Technical Interview Case: API Testing and AI Research (Focus on Node.js)

This case consists of two parts: a Node.js API testing challenge with hidden vulnerabilities and a research task focused on Natural Language Processing (NLP) for disease detection from chat history. The aim is to assess the candidate's skills in both API testing and AI research.

Part 1: Node.js API Testing and Bug Fixing

Scenario:

You are building a backend for a healthcare app that logs user chat histories. The provided code contains vulnerabilities and potential issues that need to be identified, fixed, and tested.

Task:

The candidate will:

- Review the provided API code.
- Identify and address potential vulnerabilities and bugs.
- Write automated tests for the API.

Node.js Code with Vulnerabilities:

```
javascript
Copy code
// Import required modules
const express = require('express');
const app = express();

// Middleware to parse incoming JSON requests
app.use(express.json());

// Dummy data to simulate a database
let chatLogs = [];

// POST /log-chat endpoint (vulnerable version)
app.post('/log-chat', (req, res) => {
    const { user_id, chat_history } = req.body;


    if (!user_id || !chat_history) {
        return res.json({
            status: 'error',
            message: 'Invalid input. user_id and chat_history are
required.',
        });
    }
    const logEntry = {
        user_id,
        chat_history,
        timestamp: new Date().toISOString(), // Timestamp format might be
incorrect for certain clients
    };


    chatLogs.push(logEntry);


    res.json({
        status: 'success',
        message: 'Chat logged successfully.',
        logEntry, // Vulnerability: returning internal data directly
    });
});

// Start the server on port 3000
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
});
```

Candidate Instructions:

1. **Identify Vulnerabilities:**
   - Review the `POST /log-chat` API code for vulnerabilities.
   - Identify security and validation issues in the API code.
2. **Fix the Code:**
   - Implement fixes for the issues you identified.
   - Ensure proper validation and sanitization of inputs (`user_id` and `chat_history`).
   - Return appropriate HTTP status codes for errors (e.g., `400 Bad Request` for invalid inputs).
   - Avoid returning sensitive data in the API response.
3. **Write Automated Tests:**
   - Write tests using **Jest** or **Mocha** and **Supertest** to cover the following cases:
     - Valid request (should return `200 OK` with a success message).
     - Invalid requests:
       - Missing `user_id`.
       - Empty `chat_history`.
       - Invalid data types for `chat_history` (not an array).
   - Ensure the tests cover all edge cases.
4. **Dummy Data for Testing:**
   - **Valid Request:**

```
json
Copy code
{
  "user_id": "user123",
  "chat_history": [
    "Hello, I am feeling sick.",
    "I have been coughing for three days.",
    "Do you have any advice for me?"
  ]
}
```

- **Invalid Request (Missing user_id):**

  ```
  json
  Copy code
  {
    "chat_history": [
      "I have a headache and a fever.",
      "What should I do?"
    ]
  }
  ```

5. **Deliverables:**
   - Corrected API code.
   - Automated test scripts.
   - A brief explanation of the vulnerabilities and how you addressed them.
   - Test results (e.g., screenshots or a test report).

Part 2: NLP Research (Disease Detection Based on Chat History)

Scenario:

Your healthcare app is designed to detect diseases based on users' chat histories using NLP. The candidate will research and propose an NLP approach for this task.

Tasks:

1. **Research NLP Techniques:**
   - Investigate state-of-the-art NLP models (e.g., **BERT**, **GPT**, **BioBERT**, **ClinicalBERT**) that can help detect diseases from user chat histories.
   - Recommend a specific NLP method based on your research, considering the healthcare context.
2. **Data Requirements:**
   - Identify the type of data needed to train the model (e.g., labeled datasets with disease annotations, chat transcripts).
   - Propose a preprocessing strategy for the chat data, including handling medical terminology, tokenization, and other language processing techniques.
3. **Model Evaluation:**
   - Suggest suitable metrics for evaluating the disease detection model (e.g., **precision**, **recall**, **F1-score**, **ROC-AUC**).
   - Discuss how to evaluate the model in production, considering real-world healthcare applications.
4. **Bonus (optional):**

    - Implement a basic prototype of your proposed NLP model using a framework like **spaCy**, **Hugging Face Transformers, Tensorflow or PyTorch**.

Deliverables:

- A concise report (1-2 pages max) summarizing your research findings.
- If applicable, provide a Jupyter Notebook or Python script with a basic prototype of the NLP model.