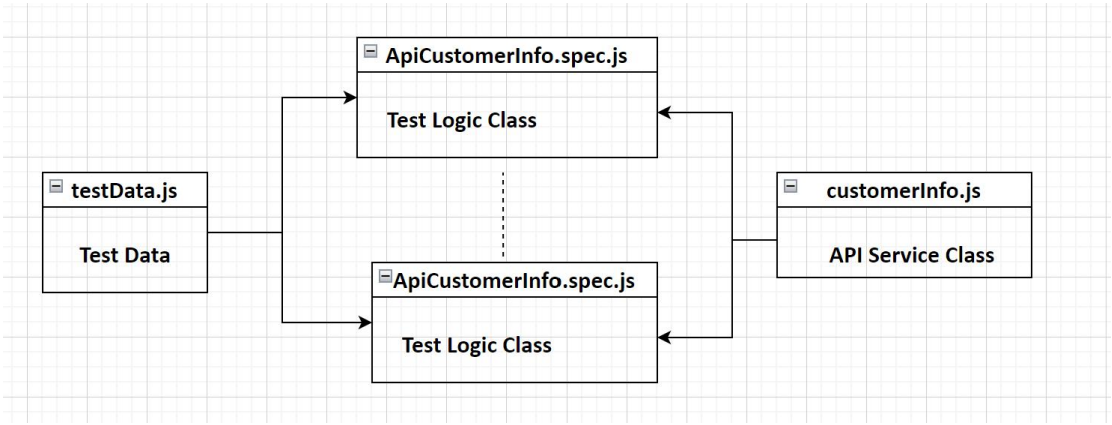
Explanation:

1. **Tool selection:** Since I already had Playwright set up on my personal laptop, using Playwright with JavaScript for API testing was a temporary solution. While it may not be the ideal tool for long-term API testing, it was the most convenient option for completing this demo and demonstrating my approach to API automation.

2. **Design Pattern:**



· **API Service Class(customerInfo.js)**: Encapsulates all the API calls (GET, POST, DELETE) and handles requests and responses.

· **Test Logic Class(ApiCustomerInfo.spec.js)**:

- Create multiple users one by one ;
- Get users' information and verify(e.g.,response code, email and phone number formats);
- Delete all created users at the end

· **Test Data(testData.js):** ·

- Modular Data: The test data (userData) is in the separate file and can be imported wherever needed.
- Data Management: It allows to manage multiple sets of test data if needed, just by adding more data sets to testData.js.

3. **Benefit of Design Pattern:**

- Separation of Concerns: The API service class is responsible for the actual API calls while the test class is responsible for writing assertions based on the API responses, and the separate test data file is used to store and manage test data allowing that reuse and modification of test data without altering the core test logic.

- Reusability: The API service class can be used across multiple test cases, allowing to reuse API calls without duplicating code.

- **Scalability**: If the number of API endpoints and tests grows, it's easy to add more methods to the service class and corresponding tests without increasing complexity.

- **Maintainability**: If the API endpoint changes, we only need to update the service class. The test class remains unaffected.

4. Further Enhancements:

Environment Variables: Use environment variables to store the base URL and other configurations.

Error Handling: Add better error handling for API failures or unexpected responses, such as non-JSON responses

5. ApiCustomerInfo.spec.js

· **Creating Multiple Users:** Loop through the users array (from testData.js) and create each user by calling user.createUser().

· **Storing User IDs:** Store the users' ID after being created in the createdUsers array, so that can delete them later.

· **Verifying Each User:** After creating the users, we make a GET request to fetch the list of users and verify email and phone_number fields as required.

· **Deleting Multiple Users:** After verifying, loop through the createdUsers array and delete each user based on users' ID using user.deleteUser(userId).

6. customerInfo.js

· **Constructor**: Initializes the class with request, userData, and userID, and sets the base API URL. It also defines regular expressions to validate email and phone number formats.

· **createUser**(): Makes a POST request to create a user with the data provided in userData and logs the status code. The status check is commented out to avoid affecting other API tests.

· **getUsers**(): Makes a GET request to retrieve the list of users from the API.

· **deleteUser**(userID): Deletes a user based on the given userID by making a DELETE request and logs the status code. The status check is similarly commented out to avoid affecting other API tests.

7. testData.js

Strore and manage test data. Here defined normal and abnormal test data, such as user with normal email, phone number and user with abnormal email address or abnormal phone number.

Prerequisites: To run this script, ensure have Node.js and Playwright installed:

1. download and install node.js
   https://nodejs.cn/download/

2. Install Playwright with command:

npm install playwright

3. Run the test:

npx playwright test ApiCustomerInfo.spec.js --project chromium --headed