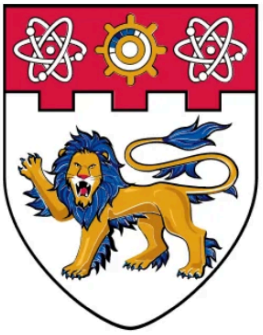


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2006 - Software Engineering
Lab 4 Deliverables (Test Cases)

Lab Group	TDDA
Team Name :	Project PolyQuest
Members	Lam Yan Yee U2323642L
	Dallas Ng Zhi Hao U2323084C
	Malcolm Fong Cheng Hong U2321081B
	Yeo, Ruizhi Carwyn U2322527C
	Khoo Ze Kang U2321266K
	Jerwin Lee Chu Hao U2321487B

1. Black Box Testing

We conducted black-box testing of the frontend by having our backend team test the application to identify potential issues. This approach allowed us to evaluate the frontend purely from a user's perspective, ensuring functionality and usability were aligned with expected user experiences. Our testing focused on exploring user interactions, validating form inputs, checking data display, and confirming that the frontend worked smoothly across different browsers and devices.

Basic security checks, such as trying unexpected inputs to detect vulnerabilities, were also part of our process, along with performance tests to observe how the UI handled varying loads. We documented all issues found and provided detailed reports for the frontend team to review and address. Once fixes were made, regression tests were performed to ensure new updates did not break any existing functionality. This method helped us uncover and resolve frontend issues effectively, fostering a collaborative effort between our backend and frontend teams.

Scenario	Test Case	Equivalence Class	Boundary Value Testing
Sign up with Google	1. Sign up with an existing Google account	Valid: Existing Google account	N/A (No direct boundary)
	2. Sign up with a new Google account	Valid: New Google account	N/A (No direct boundary)
	3. Cancel Google sign-up midway	Invalid: Cancelled sign-up	N/A (No direct boundary)
Sign up with Email and Password	1. Sign up with valid email and password	Valid: Valid email and strong password	Password: Test with minimum (e.g., 7) and maximum length (e.g., 16)
	2. Attempt to sign up with an already registered email	Invalid: Already registered email	Email: Test with minimum length (e.g., 1) and maximum (e.g., 256)
	3. Sign up with a weak password	Invalid: Weak password (e.g., too short)	Password: Test with minimum (e.g., 7)
Sign in with Google	1. Sign in with an existing Google account	Valid: Existing Google account	N/A (No direct boundary)
	2. Attempt sign-in with an unlinked Google account	Invalid: Unlinked Google account	N/A (No direct boundary)

	3. Cancel Google sign-in midway	Invalid: Cancelled sign-in	N/A (No direct boundary)
Sign in with Email and Password	1. Sign in with correct credentials	Valid: Correct email and password	Email: Test with minimum (e.g., 1) and maximum (e.g., 256)
	2. Sign in with incorrect credentials	Invalid: Incorrect email or password	Password: Test with minimum (e.g., 7) and maximum (e.g., 16)
Forget Password	1. Request password reset with a registered email	Valid: Registered email	Email: Test with minimum (e.g., 1) and maximum (e.g., 256)
	2. Request reset with an unregistered email	Invalid: Unregistered email	Email: Test with minimum (e.g., 1) and maximum (e.g., 256)
	3. Complete the password reset process	Valid: Successful reset with new password	Password: Test with minimum (e.g., 7) and maximum (e.g., 16)

1.1 Equivalence class and Boundary Value Testing

1.1.1 Login (Email & Password)

No.	Email Password	Expected Output	Actual Output	Pass / Fail ?
1.	abc@gmail.com "Empty password"	"Please fill in all fields"	"Please fill in all fields"	Pass
2.	"Empty email" abc123	"Please fill in all fields"	"Please fill in all fields"	Pass
3.	abc@ abc123	"Please enter a part following '@','abc@' is incomplete."	"Please enter a part following '@','abc@' is incomplete."	Pass
4.	abc abc123	"Please include an '@' in the email address. 'Abc' is missing an '@'.	"Please include an '@' in the email address. 'Abc' is missing an '@'.	Pass
5.	"Non-existing email" abc123	Firebase: Error (auth/invalid-credential).	Firebase: Error (auth/invalid-credential).	Pass
5.	abc@gmail.com "Wrong password"	Firebase: Error (auth/invalid-credential).	Firebase: Error (auth/invalid-credential).	Pass

6.	abc@gmail.com “Correct password”	Login Successful	Login Successful	Pass

1.1.2 Login (Google)

No.	Choose Gmail	Expected Output	Actual Output	Pass / Fail ?
1.	abc@gmail.com	Login Successful	Login Successful	Pass

1.1.3 Sign Up

No.	First Name Last Name Email Password Confirm Password	Expected Output	Actual Output	Pass / Fail ?
1.	“Empty First Name” “Empty Last Name” “Empty Email” “Empty Password” “Empty Confirm Password”	“Please fill in all fields”	“Please fill in all fields”	Pass
2.	“Empty First Name” “Empty Last Name” abc@gmail.com “Empty Password” “Empty Confirm Password”	“Please fill in all fields”	“Please fill in all fields”	Pass
3.	“abc” “def” abc@gmail.com “abcdef” “abcdefgh”	“Passwords do not match”	“Passwords do not match”	Pass

4.	“abc” “def” abc@gmail.com “abcdef” “abcdef”	“Passwords is weak”	“Passwords is weak”	Pass
5.	“abc” “def” abc@gmail.com “Abcdef!@” “Abcdef!@”	“Login Successfully”	“Login Successfully”	Pass

1.1.4 Forgot password

No.	Email	Expected Output	Actual Output	Pass / Fail ?
1.	“abc”	“Please include an ‘@’ in the email address. ‘Abc’ is missing an ‘@’.	“Please include an ‘@’ in the email address. ‘Abc’ is missing an ‘@’.	Pass
2.	“abc@”	“Please enter a part following ‘@’,’abc@’ is incomplete.”	“Please enter a part following ‘@’,’abc@’ is incomplete.”	Pass
3.	“Actual Email”	“Please check your email to reset your password.”	“Please check your email to reset your password.”	Pass

1.1.5 Change Password

No.	Old Password New Password Confirm Password	Expected Output	Actual Output	Pass / Fail ?
1.	“Correct Password” “Abcd!23” “Abcd!23”	"Password must be at least 8 characters long."	"Password must be at least 8 characters long."	Pass

