**1. What Types of Tests Would You Create to Cover This App?**

To ensure the reliability and functionality of the insurance premium calculator app and API integration, the following types of tests would be implemented:

1. **End-to-End (E2E) Tests**:
   * These tests simulate real user interactions with the application, covering the entire workflow from form input to premium calculation. They validate the complete user journey, ensuring that each step in the process works seamlessly together.
2. **API Tests**:
   * These tests verify the responses from external APIs (e.g., PokeAPI) to ensure that the correct data is retrieved and handled by the application. This includes validating both successful responses (200 status) and error scenarios (e.g., 404 status).
3. **Form Validation Tests**:
   * These tests focus on the validation logic for form inputs, ensuring that invalid data (e.g., incorrect birthdate format) triggers appropriate error messages, and valid data allows users to proceed.
4. **Mocking and Simulation Tests**:
   * These tests use mock responses to simulate various scenarios (e.g., network failures, server errors, and invalid API responses) to validate how the app handles unexpected situations.

**2. Describe Briefly What Test Cases Would You Write and Which Ones Would You Cover with Automated Tests? Name a Couple of Edge Cases You’ve Noticed.**

**Test Cases:**

1. **Form Input and Validation**:
   * **Test Case 1**: Validate the birthdate input fields by entering an invalid date (e.g., 32.13.2000). Ensure that the appropriate error message (Bitte gib ein korrektes Datum ein.) is displayed.
   * **Test Case 2**: Enter a valid birthdate (e.g., 01.01.1990) and verify that the user can proceed to the next step without errors.
2. **User Navigation and Workflow**:
   * **Test Case 3**: Simulate a user selecting "Angestellt" as the employment type, entering an income of 70,000, selecting "Vollversicherung," and proceeding through the steps to enter a birthdate. Verify that each step transitions correctly and that the correct follow-up actions (e.g., consultation, offer email, signup) are available.
3. **API Response Validation**:
   * **Test Case 4**: Send a GET request to https://pokeapi.co/api/v2/pokemon/pikachu and verify that the response status is 200, the name is pikachu, and the abilities include lightning-rod.
   * **Test Case 5**: Mock a GET request for charmander to return a 404 status and verify that the application correctly handles this error scenario.

**Automated Tests:**

1. **Automated Form Validation Tests**:
   * Automate the birthdate validation test cases using fixtures to store valid and invalid dates. This ensures consistency in test data and facilitates easier updates.
2. **Automated Navigation Flow**:
   * Automate the end-to-end user journey from employment selection to birthdate entry, validating that each step is correctly implemented and the user is guided through the process without issues.
3. **Automated API Tests**:
   * Automate API calls to validate responses for pikachu and charmander, using mocking to simulate different response statuses (e.g., 200 for successful retrieval, 404 for not found).

**Edge Cases:**

1. **Invalid Birthdate Entries**:
   * Testing with dates like 32.13.2000 or edge cases like a leap year date 29.02.2001 (which is not a leap year) to ensure the application handles invalid dates correctly.
2. **Mocked API Responses**:
   * Simulating different HTTP status codes (e.g., 404, 500) for the PokeAPI to ensure the application handles these gracefully and provides meaningful feedback to the use.