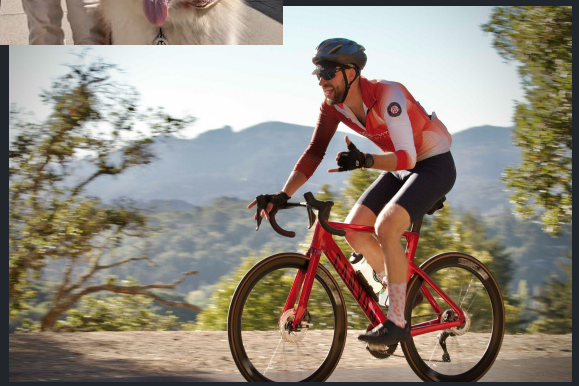
Test Driven Development

Tyler McGoffin - `@jtmcg`
Senior Software Engineer - GitHub CLI

Intro to `@jtmcg`



Professionally:

- Sr. Software Eng for GitHub CLI
- Formerly Manager and Engineer at CircleCI, Atlassian

Historically:

- B.S. Physics & B.S. Mathematics
- Research Scientist - Photovoltaics
- Teacher - Game Design & Coding
- Game Designer - Offworld Games

Personally:

- Triathlete
- D&D nerd
- Husband and dog dad

Agenda

- 👋 ~~Introduction to `@jtmcg`~~
- 🏰 Let’s start with a story...
- 🚀 Test Driven Development
- 🤖 AI & automation
- 🤔 Q&A

Story time

“

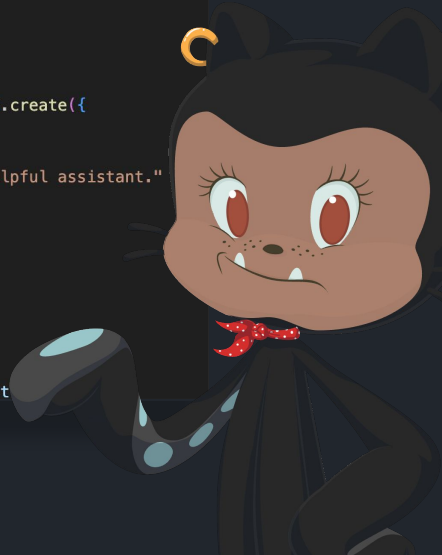
I need a feature that will chat with OpenAI via a file

Your Product Manger, once upon a time

Easy-peasy!

You, the baddest developer in the room

```
const queryOpenAI = async () => {
    // Set up the OpenAI Client
    const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
    const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
    const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID

    const openai = new OpenAI({
        organization: ORGANIZATION_ID,
        project: PROJECT_ID,
        apiKey: API_KEY,
    });

    // Read the query from our text file
    const filePath = "./query.txt";
    const query = fs.readFileSync(filePath, "utf8");

    // Query OpenAI
    const completion = await openai.chat.completions.create({
        model: "gpt-4o-mini",
        messages: [
            { role: "system", content: "You are a helpful assistant." },
            {
                role: "user",
                content: query,
            },
        ],
    });

    // Log the response
    console.log(completion.choices[0].message.content
```



“

Great! Now make it work for Claude, Gemini, and LLaMa too. Oh, and let users pick which one to use

Your Product Manger, twice upon a time?

“Cool, I’ll just change this here…”

Your teammate

```
const queryOpenAI = async () => {
    // Set up the OpenAI Client
    const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
    const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
    const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID

    const openai = new OpenAI({
        organization: ORGANIZATION_ID,
        project: PROJECT_ID,
        apiKey: API_KEY,
    });

    // Read the query from our text file
    const filePath = "./query.txt";
    const query = fs.readFileSync(filePath, "utf8");

    // Query OpenAI
    const completion = await openai.chat.completions.create({
        model: "gpt-4o-mini",
        messages: [
            { role: "system", content: "You are a helpful assistant." },
            {
                role: "user",
                content: query,
            },
        ],
    });

    // Log the response
    console.log(completion.choices[0].message.content);
```

How do you know it all still works?

Why write tests?