2.	“Correct Password” “abcd!234” “abcd!234”	"Password must contain at least one uppercase letter."	"Password must contain at least one uppercase letter."	Pass
3.	“Correct Password” “Abcd!efg” “Abcd!efg”	"Password must contain at least one number."	"Password must contain at least one number."	Pass
	“Correct Password” “Abcdefg1” “Abcdefg1”	"Password must contain at least one symbol."	"Password must contain at least one symbol."	Pass
	“Correct Password” “Abcd!123” “Abcd!1234”	"Passwords do not match."	"Passwords do not match."	Pass
	“Correct Password” “Abcd!123” “Abcd!123”	"Password updated successfully!"	"Password updated successfully!"	Pass

1.1.6 Bookmark

No.	Courses Selected	Expected Output	Actual Output	Pass / Fail ?
1.	“Not Selected”	Not Displayed	Not Displayed	Pass
2.	“Selected”	Displayed	Displayed	Pass

1.1.7 Feedback

No.	Survey Questions Done	Expected Output	Actual Output	Pass / Fail ?
1.	“Did not do all 8 questions”	“Please complete the following question(s):”	“Please complete the following question(s):”	Pass
2.	“Did all 8 questions”	Redirect to ‘Thank You’ page	Redirect to ‘Thank You’ page	Pass

1.1.8 Test

No.	Test Questions Done	Expected Output	Actual Output	Pass / Fail ?
1.	“Did not do all 25”	“Please complete the following question(s):”	“Please complete the following question(s):”	Pass
2.	“Did all 25 questions”	ML ran and redirect to Recommended page	ML ran and redirect to Recommended page	Pass

1.1.9 Search

No.	Search Input	Expected Output	Actual Output	Pass / Fail ?
1.	“Buisness”	No courses generated	No courses generated	Pass
2.	“Engineering”	“Diploma in Infocomm & Media Engineering Diploma in Biomedical Engineering . . . Common Engineering Programme”	“Diploma in Infocomm & Media Engineering Diploma in Biomedical Engineering . . . Common Engineering Programme”	Pass
3.	“Bi”	“Diploma in Biomedical Engineering Diploma in Biotechnology . . . Diploma in Biologics & Process Technology”	“Diploma in Biomedical Engineering Diploma in Biotechnology . . . Diploma in Biologics & Process Technology”	Pass
4.	“Business” “Bus”	“Diploma in Business Studies Diploma in Business Administration . . . Diploma in Business Information Systems”	“Diploma in Business Studies Diploma in Business Administration . . . Diploma in Business Information Systems”	Pass

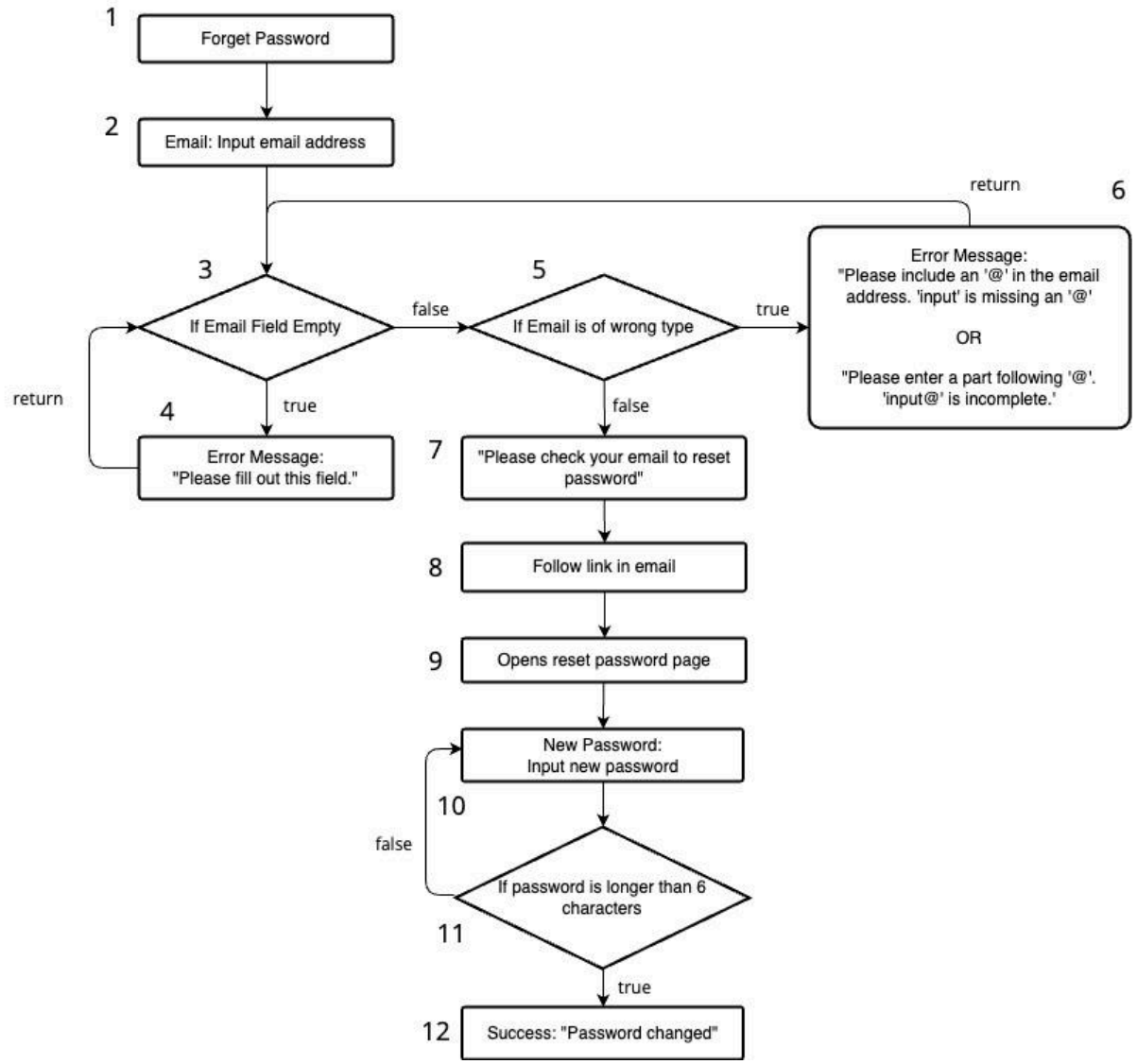
2. White Box Testing

For white-box testing, we focused on testing the APIs from the frontend due to time constraints. This allowed us to ensure that the data flows correctly between the frontend and backend, verifying that API calls were correctly structured, responded as expected, and handled various response states properly, including success, failure, and edge cases. Our testing confirmed that data validation, error handling, and appropriate feedback mechanisms were effectively in place.

