TESTPLAN
Project Cloud Computing 2

Group 1:

Brandon Palmer 616702
Tanzeel Rehman 648882
Mahedi Mridul 632901
Jesse van Evert 625868

2 Febuary 2023

Table of Contents

Context

The Clients are looking for an application that allows them to send SMS messages to an entire class or a student, in case of absence due to illness or otherwise.

Requirements

- Requirements, functional and non-functional
    - Functional Requirements:
        - Ability to send SMS messages to an entire class at once
        - Contact list or database of students belonging to a class
        - Function for composing and sending email messages using SendGrid
        - Ability to create, read, update, and delete class with students
    - Non-Functional Requirements
        - User-Friendly
        - Secure Storage of student personal information
        - Reliability and availability of the application
        - Application performance (e.g. speed of sending messages, response time)
        - Scalability to handle a large number of students

Risk’s

- Privacy risks: if the application is not secured correctly, the personal data of students and teachers, including their telephone numbers, are violated.

- Risks of miscommunication: if the application is not user-friendly or reliable, the teacher cannot send the messages effectively, which leads to confusion and miscommunication among the students.

- Compliance Risks: if the application does not comply with the regulations data protection, the teacher and the school may face legal consequences.

- Technical risks: if the application is not scalable, it can handle a large number students, and if it is not compatible with different mobile platforms, it is some students may not be able to receive the messages.

- Risk of not reaching all students: There may be instances where a student has changed his telephone number and the teacher is not aware of this the the message will not reach the student.

- Risk of no access to the application: If the teacher due to technical problems does not have access to the application, and he cannot send messages.

Results

Message-MicroService Test Plan

A test plan for the "http://localhost:8080/messages" endpoint:

1. Test for a valid message ID:
- Input: A valid message ID as a URL parameter
- Expected Output: HTTP response code 200 (OK) and a JSON response containing the message with the matching ID
2. Test for an invalid message ID:
- Input: An invalid message ID as a URL parameter
- Expected Output: HTTP response code 400 (Bad Request) and a JSON response with an error message indicating that the message was not found
3. Test for a missing message ID:
- Input: No message ID in the URL parameter
- Expected Output: HTTP response code 400 (Bad Request) and a JSON response with an error message indicating that the message ID is required
4. Test for a non-numeric message ID:
- Input: A non-numeric message ID as a URL parameter
- Expected Output: HTTP response code 400 (Bad Request) and a JSON response with an error message indicating that the message ID must be a number

A test plan for the " http://localhost:8080/messages/${this.messageID} " endpoint:
Test Objective: Verify that the endpoint returns a correct response when a valid message ID is passed in the URL.

- Input: messageID = 1
- Expected Output: A JSON object containing information about the message with the specified ID, such as message content and creation date.
- Test Steps:
    - Send a GET request to the endpoint with the messageID parameter set to 1.
    - Verify that the response has a 200 OK status code.
    - Verify that the response body contains the expected information for the message with ID 1.

5. Test Objective: Verify that the endpoint returns a 404 Not Found response when an invalid message ID is passed in the URL.

- Input: messageID = 100
- Expected Output: A 404 Not Found response.
- Test Steps:
    - Send a GET request to the endpoint with the messageID parameter set to 100.

- Verify that the response has a 404 Not Found status code.
- Verify that the response body contains a relevant error message.

Test Plan for Endpoint: http://localhost:8080/messages/lecturer/${this.lecturerEmail}
Objective: To validate the functionality of the endpoint that retrieves messages sent to a particular lecturer.
Test Scenario:

6. Verify that the endpoint is accessible using GET method.
7. Verify that the endpoint returns a 200 OK status code when a valid lecturer email is provided.
8. Verify that the endpoint returns a 400 Bad Request status code when an invalid lecturer email format is provided.
9. Verify that the endpoint returns a 404 Not Found status code when a non-existing lecturer email is provided.
10. Verify that the endpoint returns the messages sent to the specified lecturer in the correct format (i.e. JSON).
11. Verify that the endpoint only returns the messages sent to the specified lecturer and not messages sent to other lecturers.

Test Data:

- Valid lecturer email: lecturer1@university.com
- Invalid lecturer email format: lecturer1university.com
- Non-existing lecturer email: lecturer2@university.com
- Messages sent to lecturer1@university.com:
  - Message 1: Subject: "Class Assignment" Body: "Please submit the class assignment by Friday."
  - Message 2: Subject: "Mid-term Exam" Body: "The mid-term exam will be held next week."

Expected Results:

12. Endpoint is accessible using GET method.
13. Endpoint returns a 200 OK status code when a valid lecturer email is provided.
14. Endpoint returns a 400 Bad Request status code when an invalid lecturer email format is provided.

