4

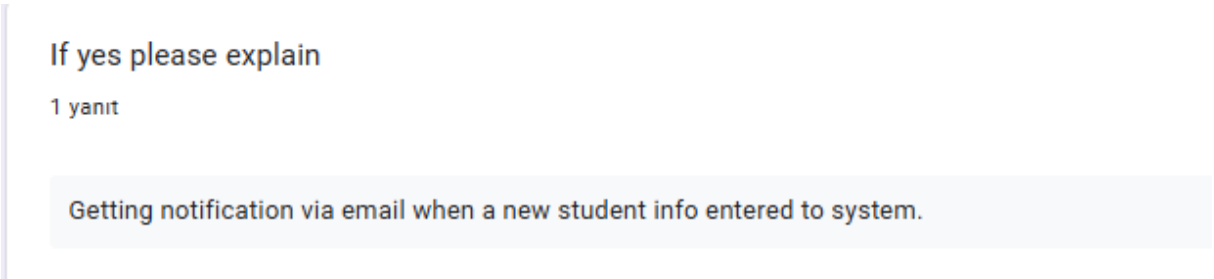


Figure 84 : Student Affairs Survey Result - 5

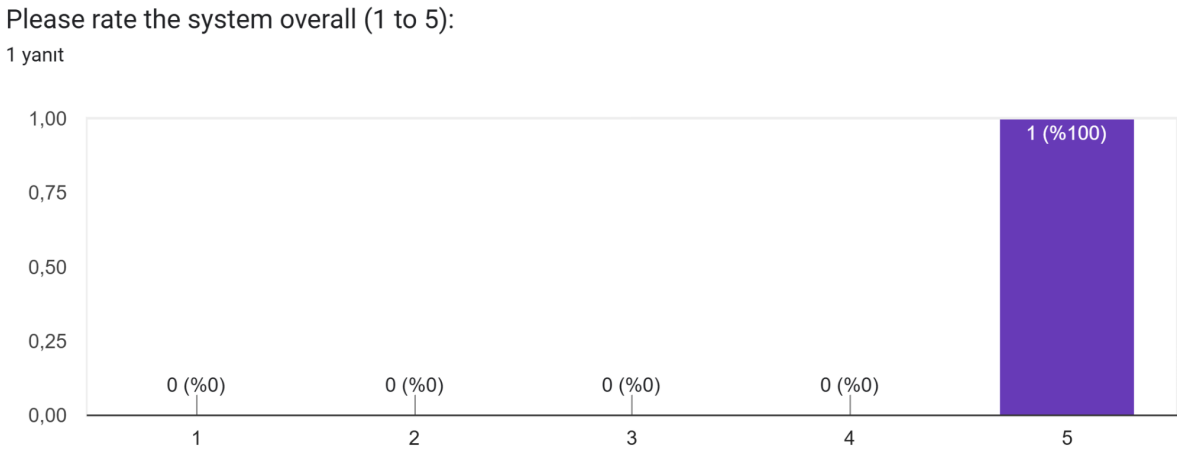


Figure 85 : Student Affairs Survey Result - 6

✓ **The feedback was implemented immediately after testing.**

2.5.4 Test Log

- Location: METU NCC Campus, face-to-face testing.
- Device: Personal laptop (with local server deployment).
- Participants: Surveyed immediately after testing via Google Forms.
- Record Format: Charts, multiple-choice, open-ended responses.

Each participant was observed while using the system and their actions and expressions were noted. Any confusion, hesitation, or feedback was recorded. Their survey responses were later visualized through bar and pie charts, which can be found in the above.

3 Test Tools

For testing purposes, Postman, Apache JMeter, Mockito and Playwright were used to support different types of testing strategies:

Postman was utilized for functional API testing. Automated test scripts were written using Postman's JavaScript-based scripting capabilities in the "Tests" tab. These scripts validated response status codes, JSON body properties (such as assignedInstructor, student, etc.), and response times. Assertions ensured each API met the expected functional behaviors.