However, if we had more time, we would extend our white-box testing to include more comprehensive checks such as code coverage analysis, testing internal logic and conditions, validating function calls and paths, and ensuring that edge cases within the frontend logic were handled. Additionally, we would delve deeper into testing state management, component-level data flow, and performance optimisations to guarantee that the code was robust, efficient, and maintainable. This would help ensure that all logical paths within the application are functioning as intended and improve overall code reliability and security.

2.1 Basis Path Testing

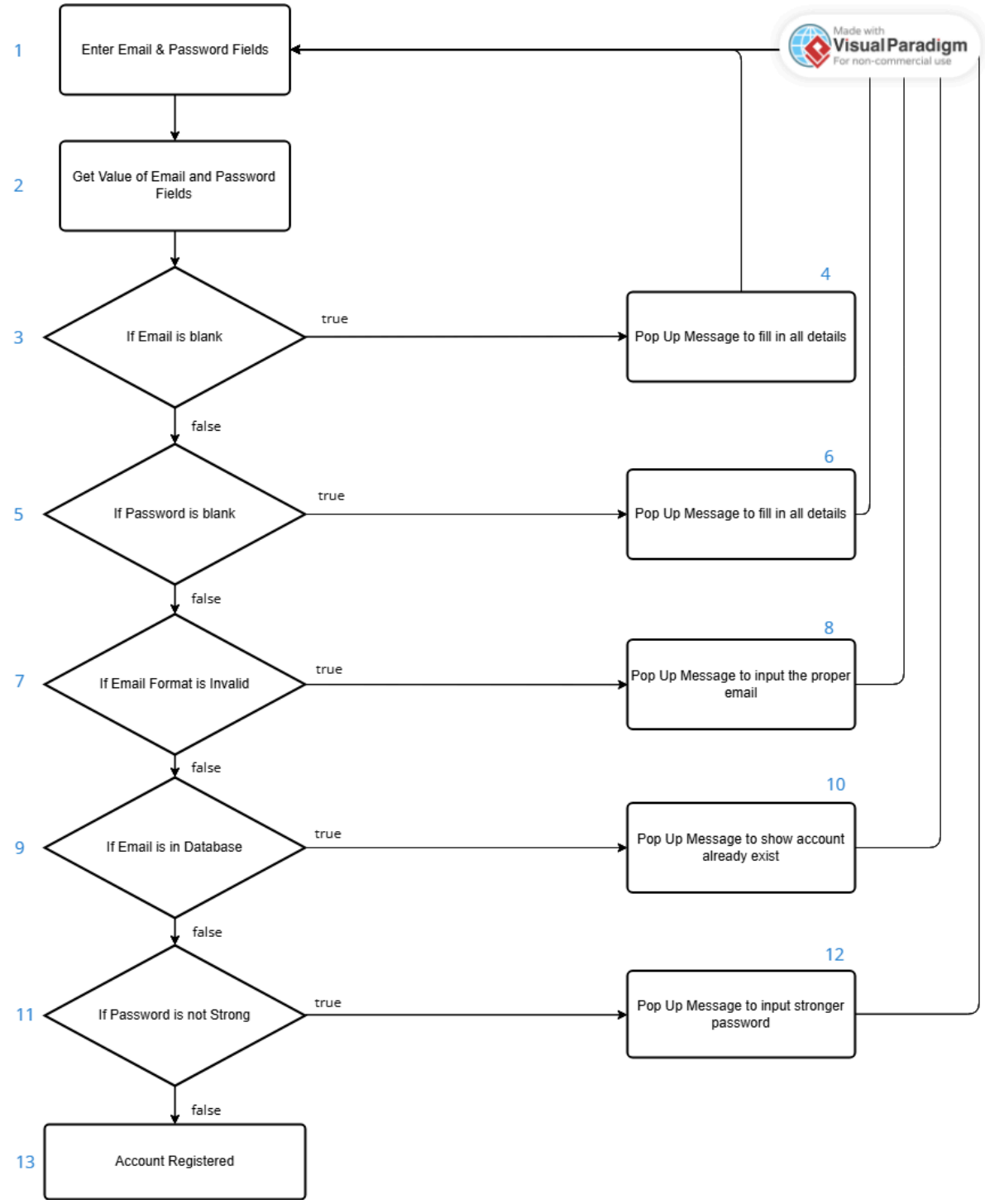
2.1.1 Forget Password



CC = 5 Binary Decision points + 1 = 4

No.	Path	Email	Email Field Empty	Email Format Invalid	NewPassword	NewPassword longer than 6 characters?
Path 1	1,2,3,5,7,9,11,13	rocky123@gmail.com	No	No	newpassword123	Yes
Path 2	1,2,3,4,1,2,3,5,7,9,11,13	Blank	Yes	No	newpassword456	Yes
Path 3	1,2,3,5,6,1,2,3,5,7,9,11,13	invalid-email-format	No	Yes	newpassword789	Yes
Path 4	1,2,3,5,7,9,10,9,11,13	rocky123@gmail.com	No	No	short	No

2.1.2 Registration for an Account



CC = 5 Binary Decision points + 1 = 6

No.	Path	Email	Password	Email Format Invalid	Email is Registered	Password is not strong
Path 1	1,2,3,5,7,9,11,13	rocky123@gmail.com	Rocky123!	No	No	No
Path 2	1,2,3,4,1,2,3,5,7,9,11,13	Blank	Rocky123!	-	-	-
Path 3	1,2,3,5,6,1,2,3,5,7,9,11,13	rocky123@gmail.com	Blank	-	-	-
Path 4	1,2,3,5,7,8,1,2,3,5,7,9,11,13	rocky123	Rocky123!	Yes	-	-
Path 5	1,2,3,5,7,9,10,1,2,3,5,7,9,11,13	rocky123@gmail.com	Rocky123!	No	Yes	-
Path 6	1,2,3,5,7,9,11,12,1,2,3,5,7,9,11,13	rocky123@gmail.com	Rocky123!	No	No	Yes