Reliability:

- Early bug detection
- Automation

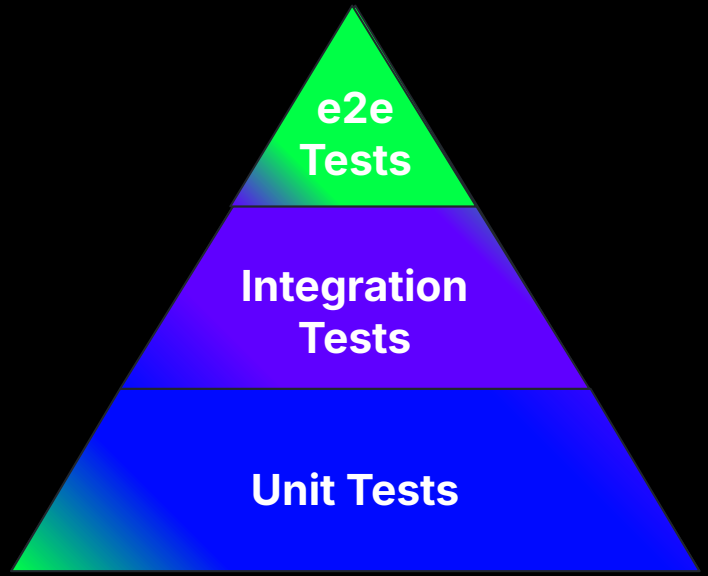
Maintainability:

- Safe refactoring
- Edge case confidence/discovery
- Reduction of tech debt

Productivity:

- Facilitates collaboration
- Developer confidence
- Faster delivery
- Documentation of behavior