Apache JMeter was the main tool for conducting performance and load tests. Scenarios with concurrent users were created to simulate real-world API traffic. JMeter enabled capturing metrics like average response time, error rate, throughput, and standard deviation. This helped us evaluate how each API performs under stress and high concurrency.

Playwright was the main tool for conducting interaction tests. For each case a test case function was carefully written in TypeScript. Critical scenarios involving data exchange and synchronization between frontend and backend were tested in this way.

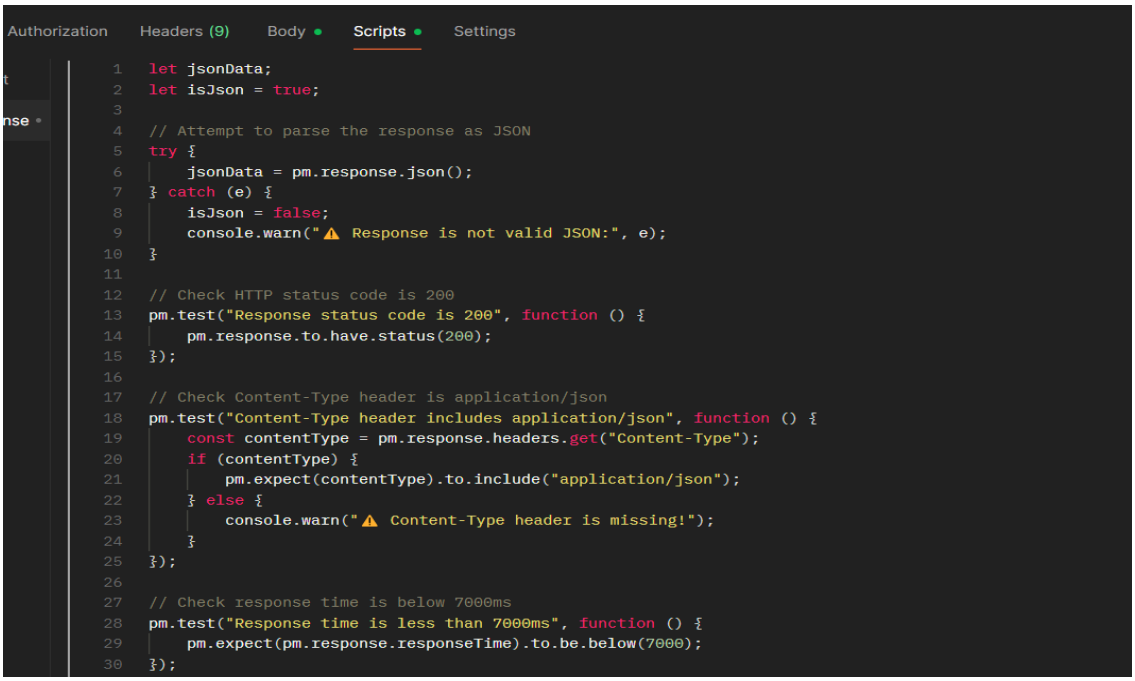
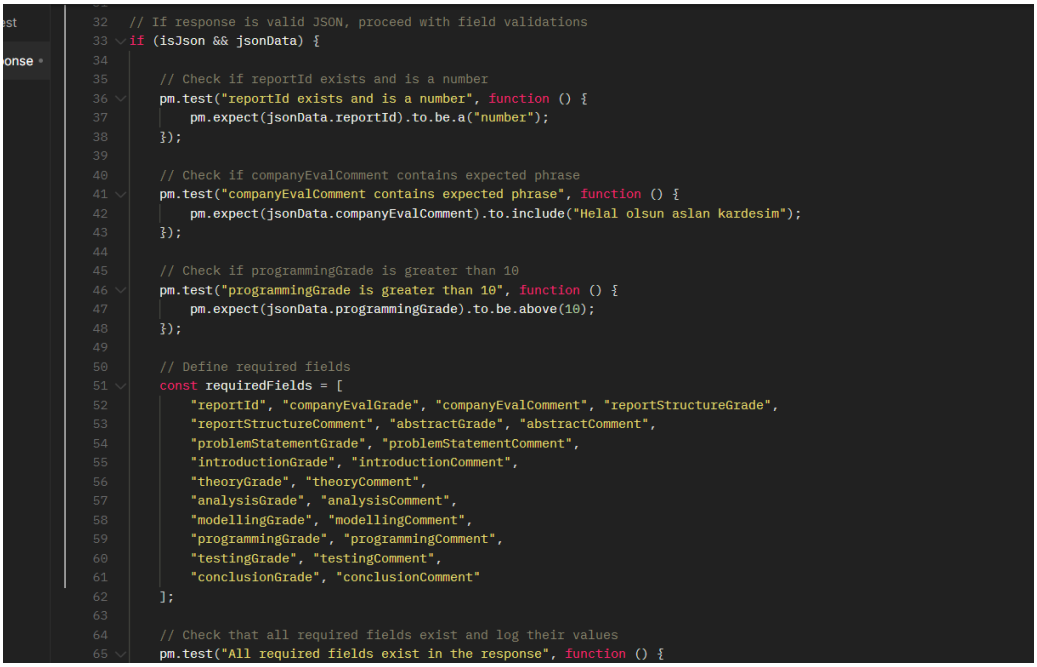