2.2 Manual Testing

2.2.1 Academic Interest

```
describe("Academic Interest API Tests", () => {
  const studentId = 49; // Example student ID for testing
  const resultData = [{ interestname: "Languages", interested: true, student_id: studentId }]; // Example input for creating interest results
  const mockResult = { id: 1, interestName: "Languages", interested: true, student_id: studentId }; // Mocked response
  const mockInterestData = [
      { id: 1, interestName: "Languages", interested: true, student_id: studentId },
      { id: 2, interestName: "Mathematics", interested: false, student_id: studentId },
    ];

  beforeEach(() => {
      vi.clearAllMocks(); // Clear mocks before each test to prevent interference from previous tests
    });

  // Test for successful creation of interest record
  test("Create interest record - success", async () => {
      (post as vi.Mock).mockResolvedValue({ success: true, data: mockResult });

      const createdInterestRecord = await createAcadInterest(resultData, studentId);
      expect(createdInterestRecord).toBe(true); // Expect `true` on success
      expect(post).toHaveBeenCalledTimes(resultData.length);
    });

// Test for failure case when creating interest record
test("Create interest record - failure", async () => {
  (post as vi.Mock).mockResolvedValue({ success: false, error: "Failed to create interest" });

  const createdInterestRecord = await createAcadInterest(resultData, studentId);
  expect(createdInterestRecord).toBeNull(); // Expect `null` on failure
  expect(post).toHaveBeenCalledTimes(resultData.length);
});
```

Figure 1. Create Interest Record

```
// Test for successfully fetching academic interests
test("Get academic interests - success", async () => {
  // Mock a successful get response with interest data
  (get as vi.Mock).mockResolvedValue({ success: true, data: mockInterestData });

  const interests = await getAcadInterest(); // Fetch interest data
  expect(interests).toHaveLength(2); // Expect two interest records to be returned
  expect(interests[0].interestName).toBe("Languages"); // Check if the first interest name matches
  expect(interests[1].interestName).toBe("Mathematics"); // Check if the second interest name matches
});

// Test for failure case when fetching academic interests
test("Get academic interests - failure", async () => {
  // Mock a failure response
  (get as vi.Mock).mockResolvedValue({ success: false });

  const interests = await getAcadInterest(); // Fetch interest data
  expect(interests).toEqual([]); // Expect an empty array to be returned on failure
});
```

Figure 2. Get Interest

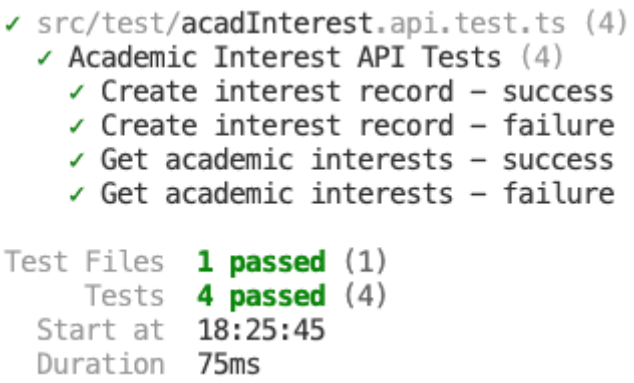


Figure 3. Interest Test Case Result

2.2.2 Academic Result

```
describe("Academic Results API Tests", () => {
  const studentId = 49 // Example student ID for testing
  const resultData = [
    { subjectname: "Geography", grade: "A1", student_id: studentId }
  ] // Example input for creating academic results
  const mockResult = {
    id: 1,
    subjectName: "Geography",
    grade: "A1",
    student_id: studentId
  } // Mocked result response for API responses

  beforeEach(() => {
    vi.clearAllMocks() // Clear mocks before each test to prevent interference from previous tests
  })

  // Test for successful creation of academic results
  test("Create academic results - success", async () => {
    // Mock successful response for post request
    ;(post as vi.Mock).mockResolvedValue({ success: true, data: mockResult })

    const createdResults = (await createAcadResult(
      resultData,
      studentId
    )) as any // Call function to create results
    expect(createdResults).toHaveLength(1) // Expect one result to be created
    expect(createdResults[0].subjectName).toBe("Geography") // Check if the subject name matches
  })

  // Test for failure case when creating academic results
  test("Create academic results - failure", async () => {
    // Mock a failure response
    ;(post as vi.Mock).mockResolvedValue({ success: false })

    const createdResults = await createAcadResult(resultData, studentId) // Call function to create results
    expect(createdResults).toEqual([]) // Expect an empty array to be returned on failure
  })
```

Figure 4. Create Academic result

```
// Test for successfully fetching academic results for a student
test("Get academic results - success", async () => {
  // Mock successful get request response with academic results
  ;(get as vi.Mock).mockResolvedValue({ success: true, data: [mockResult] })

  const results = await getAcadResult(studentId) // Fetch results for the student
  expect(results).toHaveLength(1) // Expect one result to be returned
  expect(results[0].grade).toBe("A1") // Check if the grade matches the expected value
})

// Test for failure case when fetching academic results
test("Get academic results - failure", async () => {
  // Mock a failure response
  ;(get as vi.Mock).mockResolvedValue({ success: false })

  const results = await getAcadResult(studentId) // Fetch results for the student
  expect(results).toEqual([]) // Expect an empty array to be returned on failure
})
```

Figure 5. Get Academic result

```
// Test for successfully fetching academic subjects
test("Get academic subjects - success", async () => {
  const mockSubjects = [
    { subjectName: "Geography" },
    { subjectName: "Literature" }
  ] // Mocked data for academic subjects
  ;(get as vi.Mock).mockResolvedValue({ success: true, data: mockSubjects }) // Mock a successful response for subjects

  const subjects = await getAcadSubject() // Fetch academic subjects
  expect(subjects).toHaveLength(2) // Expect two subjects to be returned
  expect(subjects[0].subjectName).toBe("Geography") // Check if the first subject name matches
})

// Test for failure case when fetching academic subjects
test("Get academic subjects - failure", async () => {
  // Mock a failure response
  ;(get as vi.Mock).mockResolvedValue({ success: false })

  const subjects = await getAcadSubject() // Fetch academic subjects
  expect(subjects).toEqual([]) // Expect an empty array to be returned on failure
})
```

Figure 6. Get Academic Subjects

```
// Test for successfully updating an academic result
test("Update academic result - success", async () => {
  const updatedData = { subjectName: "Geography", grade: "A2" } // Data to update the result with
  ;(patch as vi.Mock).mockResolvedValue({
    success: true,
    data: { ...mockResult, grade: "A2" }
  }) // Mock a successful update response

  const updatedResult = await updateAcadResult(1, studentId, updatedData) // Call function to update result
  expect(updatedResult).toBeDefined() // Ensure the updated result is defined
  expect(updatedResult.grade).toBe("A2") // Check if the grade has been updated correctly
})

// Test for failure case when updating an academic result
test("Update academic result - failure", async () => {
  // Mock a failure response
  ;(patch as vi.Mock).mockResolvedValue({ success: false })

  const updatedResult = await updateAcadResult(1, studentId, {
    subjectName: "Geography",
    grade: "A2"
  }) // Call function to update result
  expect(updatedResult).toBeNull() // Expect null to be returned on failure
})
```