The testing pyramid

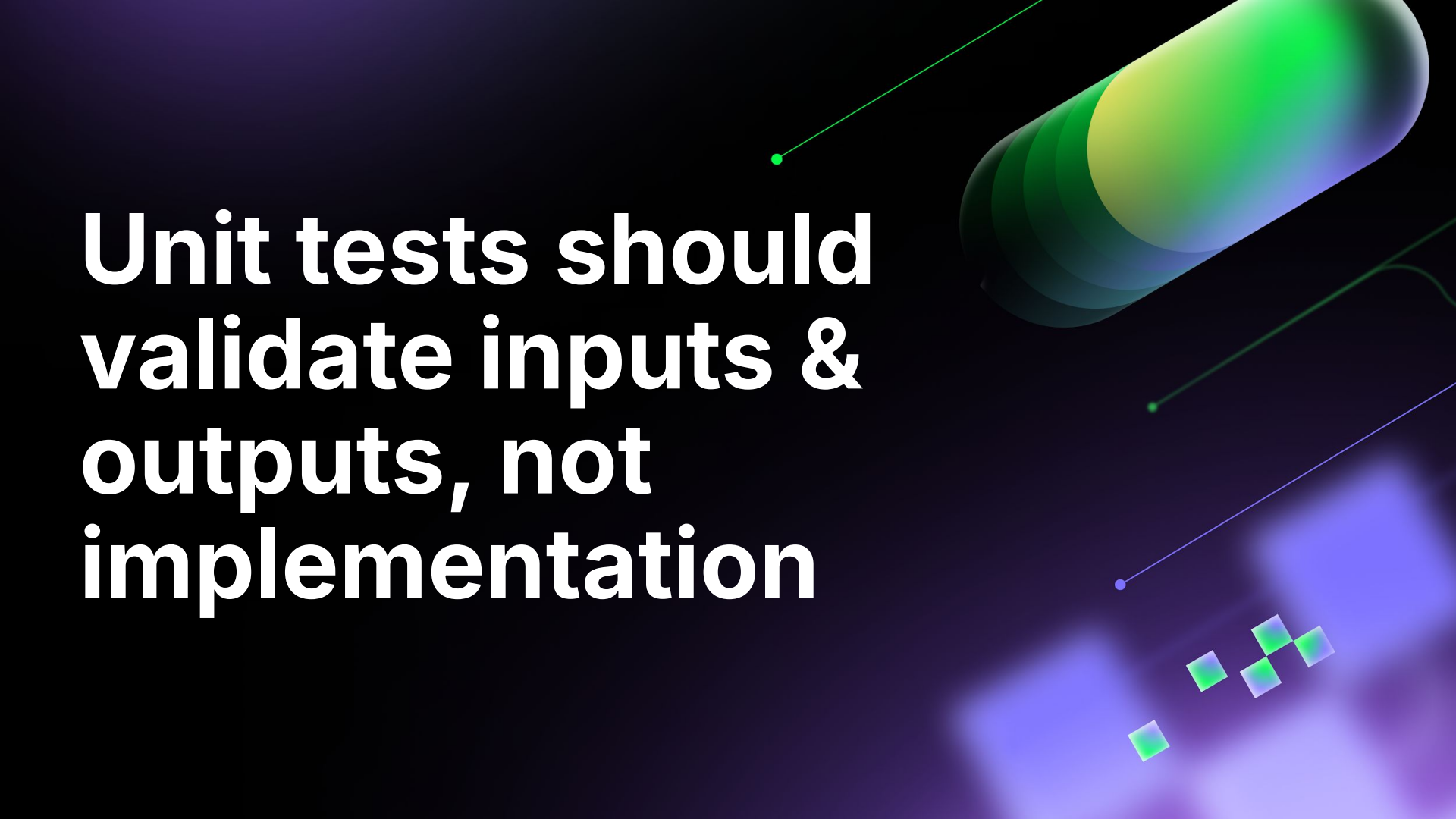


Unit Tests
Verify units of code are working as expected

Integration Tests
Verify features are working as expected

End-to-end (e2e) Tests
Verify features are working in production

Unit tests should validate inputs & outputs, not implementation



Code that is easy to test is good code

Idiomatic
Facilitates language-specific patterns in coding

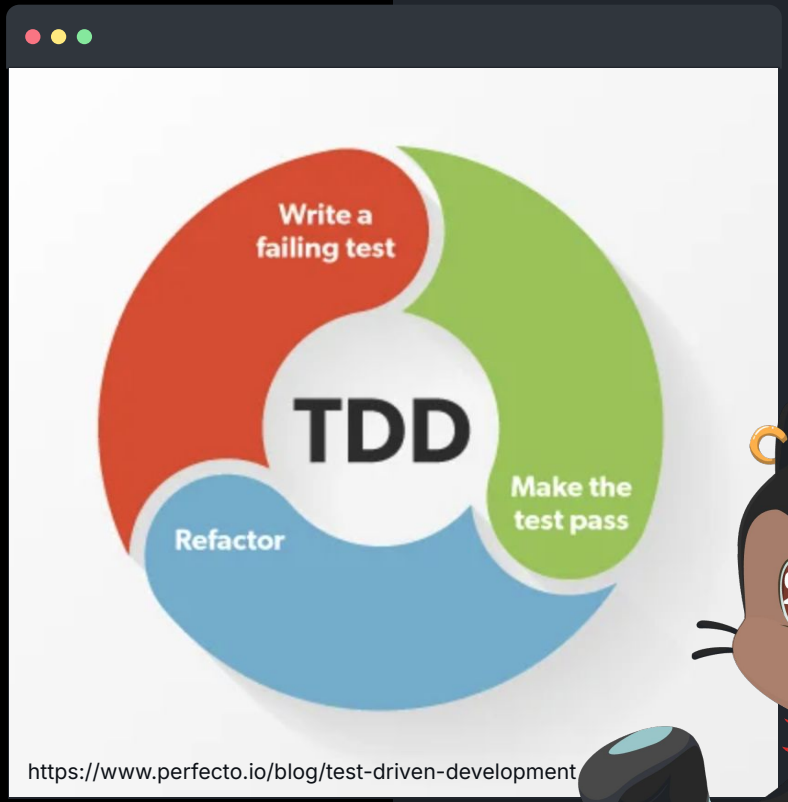
Extensible
Small, orthogonal, and composable pieces of code

