**Project Overview**

* **Objective**: Automate testing of the https://jsonplaceholder.typicode.com/ API using RestAssured for REST API calls, Cucumber for BDD-style test cases, and Jenkins for continuous integration.
* **Technologies**:
  + **RestAssured**: For making REST API calls and validating responses.
  + **Cucumber**: For writing tests in a Gherkin language (BDD format).
  + **JUnit**: For running Cucumber tests.
  + **Maven**: For project and dependency management.
  + **Jenkins**: For CI/CD pipeline setup

**Example Test Scenarios**

**1. GET /posts**

* **Scenario**: Retrieve all posts.
* **Expected Results**:
  + Status code should be 200.
  + The response should contain a list of 100 posts.
  + Each post should have a userId, id, title, and body.

**2. GET /posts/{id}**

* **Scenario**: Retrieve a specific post by ID.
* **Expected Results**:
  + Status code should be 200.
  + The post with the specified ID should be returned.
  + The post should contain the correct userId, title, and body.

**3. GET /posts/{id} with invalid ID**

* **Scenario**: Retrieve a post with an invalid ID (e.g., 9999).
* **Expected Results**:
  + Status code should be 404.
  + The response body should be empty.

**4. POST /posts**

* **Scenario**: Create a new post.
* **Expected Results**:
  + Status code should be 201.
  + The response should contain the newly created post with the correct userId, title, and body.
  + The id of the new post should be greater than 100.

**5. POST /posts with missing fields**

* **Scenario**: Attempt to create a post with missing required fields (e.g., missing title).
* **Expected Results**:
  + Status code should be 400 or 500 depending on the server's validation logic.
  + The response should contain an error message indicating the missing field.

**6. PUT /posts/{id}**

* **Scenario**: Update an existing post by ID.
* **Expected Results**:
  + Status code should be 200.
  + The post should be updated with the new title and body.
  + Verify that the id and userId remain unchanged.

**7. PUT /posts/{id} with invalid ID**

* **Scenario**: Attempt to update a post with an invalid ID.
* **Expected Results**:
  + Status code should be 404.
  + The response should be empty or contain an error message.

**8. DELETE /posts/{id}**

* **Scenario**: Delete a specific post by ID.
* **Expected Results**:
  + Status code should be 200 or 204.
  + Verify that the post is no longer accessible by sending a GET request to the same id and expecting a 404 status code.

**9. DELETE /posts/{id} with invalid ID**

* **Scenario**: Attempt to delete a post with an invalid ID.
* **Expected Results**:
  + Status code should be 404.
  + The response should be empty or contain an error message.

**10. GET /comments?postId={postId}**

* **Scenario**: Retrieve comments for a specific post.
* **Expected Results**:
  + Status code should be 200.
  + The response should contain a list of comments associated with the specified postId.
  + Each comment should have a postId, id, name, email, and body.

**11. POST /comments**

* **Scenario**: Add a new comment to a post.
* **Expected Results**:
  + Status code should be 201.
  + The response should contain the new comment with the correct postId, name, email, and body.

**12. GET /users**

* **Scenario**: Retrieve all users.
* **Expected Results**:
  + Status code should be 200.
  + The response should contain a list of 10 users.
  + Each user should have an id, name, username, email, address, phone, website, and company.

**13. GET /users/{id}**

* **Scenario**: Retrieve a specific user by ID.
* **Expected Results**:
  + Status code should be 200.
  + The user with the specified id should be returned.
  + The user's information should include name, username, email, and other details.

**14. GET /albums**

* **Scenario**: Retrieve all albums.
* **Expected Results**:
  + Status code should be 200.
  + The response should contain a list of albums.
  + Each album should have an id, title, and userId.

**15. GET /albums/{id}**

* **Scenario**: Retrieve a specific album by ID.
* **Expected Results**:
  + Status code should be 200.
  + The album with the specified id should be returned.
  + The album should contain the correct title and userId.

**16. GET /photos**

* **Scenario**: Retrieve all photos.
* **Expected Results**:
  + Status code should be 200.
  + The response should contain a list of photos.
  + Each photo should have an albumId, id, title, url, and thumbnailUrl.

**17. GET /photos?albumId={albumId}**

* **Scenario**: Retrieve photos from a specific album.
* **Expected Results**:
  + Status code should be 200.
  + The response should contain a list of photos associated with the specified albumId.
  + Each photo should belong to the specified album.

**18. GET /todos**

* **Scenario**: Retrieve all todos.
* **Expected Results**:
  + Status code should be 200.
  + The response should contain a list of todos.
  + Each todo should have an id, title, userId, and completed status.

**19. PATCH /posts/{id}**

* **Scenario**: Partially update an existing post by ID (e.g., update only the title).
* **Expected Results**:
  + Status code should be 200.
  + The response should reflect the partial update with the new title while other fields remain unchanged.

**20. Comprehensive API Health Check**

* **Scenario**: Perform a comprehensive health check of the API by sending requests to all endpoints (/posts, /comments, /albums, /photos, /todos, /users).
* **Expected Results**:
  + Status code should be 200 for all requests.
  + Each endpoint should return a valid and correctly structured response.
  + The test should validate that all expected resources exist and are accessible.

**Final Steps**

* **Review Test Reports**: After each Jenkins build, review the generated reports for test results.
* **Improve Test Coverage**: Add more scenarios to cover other API endpoints.
* **Maintain the Project**: Keep the project up-to-date with changes in the API or test requirements.