Figure 7. Update Academic Results

```
// Test for successfully deleting an academic result
test("Delete academic result - success", async () => {
  ;(del as vi.Mock).mockResolvedValue({
    success: true,
    data: { message: "Deleted successfully" }
  }) // Mock a successful delete response

  const deletedResult = await deleteAcadResult(1) // Call function to delete result
  expect(deletedResult).toBeDefined() // Ensure the delete response is defined
  expect(deletedResult).toEqual({ message: "Deleted successfully" }) // Check if the response matches the expected message
})

// Test for failure case when deleting an academic result
test("Delete academic result - failure", async () => {
  // Mock a failure response
  ;(del as vi.Mock).mockResolvedValue({ success: false })

  const deletedResult = await deleteAcadResult(1) // Call function to delete result
  expect(deletedResult).toBeNull() // Expect null to be returned on failure
})
)
```

Figure 8. Delete Academic Results

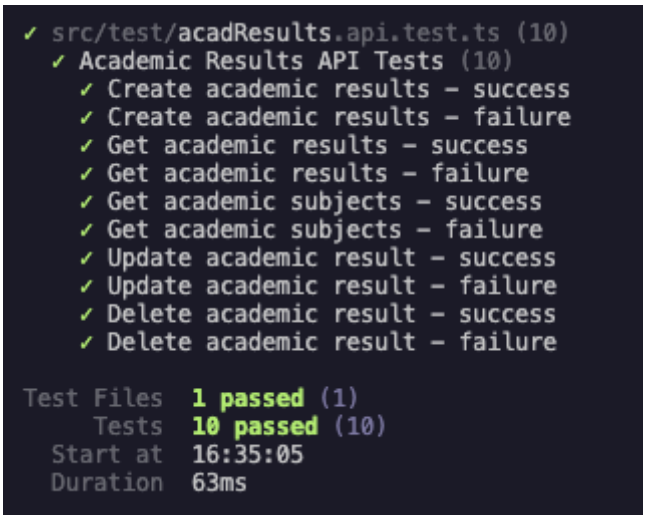


Figure 9. Academic Test Case Result

2.2.3 Student

```
test("Get all students", async () => {
  const students = await getStudents()
  expect(students.length).toBeGreaterThan(0)
})

test("Get student by email", async () => {
  const student = await getStudentByEmail("kh0008ng@gmail.com")
  expect(student).toBeDefined()
})

test("Get student by id", async () => {
  const student = await getStudent(49)
  expect(student).toBeDefined()
})
```

Figure 10. Student Test Cases

```
✓ src/test/student.test.ts (3)
  ✓ Get all students
  ✓ Get student by email
  ✓ Get student by id

Test Files  1 passed (1)
     Tests  3 passed (3)
  Start at  16:27:49
  Duration  253ms
```

Figure 11. Student Test Cases Result

2.2.4 Test

```
describe("Test API Tests", () => {
  const studentId = 49; // Example student ID for testing
  const testId = 1;
  const questionId = 10;
  const responseValue = 5;
  const questionData = [
    { id: 1, questionText: "How much do you enjoy subjects like Math and Science? (0 = Not at all, 5 = Very much)", createdat: "1730038282000", updatedat: null },
    { id: 2, questionText: "How interested are you in pursuing Engineering or Technical fields? (0 = Not interested, 5 = Very interested)", createdat: "1730038283000", updatedat: null },
  ];

  beforeEach(() => {
    vi.clearAllMocks(); // Clear mocks before each test to prevent interference from previous tests
  });

  // Test for creating test result - success
  test("Create test result - success", async () => {
    // Mock successful response for post request
    (post as vi.Mock).mockResolvedValue({ success: true });

    const result = await createTestResult(testId, questionId, responseValue);
    expect(result).toBeUndefined(); // Expect no error on success
    expect(post).toHaveBeenCalledTimes(1); // Ensure the post function is called once
  });

  // Test for creating test result - failure
  test("Create test result - failure", async () => {
    // Mock a failure response
    (post as vi.Mock).mockResolvedValue({ success: false });

    const result = await createTestResult(testId, questionId, responseValue);
    expect(result).toBeNull(); // Expect `null` on failure
  });
```

Figure 12. Create Test Result

```
// Test for creating testId record - success
test("Create testId record - success", async () => {
  (post as vi.Mock).mockResolvedValue({ success: true });

  const result = await createTestId(studentId);
  expect(result).toBeUndefined(); // Expect no error on success
  expect(post).toHaveBeenCalledTimes(1); // Ensure the post function is called once
});

// Test for creating testId record - failure
test("Create testId record - failure", async () => {
  (post as vi.Mock).mockResolvedValue({ success: false });

  const result = await createTestId(studentId);
  expect(result).toBeNull(); // Expect `null` on failure
});
```

Figure 13. Create TestId

```
// Test for getting questions - success
test("Get questions - success", async () => {
  (get as vi.Mock).mockResolvedValue({ success: true, data: questionData });

  const questions = await getQuestions();
  expect(questions).toHaveLength(2); // Expect two questions to be returned
  expect(questions[0].questionText).toBe("How much do you enjoy subjects like Math and Science? (0 = Not at all, 5 = Very much)");
});

// Test for getting questions - failure
test("Get questions - failure", async () => {
  (get as vi.Mock).mockResolvedValue({ success: false });

  const questions = await getQuestions();
  expect(questions).toEqual([]); // Expect an empty array on failure
});
```

Figure 14. Get Question

```
// Test for getting testId - success
test("Get testId - success", async () => {
  (get as vi.Mock).mockResolvedValue({ success: true, data: { id: testId } });

  const result = await gettestId(studentId);
  expect(result).toBe(testId); // Expect to receive the testId
});

// Test for getting testId - failure
test("Get testId - failure", async () => {
  (get as vi.Mock).mockResolvedValue({ success: false });

  const result = await gettestId(studentId);
  expect(result).toBeNull(); // Expect `null` on failure
});
```

Figure 15. Get TestId

```
// Test for updating test completion - success
test("Update test complete - success", async () => {
  (patch as vi.Mock).mockResolvedValue({ success: true, data: { complete: true } });

  const result = await updateTestComplete(testId, true);
  expect(result).toEqual({ complete: true });
  expect(patch).toHaveBeenCalledTimes(1); // Ensure the patch function is called once
});

// Test for updating test completion - failure
test("Update test complete - failure", async () => {
  (patch as vi.Mock).mockResolvedValue({ success: false });

  const result = await updateTestComplete(testId, true);
  expect(result).toBeNull(); // Expect `null` on failure
});
```