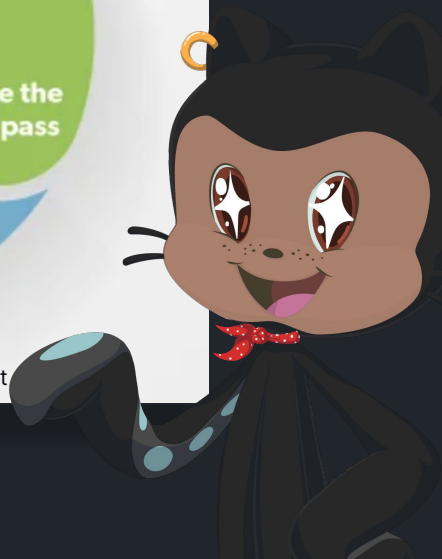
Understandable
Easy to read and maintain

Test Driven Development (TDD)

"

I need a feature that will chat with OpenAI via a file





Principles of TDD

Write tests first
You must have a failing test in place before writing code to make it pass

Keep it simple & stupid
Results in easy to understand code, reliable tests, and novel solutions

Refactor often
Be willing to change everything to accommodate the next test

Don’t overtest
If you can’t write a new failing test, you’re done

Red: Write a failing test

goodExample > JS goodExample.test.js > ...

```
1  import queryOpenAI from './goodExample'
2
3  describe('goodExample', () => {
4      it('should return a response string', () => {
5          expect(typeof queryOpenAI()).toBe("string");
6      })
7  })
```

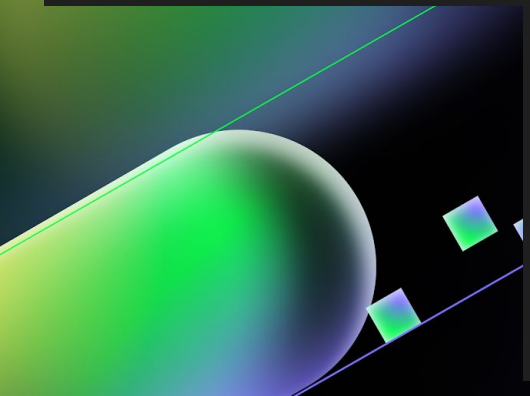
Green: Make the test pass

goodExample > JS goodExample.test.js > ...

```
1  import queryOpenAI from './goodExample'
2
3  describe('goodExample', () => {
4      it('should return a response string', () => {
5          expect(typeof queryOpenAI()).toBe("string");
6      })
7  })
```

oodExample > JS goodExample.js > [⊘] default

```
1  const queryOpenAI = () => {
2      return "Asshole Driven Development 😝"
3  }
4
5  export default queryOpenAI;
```



Repeat: Red → Green

```
import queryOpenAI from './goodExample'

const mockCreate = jest.fn();

describe('goodExample', () => {
    var openAIClientMock;

    beforeEach(() => {
        mockCreate.mockReset();
        openAIClientMock = {
            chat: {
                completions: {
                    create: mockCreate
                }
            }
        }
    })

    it('should return a response string', () => {
        expect(typeof queryOpenAI()).toBe("string");
    })

    it('should call OpenAI for a completions', () => {
        queryOpenAI(openAIClientMock);
        expect(mockCreate).toBeCalled();
    })
})
```

```
const queryOpenAI = (openAIClient) => {

    const completion = openAIClient.chat.completions.create()
    return "Asshole Driven Development 😝"
}

export default queryOpenAI;
```

```
$ npm test ./doItLive/goodExample.test.js

> tdd-workshop@0.0.0 test
> jest ./doItLive/goodExample.test.js

FAIL doItLive/goodExample.test.js
  secondTest
    ✓ should return a response string (2 ms)
    × should call OpenAI for a completion (1 ms)

  ● secondTest › should call OpenAI for a completion

    expect(jest.fn()).toBeCalled()

    Expected number of calls: >= 1
    Received number of calls:    0

      23 |     it('should call OpenAI for a completion', () => {
      24 |         queryOpenAI(openAIClientMock);
    > 25 |         expect(mockCreate).toBeCalled();
         |                            ^
      26 |     })
      27 | })

      at Object.toBeCalled (doItLive/goodExample.test.js:25:28)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 passed, 2 total
Snapshots:   0 total
Time:        2.176 s
Ran all test suites matching /.\/doItLive\/goodExample.test.js/i.
```

