Test Plan

Scenario 1: Plivo Send SMS Automation

Objective: Automate the sending of SMS messages to specific customers based on user input from a CSV file using Plivo Messaging APIs.

Preconditions:

1. A Plivo account is created, and authentication details (auth ID and token) are obtained.
2. Plivo Messaging API documentation is reviewed.

Test Cases:

1. **Test Case 1: Create Customer Message CSV File**
   - **Input**: None
   - **Action**: Create `customer_message.csv` with headers `ID`, `SourceNumber`, `DestinationNumber`, and `Message`.
   - **Expected Result**: File is created with the specified headers.
2. **Test Case 2: Add Data to CSV File**
   - **Input**: Customer data (ID, SourceNumber, DestinationNumber, Message)
   - **Action**: Add four lines of customer data to `customer_message.csv`.
   - **Expected Result**: Data is correctly added to the CSV file.
3. **Test Case 3: Read CSV and Send Single SMS**
   - **Input**: Customer ID = 1
   - **Action**: Read `customer_message.csv` and send an SMS to customer ID 1.
   - **Expected Result**: SMS is sent successfully, and message details are logged in `result.txt`.
4. **Test Case 4: Read CSV and Send Multiple SMS**
   - **Input**: Customer IDs = 1,4
   - **Action**: Read `customer_message.csv` and send SMS to customer IDs 1 and 4.
   - **Expected Result**: SMS messages are sent successfully, and details are logged in `result.txt`.
5. **Test Case 5: Verify Message Queued Successfully**
   - **Input**: None
   - **Action**: Use GET Message API to verify the message status.
   - **Expected Result**: Message status is "queued", and details are documented in `result.txt`.

Automation Steps:

1. Create `customer_message.csv` file with the specified headers.
2. Add four lines of customer data to the CSV file.
3. Accept customer IDs as input.
4. Read the contents of the CSV file based on the input IDs.
5. Send SMS using Plivo Send Message API.
6. Verify message status using GET Message API and document details in `result.txt`.

Scenario 2: Open Weather Automation

Objective: Get weather statistics for a list of cities and identify the top N cities with the lowest temperature and highest humidity using Openweathermap APIs.

Preconditions:

1. An Openweathermap account is created, and an API Key is obtained.
2. Openweathermap API documentation is reviewed.

Test Cases:

1. **Test Case 1: Create City CSV File**
   - **Input**: None
   - **Action**: Create `city.csv` with a list of cities.
   - **Expected Result**: File is created with the specified cities.
2. **Test Case 2: Get Geocoding Data for Cities**
   - **Input**: City names from `city.csv`
   - **Action**: Retrieve longitude and latitude for each city.
   - **Expected Result**: Longitude and latitude data are correctly retrieved.
3. **Test Case 3: Get Weather Data for Cities**
   - **Input**: Longitude and latitude of cities
   - **Action**: Retrieve weather data using One Call API.
   - **Expected Result**: Weather data is correctly retrieved and saved in `city_stats.csv`.
4. **Test Case 4: Identify Top N Coldest Cities**
   - **Input**: N (default = 3)
   - **Action**: Identify top N cities with the lowest temperature.
   - **Expected Result**: Correct cities are identified and documented.
5. **Test Case 5: Identify Top N Cities with Highest Humidity**
   - **Input**: N (default = 3)
   - **Action**: Identify top N cities with the highest humidity.
   - **Expected Result**: Correct cities are identified and documented.

Automation Steps:

1. Create `city.csv` file with a list of cities.
2. Retrieve longitude and latitude for each city using the Geocoding API.
3. Retrieve weather data for each city using the One Call API.
4. Save weather statistics in `city_stats.csv`.
5. Identify top N cities with the lowest temperature and highest humidity.
6. Document the results in a CSV file and a report.

Scenario 3: Slack Automation

Objective: Automate various use cases for Slack channels using Slack APIs.

Preconditions:

1. A Slack API account is created, and an API Token is obtained.
2. Slack API documentation is reviewed.

Test Cases:

1. **Test Case 1: Create New Channel**
   - **Input**: Channel name
   - **Action**: Create a new Slack channel.
   - **Expected Result**: Channel is created successfully.
2. **Test Case 2: Join Newly Created Channel**
   - **Input**: Channel name
   - **Action**: Join the newly created Slack channel.
   - **Expected Result**: Successfully joined the channel.
3. **Test Case 3: Rename Channel**
   - **Input**: Old and new channel names
   - **Action**: Rename the Slack channel.
   - **Expected Result**: Channel name is updated successfully.
4. **Test Case 4: List All Channels**
   - **Input**: None
   - **Action**: List all Slack channels.
   - **Expected Result**: All channels are listed, and the renamed channel is validated.
5. **Test Case 5: Archive Channel**
   - **Input**: Channel name
   - **Action**: Archive the Slack channel.
   - **Expected Result**: Channel is archived successfully.
6. **Test Case 6: Validate Archived Channel**
   - **Input**: Channel name
   - **Action**: Validate the archived channel status.
   - **Expected Result**: Channel status is archived.

Automation Steps:

1. Create a new Slack channel.
2. Join the newly created channel.
3. Rename the channel.
4. List all channels and validate the renamed channel.
5. Archive the channel.
6. Validate the archived channel status.