Figure 16. Update Test Complete

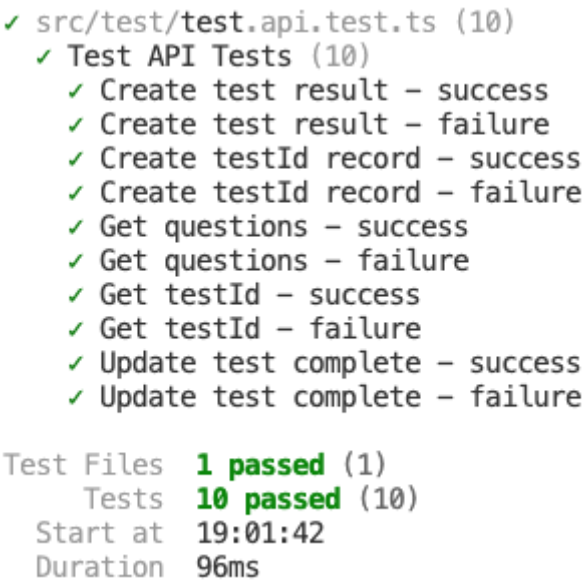


Figure 17. Test Case Result

2.2.5 Bookmark

```
describe("Bookmark API Tests", () => {
  const studentId = 49; // Example student ID for testing
  const courseId = 101; // Example course ID for testing
  const mockBookmark = {
    id: 1,
    studentID: studentId,
    course: courseId,
    createdAt: new Date(),
    updatedAt: new Date()
  }; // Mocked bookmark response for API responses

  beforeEach(() => {
    vi.clearAllMocks(); // Clear mocks before each test to prevent interference from previous tests
  });

  // Test for successfully creating a bookmark
  test("Create bookmark - success", async () => {
    // Mock successful response for post request
    (post as vi.Mock).mockResolvedValue({ success: true, data: mockBookmark });

    const createdBookmark = await createBookmark(studentId, courseId); // Call function to create bookmark
    expect(createdBookmark).toBeDefined(); // Expect a bookmark to be created
    expect(createdBookmark?.studentID).toBe(studentId); // Check if the student ID matches
    expect(createdBookmark?.course).toBe(courseId); // Check if the course ID matches
  });

  // Test for failure case when creating a bookmark
  test("Create bookmark - failure", async () => {
    // Mock a failure response
    (post as vi.Mock).mockResolvedValue({ success: false });

    const createdBookmark = await createBookmark(studentId, courseId); // Call function to create bookmark
    expect(createdBookmark).toBeNull(); // Expect null to be returned on failure
  });
```

Figure 18. Create Bookmark

```
// Test for successfully fetching bookmarks for a student
test("Get bookmarks by student - success", async () => {
  // Mock successful get request response with bookmarks
  (get as vi.Mock).mockResolvedValue({ success: true, data: [mockBookmark] });

  const bookmarks = await getBookmarksByStudent(studentId); // Fetch bookmarks for the student
  expect(bookmarks).toHaveLength(1); // Expect one bookmark to be returnedq
  expect(bookmarks[0].studentID).toBe(studentId); // Check if the student ID matches
  expect(bookmarks[0].course).toBe(courseId); // Check if the course ID matches
});

// Test for failure case when fetching bookmarks
test("Get bookmarks by student - failure", async () => {
  // Mock a failure response
  (get as vi.Mock).mockResolvedValue({ success: false });

  const bookmarks = await getBookmarksByStudent(studentId); // Fetch bookmarks for the student
  expect(bookmarks).toEqual([]); // Expect an empty array to be returned on failure
});
```

Figure 19. Get Bookmarks By Student ID

```
  // Test for successfully deleting a bookmark by student and course ID
  test("Delete bookmark - success", async () => {
    // Mock successful delete response
    (del as vi.Mock).mockResolvedValue({ success: true });

    const deleteResult = await deleteBookmarkByStudent(studentId, courseId); // Call function to delete bookmark
    expect(deleteResult).toBe(true); // Expect true to be returned on successful delete
  });

  // Test for failure case when deleting a bookmark
  test("Delete bookmark - failure", async () => {
    // Mock a failure response
    (del as vi.Mock).mockResolvedValue({ success: false });

    const deleteResult = await deleteBookmarkByStudent(studentId, courseId); // Call function to delete bookmark
    expect(deleteResult).toBe(false); // Expect false to be returned on failure
  });
});
```

Figure 20. Delete Bookmarks

2.2.6 Courses

```
describe("Course API Tests", () => {
  // Test data
  const course: Course = {
    id: 1,
    courseName: "Software Engineering",
    schoolName: "Tech University",
    score: 85,
    careerProspect: ["Developer", "Project Manager"],
    keySkills: ["Programming", "Problem Solving", "Project Management"],
    intake: 2024,
    courseDescription: "An in-depth course on software engineering principles.",
    facultyName: "Dr. John Doe",
  };

  const courseResponse = { success: true, data: course }; // Mocked response for creating a course
  const courseListResponse = { success: true, data: [course] }; // Mocked list of courses response

  beforeEach(() => {
    vi.clearAllMocks(); // Clear mocks before each test
  });
```

Figure 21. Course Test Data

```
  // Test case for successfully creating a course
  test("Create course - success", async () => {
    // Mock a successful response from the post request
    (post as vi.Mock).mockResolvedValueOnce(courseResponse);

    const result = await createCourse(course); // Call the createCourse function
    expect(result).toEqual(course); // Ensure the result matches the course data
    expect(post).toHaveBeenCalledWith(expect.objectContaining({
      url: "/course",
      body: course,
    })); // Verify that the correct URL and body were passed to the API
  });

  // Test case for creating a course - failure
  test("Create course - failure", async () => {
    // Mock a failure response
    (post as vi.Mock).mockResolvedValueOnce({ success: false });

    const result = await createCourse(course); // Call the createCourse function
    expect(result).toBeNull(); // Ensure null is returned on failure
  });
```

Figure 22. Create Course

```
// Test case for getting a course by ID
test("Get course by ID - success", async () => {
  // Mock a successful response from the get request
  (get as vi.Mock).mockResolvedValueOnce(courseResponse);

  const result = await getCourseById(1); // Call the getCourseById function
  expect(result).toEqual(course); // Ensure the result matches the course data
  expect(get).toHaveBeenCalledWith(expect.objectContaining({
    url: "/course/1",
  })); // Verify that the correct URL was used for the API call
});

// Test case for getting a course by ID - failure
test("Get course by ID - failure", async () => {
  // Mock a failure response
  (get as vi.Mock).mockResolvedValueOnce({ success: false });

  const result = await getCourseById(1); // Call the getCourseById function
  expect(result).toBeNull(); // Ensure null is returned on failure
});
```