1. Verify endpoint URL: Ensure that the URL is correct and accessible.
2. Test HTTP method: Confirm that the endpoint only supports the HTTP GET method.

3. Check response status code: Verify that the endpoint returns a 200 OK status code, indicating that the request was successful.
4. Validate response content: Ensure that the response body contains valid data in JSON format, including the required fields for a classroom resource.
5. Check pagination: Test that the endpoint returns a specified number of results per page and allows for navigating through multiple pages of results.
6. Test sorting: Verify that the endpoint supports sorting the results by different fields, such as the classroom name.
7. Test filtering: Confirm that the endpoint supports filtering the results by certain criteria, such as by a specific teacher's name.
8. Test error handling: Ensure that the endpoint returns appropriate error messages for requests with missing or invalid parameters.
9. Test security: Verify that the endpoint is secured and only accessible to authorized users.
10. Performance testing: Evaluate the endpoint's performance under different conditions, such as high concurrent requests, to ensure that it meets the requirements for response time and reliability.

Student

1. Objective: To verify that the API endpoint **http://127.0.0.1:8000/api/ui/#/Student/studentservice.read_all** returns a list of all students as expected.
2. Prerequisites:
    - API endpoint is up and running
    - Valid API endpoint URL
    - API authentication credentials are available
    - API response structure is known
3. Test Cases:

1. Test for valid API endpoint URL
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.read_all
    - Expected Output: HTTP response code 200 (OK)
2. Test for proper authentication
    - Input: Use API authentication credentials
    - Expected Output: HTTP response code 200 (OK)
3. Test for expected response structure
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.read_all
    - Expected Output: JSON array of student objects with predefined keys
4. Test for expected number of students returned
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.read_all
    - Expected Output: Correct number of students as per the database

4. Test Data:
    - API endpoint URL: http://127.0.0.1:8000/api/ui/#/Student/studentservice.read_all
    - API authentication credentials
    - Expected response structure
    - Expected number of students returned
5. Test Environment:
    - API endpoint URL: http://127.0.0.1:8000/api/ui/#/Student/studentservice.read_all
    - Testing tool (e.g. Postman, cURL, etc.)
    - Test data

1. Objective: To verify that the API endpoint **http://127.0.0.1:8000/api/ui/#/Student/studentservice.create** creates a new student record as expected.

2. Prerequisites:
    - API endpoint is up and running
    - Valid API endpoint URL
    - API authentication credentials are available
    - API request payload structure is known
3. Test Cases:

1. Test for valid API endpoint URL
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.create
    - Expected Output: HTTP response code 200 (OK)
2. Test for proper authentication
    - Input: Use API authentication credentials
    - Expected Output: HTTP response code 200 (OK)
3. Test for proper request payload structure
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.create with a proper payload
    - Expected Output: HTTP response code 201 (Created)
4. Test for proper response structure
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.create with a proper payload
    - Expected Output: JSON object of the created student with predefined keys
5. Test for proper database updates
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.create with a proper payload
    - Expected Output: The student record is created in the database as expected

4. Test Data:
    - API endpoint URL: http://127.0.0.1:8000/api/ui/#/Student/studentservice.create
    - API authentication credentials
    - Proper request payload
    - Expected response structure
5. Test Environment:
    - API endpoint URL: http://127.0.0.1:8000/api/ui/#/Student/studentservice.create
    - Testing tool (e.g. Postman, cURL, etc.)
    - Test data

1. Objective: To verify that the API endpoint **http://127.0.0.1:8000/api/ui/#/Student/studentservice.update** updates the **deleted_at** field of a student record to the current date and time as expected.
2. Prerequisites:
    - API endpoint is up and running
    - Valid API endpoint URL
    - API authentication credentials are available
    - API request payload structure is known
3. Test Cases:

1. Test for valid API endpoint URL
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.update
    - Expected Output: HTTP response code 200 (OK)
2. Test for proper authentication
    - Input: Use API authentication credentials
    - Expected Output: HTTP response code 200 (OK)
3. Test for proper request payload structure
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.update with a proper payload
    - Expected Output: HTTP response code 200 (OK)
4. Test for proper response structure
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.update with a proper payload
    - Expected Output: JSON object of the updated student with predefined keys
5. Test for proper database updates
    - Input: http://127.0.0.1:8000/api/ui/#/Student/studentservice.update with a proper payload
    - Expected Output: The **deleted_at** field of the student record is updated to the current date and time as expected in the database

4. Test Data:
    - API endpoint URL: http://127.0.0.1:8000/api/ui/#/Student/studentservice.update
    - API authentication credentials
    - Proper request payload
    - Expected response structure
5. Test Environment:
    - API endpoint URL: http://127.0.0.1:8000/api/ui/#/Student/studentservice.update
    - Testing tool (e.g. Postman, cURL, etc.)