Figure 86: TypeScript Test Codes -1



```
// If response is valid JSON, proceed with field validations
if (isJson && jsonData) {

    // Check if reportId exists and is a number
    pm.test("reportId exists and is a number", function () {
        pm.expect(jsonData.reportId).to.be.a("number");
    });

    // Check if companyEvalComment contains expected phrase
    pm.test("companyEvalComment contains expected phrase", function () {
        pm.expect(jsonData.companyEvalComment).to.include("Helal olsun aslan kardesim");
    });

    // Check if programmingGrade is greater than 10
    pm.test("programmingGrade is greater than 10", function () {
        pm.expect(jsonData.programmingGrade).to.be.above(10);
    });

    // Define required fields
    const requiredFields = [
        "reportId", "companyEvalGrade", "companyEvalComment", "reportStructureGrade",
        "reportStructureComment", "abstractGrade", "abstractComment",
        "problemStatementGrade", "problemStatementComment",
        "introductionGrade", "introductionComment",
        "theoryGrade", "theoryComment",
        "analysisGrade", "analysisComment",
        "modellingGrade", "modellingComment",
        "programmingGrade", "programmingComment",
        "testingGrade", "testingComment",
        "conclusionGrade", "conclusionComment"
    ];

    // Check that all required fields exist and log their values
    pm.test("All required fields exist in the response", function () {
```

Figure 87: TypeScript Test Codes -2

```
        "conclusionGrade", "conclusionComment"
    ];

    // Check that all required fields exist and log their values
    pm.test("All required fields exist in the response", function () {
        requiredFields.forEach(field => {
            pm.expect(jsonData).to.have.property(field);
            console.log(`✅ Field exists: ${field} → ${jsonData[field]}`);
        });
    });

} else {
    console.warn("⚠️ JSON parsing failed. Skipping field validations.");
}
```

