Week- 3

Backend Engineering Launchpad

Pawan Panjwani

Airtribe

UNIT TESTING AND DEBUGGING
Node JS

Airtribe

Agenda

- Why do we need unit testing ?
- Frameworks for unit testing in node js
- Writing effective unit testing - Understanding mock, stub, code coverage
- Continuous Integration - Integrating unit testing into continuous integration
- Unit testing exercise for course rating application
- Question and answers

What are unit tests and why test ?

- Unit tests help catch errors early in the development process
- Helps improving code quality
- Allow safe refactoring and modularization of code
- Encourages collaboration and allows parallel development
- Tests serve as good documentation
- **Tests help reduce fear of things breaking on prod :)**

Difference between unit and integration testing

Unit Tests

- Isolate each part of the program
- Show that the individual parts are working correctly

Integration tests

- Test the interoperability of multiple subsystems
- Tests that “the nuts fit the bolts”

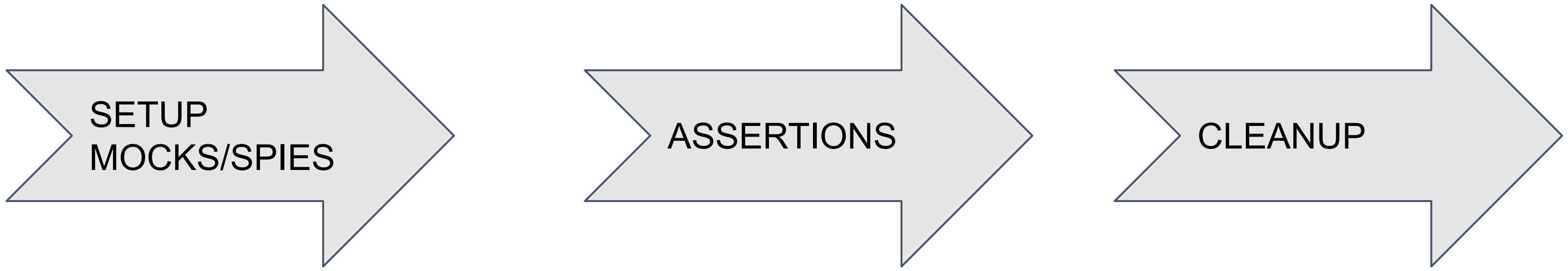
Airtribe

Frameworks for unit testing – Node JS

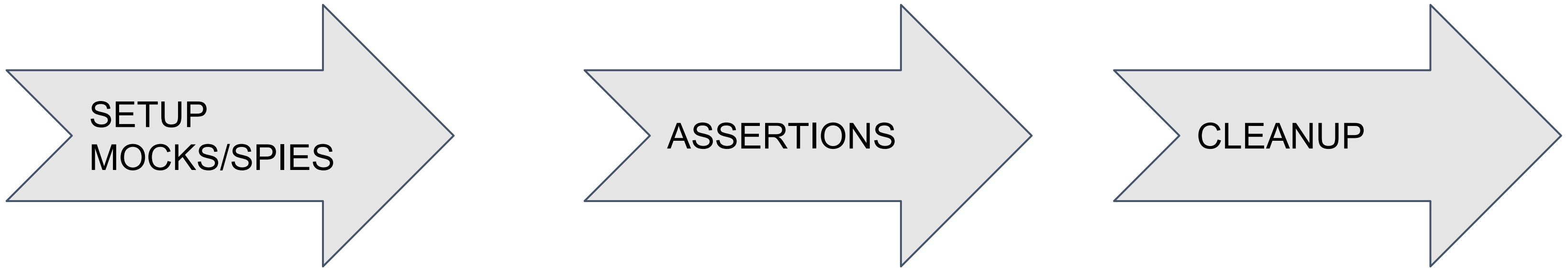
- Test Runner - Mocha.js
- Assertion Framework - Chai.js
- Stubbing/Mocking tool - Sinon.js
- All in one comprehensive framework - Jest
- Jest includes its own built in assertion library and mocking functionality
- Some other available frameworks - Jasmine, Tape, Ava

Airtribe

Anatomy of unit testing



Anatomy of unit testing – Mochafied



before()
beforeEach()