Refactor

```
import queryOpenAI from './goodExample'

const mockCreate = jest.fn();

describe('goodExample', () => {
    var openAIClientMock;

    beforeEach(() => {
        mockCreate.mockReset();
        openAIClientMock = {
            chat: {
                completions: {
                    create: mockCreate
                }
            }
        }
    })

    it('should return a response string', () => {
        expect(typeof queryOpenAI(openAIClientMock))
    })

    it('should call OpenAI for a completions', () =>
        queryOpenAI(openAIClientMock);
        expect(mockCreate).toBeCalled();
    })
})
```

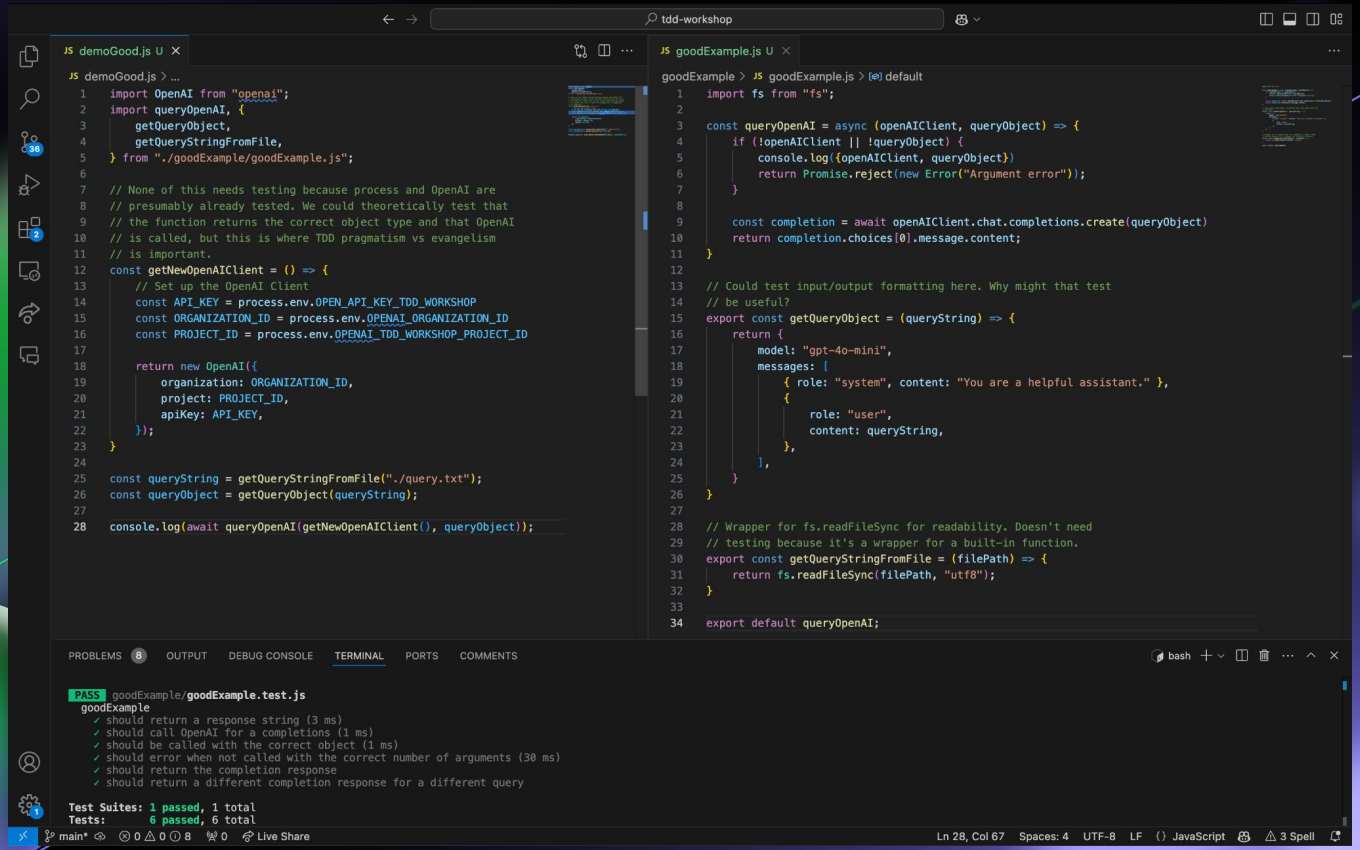
```
const queryOpenAI = (openAIClient) => {

    const completion = openAIClient.chat.completions.create()
    return "Asshole Driven Development 😝"
}

export default queryOpenAI;
```

```
$ npm test ./doItLive/goodExample.test.js

> tdd-workshop@0.0.0 test
> jest ./doItLive/goodExample.test.js

 PASS  doItLive/goodExample.test.js
  secondTest
    ✓ should return a response string (1 ms)
    ✓ should call OpenAI for a completion

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.278 s
Ran all test suites matching /.\/doItLive\/goodExample.test.js/i.
```