Figure 23. Get Course By ID

```
  // Test case for getting all courses - success
  test("Get courses - success", async () => {
    // Mock a successful response for getting the list of courses
    (get as vi.Mock).mockResolvedValueOnce(courseListResponse);

    const result = await getCourses(); // Call the getCourses function
    expect(result).toEqual([course]); // Ensure the result matches the course list
    expect(get).toHaveBeenCalledWith(expect.objectContaining({
      url: "/course",
    })); // Verify that the correct URL was used for the API call
  });

  // Test case for getting all courses - failure
  test("Get courses - failure", async () => {
    // Mock a failure response
    (get as vi.Mock).mockResolvedValueOnce({ success: false, data: [] });

    const result = await getCourses(); // Call the getCourses function
    expect(result).toEqual([]); // Ensure an empty array is returned on failure
  });

  // Test case for exception handling when fetching courses
  test("Get courses - exception handling", async () => {
    // Mock rejection for a network error
    (get as vi.Mock).mockRejectedValueOnce(new Error("Network Error"));

    const result = await getCourses(); // Call the getCourses function
    expect(result).toEqual([]); // Ensure an empty array is returned in case of an error
  });
});
```

Figure 24. Get Courses

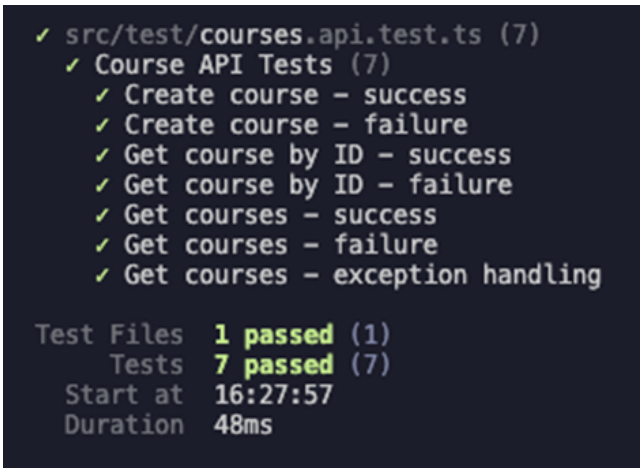


Figure 25. Course Test Case Result

2.2.7 Feedback

```
describe("Feedback API Tests", () => {
  // Test data
  const feedbackAnswers: FeedbackAnswer[] = [
    { feedbackId: 1, questionID: 1, answer: 5 },
    { feedbackId: 1, questionID: 2, answer: 4 },
  ];

  const studentID = 49; // Example student ID
  const feedbackIdResponse = { success: true, data: { id: 123 } }; //
  Mocked feedback ID response
  const feedbackQuestionResponse = [
    { id: 1, options: 3, message: "How was your experience?",
    createdAt: new Date(), updatedAt: new Date() }
  ]; // Mocked feedback question data

  let consoleErrorSpy: vi.SpiedFunction;

  // Setup and teardown for mocking console.error
  beforeEach(() => {
    vi.clearAllMocks(); // Clear mocks between tests
    consoleErrorSpy = vi.spyOn(console, "error").mockImplementation(()
    => {}); // Mock console.error
  });

  afterEach(() => {
    consoleErrorSpy.mockRestore(); // Restore original console.error
    after each test
  });
```

Figure 26. Feedback Test Data

```
// Test for successful creation of feedback answers
test("Create feedback - success", async () => {
  // Mock successful response for feedback submission
  (post as vi.Mock).mockResolvedValueOnce({ success: true, data:
  feedbackAnswers });

  await createFeedback(feedbackAnswers); // Call the function to
  create feedback

  // Check if the correct number of requests were made
  expect(post).toHaveBeenCalledTimes(feedbackAnswers.length);

  // Verify that the URL and request body are correct for each
  feedback answer
  feedbackAnswers.forEach((feedbackAnswer) => {
    expect(post).toHaveBeenCalledWith(expect.objectContaining({
      url: `/feedback-answer/feedback/${feedbackAnswer.feedbackId}/
      question/${feedbackAnswer.questionID}/answer`,
      body: { answer: feedbackAnswer.answer },
    }));
  });
});

// Test for feedback creation failure
test("Create feedback - failure", async () => {
  // Mock failure response
  (post as vi.Mock).mockResolvedValueOnce({ success: false });

  await createFeedback(feedbackAnswers); // Call the function to
  create feedback

  // Verify that error logging occurs when creation fails
  expect(consoleErrorSpy).toHaveBeenCalled();
});
```

Figure 27. Create Feedback

```
// Test for creating feedback ID
test("Create feedback ID - success", async () => {
  // Mock response for feedback ID creation
  (post as vi.Mock).mockResolvedValueOnce(feedbackIdResponse);

  const feedbackId = await createFeedbackid(studentID);
  expect(feedbackId).toBe(123); // Assert that the returned feedback
  ID is correct
});
```

Figure 28. Create Feedback ID

```
// Test for successfully fetching feedback questions
test("Get feedback questions - success", async () => {
  // Mock response for fetching feedback questions
  (get as vi.Mock).mockResolvedValueOnce({ success: true, data:
  feedbackQuestionResponse });

  const questions = await getFeedbackQuestions();
  expect(questions).toEqual(feedbackQuestionResponse); // Ensure the
  questions are returned correctly
});

// Test for failure when fetching feedback questions
test("Get feedback questions - failure", async () => {
  // Mock failure response for fetching feedback questions
  (get as vi.Mock).mockResolvedValueOnce({ success: false, data:
  [] });

  const questions = await getFeedbackQuestions();
  expect(questions).toEqual([]); // Ensure an empty array is returned
  on failure
});
```

Figure 29. Get Feedback Questions

```
  // Test for handling network errors when fetching feedback questions
  test("Get feedback questions - exception handling", async () => {
    // Mock rejection for a network error
    (get as vi.Mock).mockRejectedValueOnce(new Error("Network Error"));

    const questions = await getFeedbackQuestions();
    expect(questions).toEqual([]); // Ensure an empty array is returned
    in case of an error
  });
});
```