- Test data

Classes

Get All
Test Plan for Endpoint: http://localhost:8000/api/classrooms

Objective:
To validate the functionality of the endpoint that retrieves information about classrooms.
Test Scenario:
Verify Accessibility with GET Method: Ensure that the endpoint is accessible using the GET method.
Verify Successful Response: Confirm that the endpoint returns a 200 OK status code when the request is successful.
Verify JSON Format: Ensure that the endpoint returns the information about classrooms in the correct format (i.e. JSON).
Verify Correct Classroom Information: Verify that the endpoint returns the correct information about each classroom, such as classroom name, capacity, and location.
Verify Error Message for Unauthorized Requests: Ensure that the endpoint returns an error message (e.g. 401 Unauthorized) when a user who is not authorized to access the endpoint tries to make a request.

Test Data:
Classroom 1: Name: "IT1a"
Classroom 2: Name: "IT2a"

Expected Results:

Endpoint is accessible using GET method.
Endpoint returns a 200 OK status code when the request is successful.
Endpoint returns the information about classrooms in the correct format (i.e. JSON).
Endpoint returns the correct information about each classroom, such as classroom name.

PUT (update)

Test Plan for Endpoint: http://localhost:8000/api/classroom/${this.classroomname}

Objective:
To validate the functionality of the endpoint that updates information about a specific classroom.

Test Scenario:
Verify Accessibility with GET Method: Ensure that the endpoint is accessible using the GET method.

Verify Successful Response: Confirm that the endpoint returns a 200 OK status code when the request is successful.
Verify JSON Format: Ensure that the endpoint returns the information about the specific classroom in the correct format (i.e. JSON).
Verify Correct Classroom Information: Verify that the endpoint returns the correct information about the specified classroom, such as classroom name.
Verify Error Message for Non-Existent Classroom: Ensure that the endpoint returns an error message (e.g. 404 Not Found) when a request is made for a non-existent classroom.

Test Data:
Classroom 1: Name: "IT1a"
Classroom 2: Name: "IT2a"

Expected Results:
Endpoint is accessible using GET method.
Endpoint returns a 200 OK status code when the request is successful.
Endpoint returns the information about the specific classroom in the correct format (i.e. JSON).
Endpoint returns the correct information about the specified classroom, such as classroom name, capacity, and location.
Endpoint returns an error message (e.g. 404 Not Found) when a request is made for a non-existent classroom

DELETE

Test Plan for Endpoint: http://localhost:8000/api/classroom/${this.classroomname}

Objective:

To validate the functionality of the endpoint that deletes information about a specific classroom.

Test Scenario:

Verify Accessibility with DELETE Method: Ensure that the endpoint is accessible using the DELETE method.
Verify Successful Deletion: Confirm that the endpoint returns a 200 OK status code when the classroom is successfully deleted.
Verify Error Message for Non-Existent Classroom: Ensure that the endpoint returns an error message (e.g. 404 Not Found) when a request is made to delete a non-existent classroom.
Verify Classroom Deletion: Verify that the classroom has been deleted from the database.

Test Data:
Classroom 1: Name: "IT1a"
Classroom 2: Name: "IT2a"

Expected Results:

Endpoint is accessible using DELETE method.
Endpoint returns a 200 OK status code when the classroom is successfully deleted.
Endpoint returns an error message (e.g. 404 Not Found) when a request is made to delete a non-existent classroom.
Classroom has been deleted from the database.

POST
Test Plan for Endpoint: http://localhost:8000/api/classroom/${this.classroomname}

Objective:
To validate the functionality of the endpoint that creates information about a specific classroom.

Test Scenario:

Verify Accessibility with POST Method: Ensure that the endpoint is accessible using the POST method.
Verify Successful Creation: Confirm that the endpoint returns a 201 Created status code when the classroom is successfully created.

Verify Error Message for Duplicate Classroom: Ensure that the endpoint returns an error message (e.g. 409 Conflict) when a request is made to create a classroom with a name that already exists in the database.
Verify Classroom Creation: Verify that the classroom has been created in the database and the information provided in the request (e.g. classroom name, capacity, and location) is correct.

Test Data:
Classroom 1: Name: "IT1a"
Classroom 2: Name: "IT2a"

Expected Results:

Endpoint is accessible using POST method.
Endpoint returns a 201 Created status code when the classroom is successfully created.
Endpoint returns an error message (e.g. 409 Conflict) when a request is made to create a classroom with a name that already exists in the database.
Classroom has been created in the database with the correct information.