Red, Green, Refactor, Repeat...

See the doltLive folder in the repo

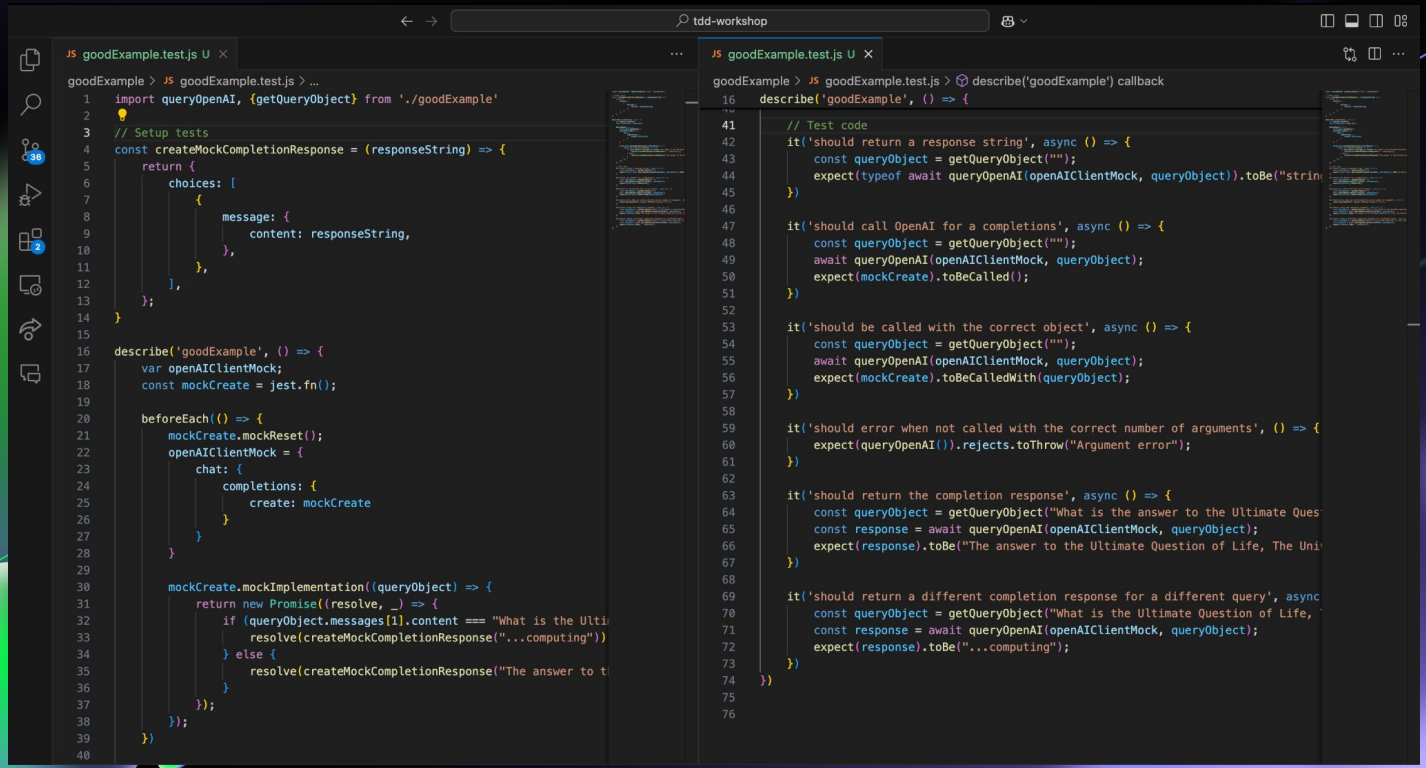
The Final Solution




```
import OpenAI from "openai";
import queryOpenAI, {
    getQueryObject,
    getQueryStringFromFile,
} from "./goodExample/goodExample.js";

// None of this needs testing because process and OpenAI are
// presumably already tested. We could theoretically test that
// the function returns the correct object type and that OpenAI
// is called, but this is where TDD pragmatism vs evangelism
// is important.
const getNewOpenAIClient = () => {
    // Set up the OpenAI Client
    const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
    const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
    const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID

    return new OpenAI({
        organization: ORGANIZATION_ID,
        project: PROJECT_ID,
        apiKey: API_KEY,
    });
}

const queryString = getQueryStringFromFile("./query.txt");
const queryObject = getQueryObject(queryString);

console.log(await queryOpenAI(getNewOpenAIClient(), queryObject));
```

```
import fs from "fs";

const queryOpenAI = async (openAIClient, queryObject) => {
    if (!openAIClient || !queryObject) {
        console.log({openAIClient, queryObject})
        return Promise.reject(new Error("Argument error"));
    }

    const completion = await openAIClient.chat.completions.create(queryObject)
    return completion.choices[0].message.content;
}

// Could test input/output formatting here. Why might that test
// be useful?
export const getQueryObject = (queryString) => {
    return {
        model: "gpt-4o-mini",
        messages: [
            { role: "system", content: "You are a helpful assistant." },
            {
                role: "user",
                content: queryString,
            },
        ],
    }
}

// Wrapper for fs.readFileSync for readability. Doesn't need
// testing because it's a wrapper for a built-in function.
export const getQueryStringFromFile = (filePath) => {
    return fs.readFileSync(filePath, "utf8");
}

export default queryOpenAI;
```