Figure 30. Exception Handling

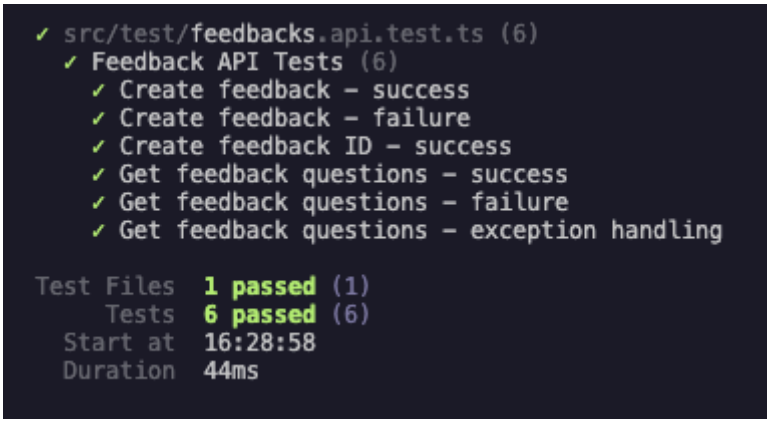


Figure 31. Feedback Test Case Result

2.2.8 Recommended Courses

```
describe("Recommendation Courses API Tests", () => {
  const studentId = 49; // Example student ID
  const recommendationID = 123; // Mock recommendation ID
  const testIdResponse: testId = { id: 1, student_id: studentId }; // Mock test ID response
  const recommendationTableResponse: recommendationTable[] = [
    { id: 1, recommendationId: recommendationID, testId: 1, studentId: studentId, createdAt: new Date() }
  ]; // Mock recommendation table response
  const recommendationCoursesResponse: Rcourses[] = [
    {
      id: 1,
      courseName: "Software Engineering",
      schoolName: "Tech University",
      score: 85,
      careerProspect: ["Developer", "Project Manager"],
      keySkills: ["Programming", "Problem Solving"],
      intake: 2024,
      courseDescription: "An in-depth course on software engineering principles.",
      facultyName: "Dr. John Doe"
    }
  ]; // Mock recommendation courses response

  const recommendedCourseResponse: Rcourses = {
    id: 1,
    courseName: "Data Science",
    schoolName: "Data University",
    score: 90,
    careerProspect: ["Data Analyst", "Data Scientist"],
    keySkills: ["Python", "Machine Learning"],
    intake: 2025,
    courseDescription: "Learn the fundamentals of data science.",
    facultyName: "Prof. Jane Doe"
  };

  beforeEach(() => {
    vi.clearAllMocks(); // Clear mocks before each test
  });

  // Test case for getting the latest test ID for a student
  test("Get latest test ID - success", async () => {
    (get as vi.Mock).mockResolvedValueOnce({ success: true, data: testIdResponse });

    const result = await getLatestTestId(studentId);
    expect(result).toBe(testIdResponse.id); // Ensure the result matches the mock test ID
    expect(get).toHaveBeenCalledWith(expect.objectContaining({
      url: `/test/student/${studentId}/latest`,
    }));
  });
```

Figure 32. Get Latest Test ID

```
// Test case for getting recommendation ID by test ID
test("Get recommendation ID by test ID - success", async () => {
  (get as vi.Mock).mockResolvedValueOnce({ success: true, data: recommendationTableResponse });

  const result = await getRecommendationIDBytestId(testIdResponse.id);
  expect(result).toEqual(recommendationTableResponse); // Ensure the result matches the mock recommendation table
  expect(get).toHaveBeenCalledWith(expect.objectContaining({
    url: `/recommendation/test/${testIdResponse.id}`,
  }));
});

// Test case for getting recommendation courses by recommendation ID
test("Get recommendation courses by recommendation ID - success", async () => {
  (get as vi.Mock).mockResolvedValueOnce({ success: true, data: recommendationCoursesResponse });

  const result = await getRecommendationCoursesByRecommendationID(recommendationID);
  expect(result).toEqual(recommendationCoursesResponse); // Ensure the result matches the mock courses data
  expect(get).toHaveBeenCalledWith(expect.objectContaining({
    url: `/recommendation/${recommendationID}/courses`,
  }));
});
```

Figure 33. Get Recommendation ID by Test ID and Get Recommendation Courses by Recommendation ID

```
// Test case for creating a recommended course
test("Create recommended course - success", async () => {
  (post as vi.Mock).mockResolvedValueOnce({ success: true, data: recommendedCourseResponse });

  const result = await createRecommendedCourse(recommendedCourseResponse);
  expect(result).toEqual(recommendedCourseResponse); // Ensure the result matches the mock course data
  expect(post).toHaveBeenCalledWith(expect.objectContaining({
    url: "/recommended",
    body: recommendedCourseResponse,
  }));
});

// Test case for getting a recommended course by student ID
test("Get recommended course - success", async () => {
  (get as vi.Mock).mockResolvedValueOnce({ success: true, data: [recommendedCourseResponse] });

  const result = await getRecommendedCourse(studentId);
  expect(result).toEqual([recommendedCourseResponse]); // Ensure the result matches the mock course data
  expect(get).toHaveBeenCalledWith(expect.objectContaining({
    url: `/recommendation/student/${studentId}`,
  }));
});
```

Figure 34. Create Recommended Course

```
  // Test case for error handling when fetching latest test ID
  test("Get latest test ID - error", async () => {
    (get as vi.Mock).mockRejectedValueOnce(new Error("Network Error"));

    const result = await getLatestTestId(studentId);
    expect(result).toBeNull(); // Ensure null is returned in case of error
  });

  // Test case for error handling when getting recommendation courses
  test("Get recommendation courses - error", async () => {
    // Mock rejection for a network error
    (get as vi.Mock).mockRejectedValueOnce(new Error("Network Error"));

    // Ensure that the error is caught and an empty array is returned
    const result = await getRecommendationCoursesByRecommendationID(recommendationID);
    expect(result).toEqual([]); // Ensure an empty array is returned in case of error
  });
});
```

Figure 35. Error handling for Get Latest Test ID and Get Recommendation Courses

```
✓ src/test/recommended.api.test.ts (7)
  ✓ Recommendation Courses API Tests (7)
    ✓ Get latest test ID - success
    ✓ Get recommendation ID by test ID - success
    ✓ Get recommendation courses by recommendation ID - success
    ✓ Create recommended course - success
    ✓ Get recommended course - success
    ✓ Get latest test ID - error
    ✓ Get recommendation courses - error

Test Files  1 passed (1)
     Tests  7 passed (7)
  Start at  16:29:36
  Duration  48ms
```

Figure 36. Recommendation Course Test Case Result