Figure 88: TypeScript Test Codes -3

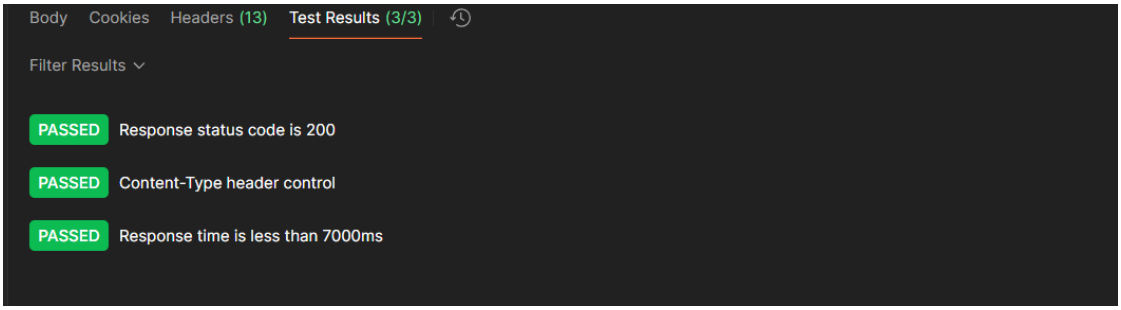


Figure 89 : TypeScript Test Codes -4

In that above Test tools using via Postman we obtain that our conditions satisfy with passed all situations.

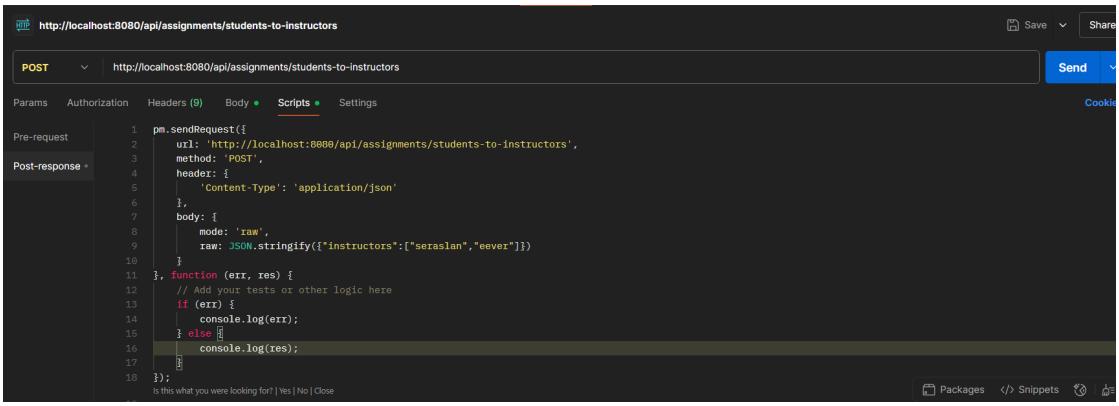


Figure 90: TypeScript Test Codes via Postman -1

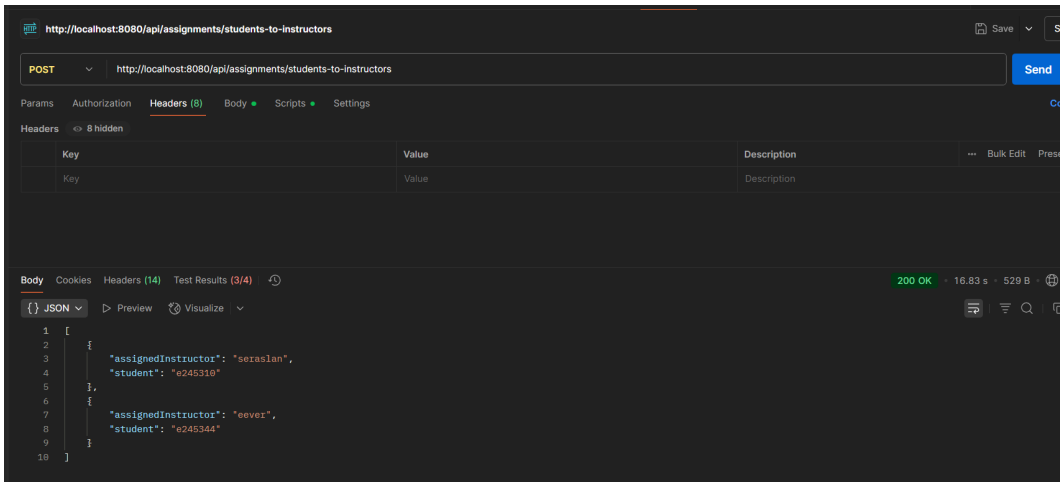


Figure 91: TypeScript Test Codes via Postman -2

In that above also that script enable to pass that all of that satisfied conditions with server response time.