The Final Tests



Holy crap, Tyler, that's a lot of work...

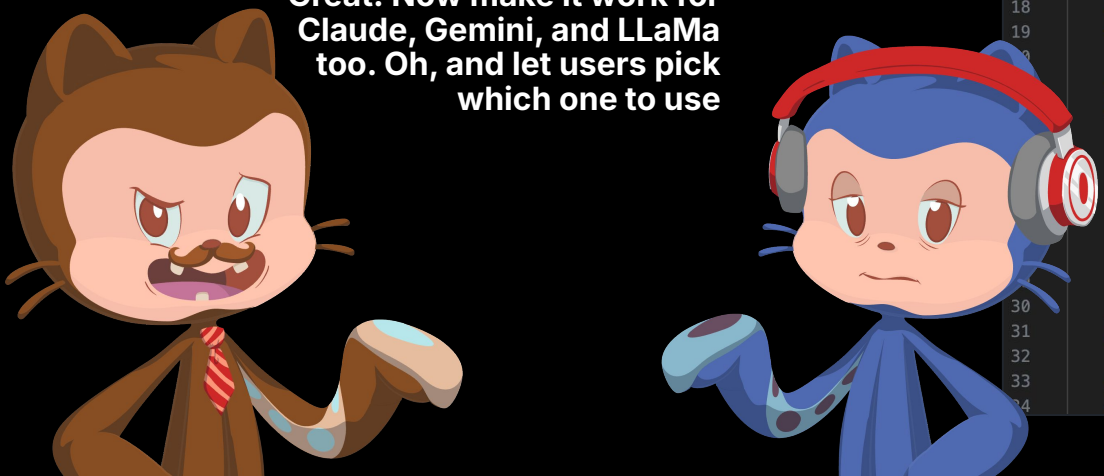
Compare Solutions

```
3   const queryOpenAI = async (openAIClient, queryObject) => {
4       if (!openAIClient || !queryObject) {
5           console.log({openAIClient, queryObject})
6           return Promise.reject(new Error("Argument error"));
7       }
8
9       const completion = await openAIClient.chat.completions.create(queryObject)
10      return completion.choices[0].message.content;
11  }
```

```
4
5   const queryOpenAI = async () => {
6       // Set up the OpenAI Client
7       const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
8       const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
9       const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID
10
11      const openai = new OpenAI({
12          organization: ORGANIZATION_ID,
13          project: PROJECT_ID,
14          apiKey: API_KEY,
15      });
16
17      // Read the query from our text file
18      const filePath = "./query.txt";
19      const query = fs.readFileSync(filePath, "utf8");
20
21      // Query OpenAI
22      const completion = await openai.chat.completions.create({
23          model: "gpt-4o-mini",
24          messages: [
25              { role: "system", content: "You are a helpful assistant." },
26              {
27                  role: "user",
28                  content: query,
29              },
30          ],
31      });
32
33      // Log the response
34      console.log(completion.choices[0].message.content);
```