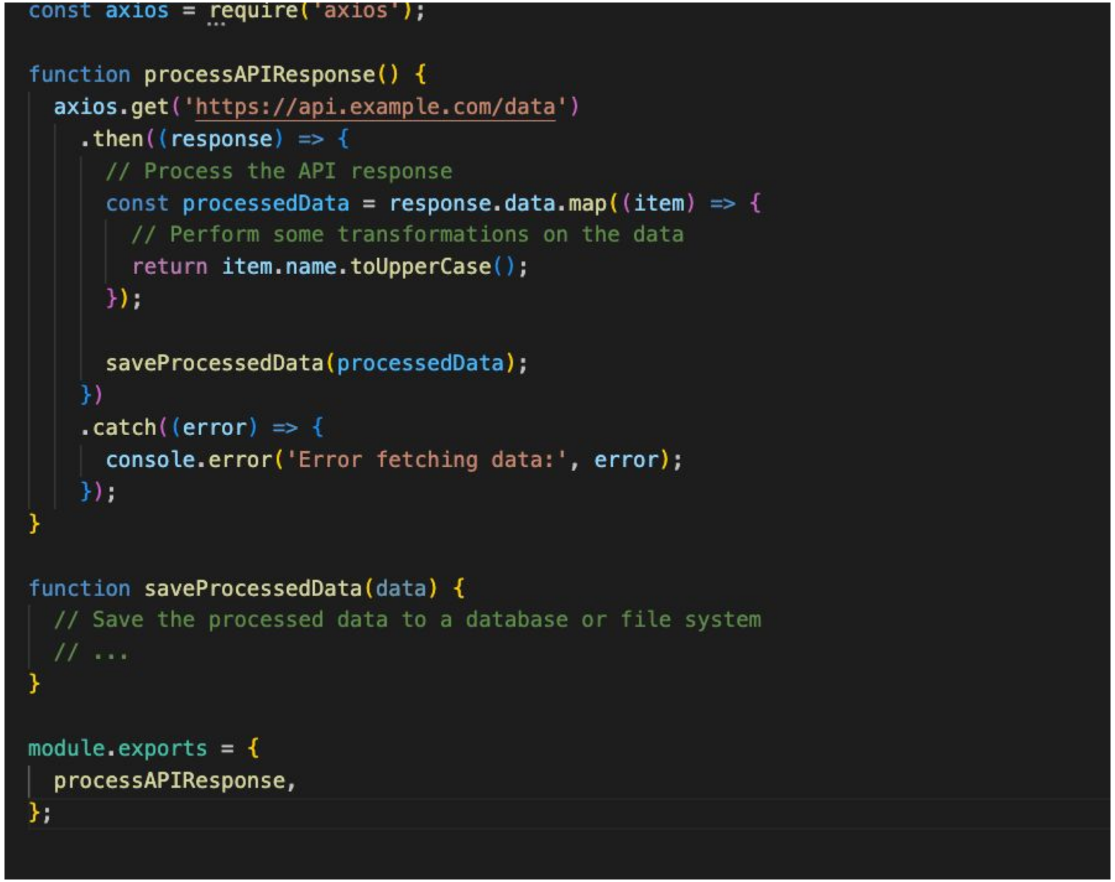
It()

after()
afterEach()

Problem statement

Airtribe

Identify problems with unit testing this piece of code



```javascript
const axios = require('axios');

function processAPIResponse() {
  axios.get('https://api.example.com/data')
    .then((response) => {
      // Process the API response
      const processedData = response.data.map((item) => {
        // Perform some transformations on the data
        return item.name.toUpperCase();
      });

      saveProcessedData(processedData);
    })
    .catch((error) => {
      console.error('Error fetching data:', error);
    });
}

function saveProcessedData(data) {
  // Save the processed data to a database or file system
  // ...
}

module.exports = {
  processAPIResponse,
};
```

Best Practices
Unit Testing

Airtribe

Isolate Dependencies

- Focus on testing a specific unit of code in isolation.
- Mock or stub external dependencies like databases, APIs, or modules.
- Tools: Sinon.js, Jest mocking capabilities.

Airtribe

Independent and Isolated Tests

- Each test should be independent and not rely on others.
- Isolating tests prevents failures from cascading.
- Test runners: Mocha, Jest.

Airtribe

Descriptive Test Names

- Use clear and descriptive names for test cases.
- Reflect the behavior or functionality being tested.
- Enhances readability and understanding of test failures.

Airtribe

Arrange Act Assert Patterns

- Structure tests into three sections: Arrange, Act, Assert.
- Arrange: Set up preconditions.
- Act: Execute the code being tested.
- Assert: Verify expected behavior and results.

Positive And Negative Scenarios

- Test both expected positive scenarios and edge cases.
- Identify and handle error conditions or unexpected behavior.

Use Assertions

- Utilize assertion libraries like Chai or Jest's built-in assertions.
- Verify return values, exceptions, and side effects.

Test Coverage

- Employ code coverage tools like Istanbul or Jest's coverage reporting.
- Measure the effectiveness of your tests.
- Aim for high test coverage.