4 Test Incidents

N/A

5 Summary

5.1 Summary of the Tests Passed

Unit Tests

- A total of 14 unit tests were executed successfully.
- Service methods such as saveAnnouncement(), getApprovedTraineeInformationForms(), updateFormStatus(), and approveInsurance()—including repository interactions and email workflows—were verified with JUnit 5 and Mockito.
- All methods returned the expected objects, persisted data correctly, sent emails, and threw no unexpected errors.

Integration Tests

- 19 out of 20 critical scenarios passed without issues.
- End-to-end flows for login, form upload, announcement creation, assignment, report upload, and application-approval worked seamlessly.
- The only failure was TC17 – Student Affairs Insurance Approval: although the backend processes the approval, the frontend does not reflect the updated status until the page is refreshed.

System Tests

- All five end-to-end scenarios (from application submission through report evaluation, announcement viewing, and form download) completed successfully.
- No critical functionality was lost in user flows; only a minor UX issue was noted where revision requests do not appear clearly in the student interface.

Performance Tests

- TC02, TC03, and TC04 passed with 100% success; response times met or closely approached targets, with a low error rate.
- TC01 – Deadline Addition Load Test: under a load of 100 concurrent requests, only 50% succeeded; average response time was ~2.9 s with a high error rate, indicating the need for optimization under high concurrency.

User Acceptance Tests

- In-person sessions with coordinators, students, and student-affairs staff yielded high overall satisfaction.
- Surveys highlighted positive feedback on UI clarity, ease of task flows, and timing of email notifications.
- Minor adjustments—such as adding country selection, improving explanatory text, and refining the report page layout—were implemented on the spot.

5.2 Summary of the Tests Failed

For each failed test, we've broken down the root cause and defined concrete development tasks to resolve the issue.

TC17 – Student Affairs Insurance Approval

Issue:

The backend marks the insurance as approved, but the frontend UI does not reflect this change until the user manually refreshes the page.

Root Cause:

No client-side listener or state update is wired to the approval endpoint.

Improvement Actions:

1) Extend API Response:

- We need to modify the /approveInsurance endpoint to return the updated insurance record (including status and timestamp).

2) State Management Integration:

- In the Angular/React client, dispatch an action (e.g., INSURANCE_APPROVED) upon success.
- Use RxJS Subject (Angular) or Redux middleware (React) to push the updated record into the store.

3) UI Update & Testing:

- Update the insurance-status component to read from the store and re-render automatically.

- Add a unit test to verify that, when the approval API resolves, the UI's status badge switches to "Approved" without refresh.
- Write an end-to-end Cypress/Selenium test covering the full approval flow.

TC01 – Deadline Addition Load Test

- Issue:

Under 100 concurrent "Add Deadline" requests, only 50% succeed; average response time ≈ 2.9 s with many time-outs.

- Root Cause:

Database connection pool exhausted and synchronous processing of heavy validation logic.

- Improvement Actions:

1) Connection Pool Tuning:

- Increase max connections in HikariCP (or equivalent) from 10 → 50.
- Monitor active/pending connection metrics during load tests.

2) Asynchronous Processing:

- Offload non-critical work (e.g., notification email, audit logging) into an async queue (e.g., RabbitMQ or Kafka).
- Keep the HTTP thread focused on minimal validation + persistence.

3) Input Validation Optimization:

- Move schema checks into a lightweight, in-memory validator (e.g., Ajv for JSON) before hitting the DB.

4) Caching & Throttling:

- Introduce an in-memory cache (e.g., Caffeine or Redis) for frequently read reference data during deadline creation.
- Apply rate limiting per user/IP to smooth spikes.

5) Re-Test & Monitor:

- Re-run JMeter with the above changes.

6 References

N/A