Readable and Maintainable Tests

- Write clean, readable, and maintainable test code.
- Follow coding conventions and use descriptive names.
- Apply the DRY principle.

Automation

- Incorporate unit tests into an automated testing framework.
- Use continuous integration (CI) systems like Jenkins, Travis CI, or GitHub Actions.
- Catch issues early and ensure consistent execution.

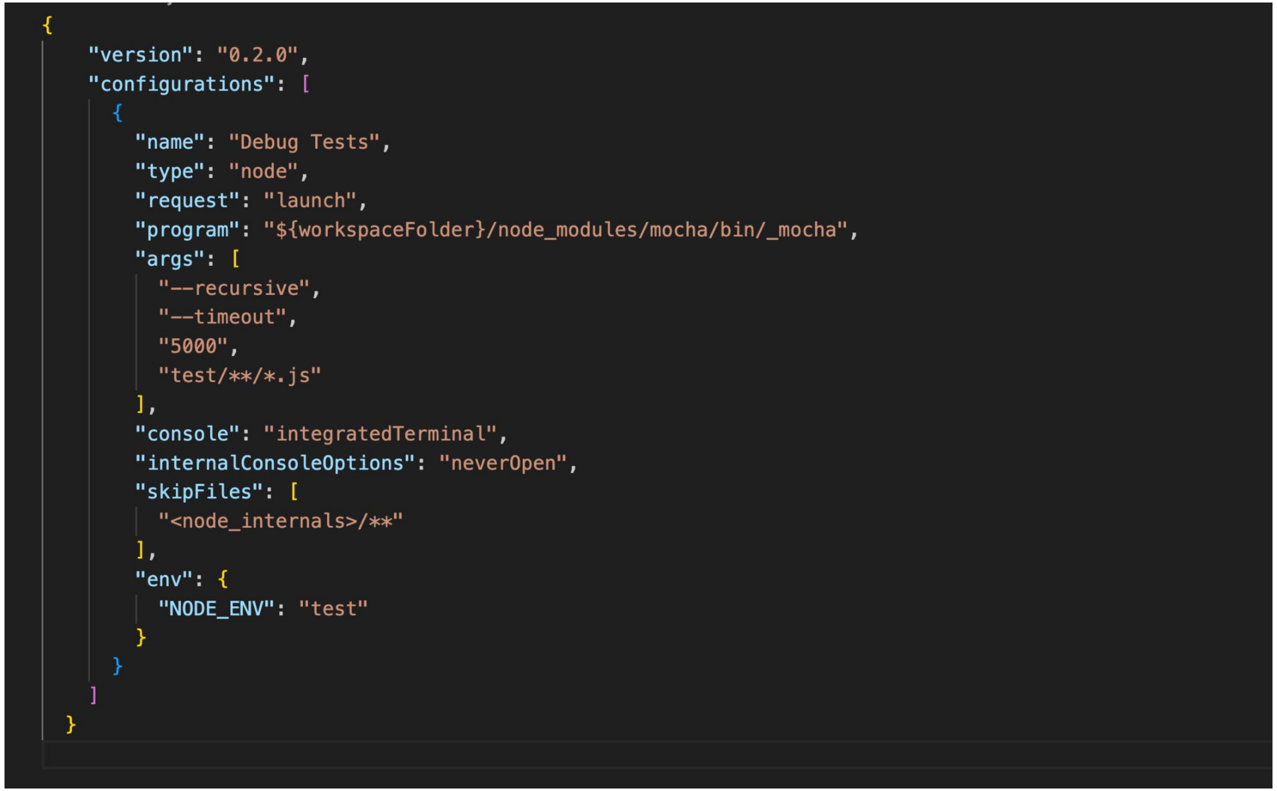
Review and Update

- Regularly review and update tests as your codebase evolves.
- Maintain the accuracy and relevance of your test suite.

Setting Debugger Node JS Unit tests

Airtribe

Setting up the VS Code Debugger



```
{
    "version": "0.2.0",
    "configurations": [
      {
        "name": "Debug Tests",
        "type": "node",
        "request": "launch",
        "program": "${workspaceFolder}/node_modules/mocha/bin/_mocha",
        "args": [
          "--recursive",
          "--timeout",
          "5000",
          "test/**/*.js"
        ],
        "console": "integratedTerminal",
        "internalConsoleOptions": "neverOpen",
        "skipFiles": [
          "<node_internals>/**"
        ],
        "env": {
          "NODE_ENV": "test"
        }
      }
    ]
  }
```

Airtribe

Thank You!

Airtribe