Compare Solutions

```
3   const queryOpenAI = async (openAIClient, queryObject) => {
4       if (!openAIClient || !queryObject) {
5           console.log({openAIClient, queryObject})
6           return Promise.reject(new Error("Argument error"));
7       }
8
9       const completion = await openAIClient.chat.completions.create(queryObject)
10      return completion.choices[0].message.content;
11  }
```




```
5   const queryOpenAI = async () => {
6       // Set up the OpenAI Client
7       const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
8       const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
9       const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID
10
11      const openai = new OpenAI({
12          organization: ORGANIZATION_ID,
13          project: PROJECT_ID,
14          apiKey: API_KEY,
15      });
16
17      // Read the query from our text file
18      const filePath = "./query.txt";
19      const query = fs.readFileSync(filePath, "utf8");
20
21      // Query OpenAI
22      const completion = await openai.chat.completions.create({
23          model: "gpt-4o-mini",
24          messages: [
25              { role: "system", content: "You are a helpful assistant." },
26              {
27                  role: "user",
28                  content: query,
29              },
30          ],
31      });
32
33      // Log the response
34      console.log(completion.choices[0].message.content);
```

Compare Solutions

```
3   const queryOpenAI = async (openAIClient, queryObject) => {
4       if (!openAIClient || !queryObject) {
5           console.log({openAIClient, queryObject})
6           return Promise.reject(new Error("Argument error"));
7       }
8
9       const completion = await openAIClient.chat.completions.create(queryObject)
10      return completion.choices[0].message.content;
11  }
```

Guess I'll just pass this a different client

```
5   const queryOpenAI = async () => {
6       // Set up the OpenAI Client
7       const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
8       const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
9       const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID
10
11      const openai = new OpenAI({
12          organization: ORGANIZATION_ID,
13          project: PROJECT_ID,
14          apiKey: API_KEY,
15      });
16
17      // Read the query from our text file
18      const filePath = "./query.txt";
19      const query = fs.readFileSync(filePath, "utf8");
20
21      // Query OpenAI
22      const completion = await openai.chat.completions.create({
23          model: "gpt-4o-mini",
24          messages: [
25              { role: "system", content: "You are a helpful assistant." },
26              {
27                  role: "user",
28                  content: query,
29              },
30          ],
31      });
32
33      // Log the response
34      console.log(completion.choices[0].message.content);
```

Key Takeaways

Write tests WITH code
Easy to test = good code and testing as you code begets good code the first time

TDD can be fun
Gamify your development experience with TDD, especially while pairing

Test interfaces, not implementation
Your tests shouldn’t care about how the code is written

Pragmatism > perfectionism
Beware TDD for the sake of TDD and overtesting

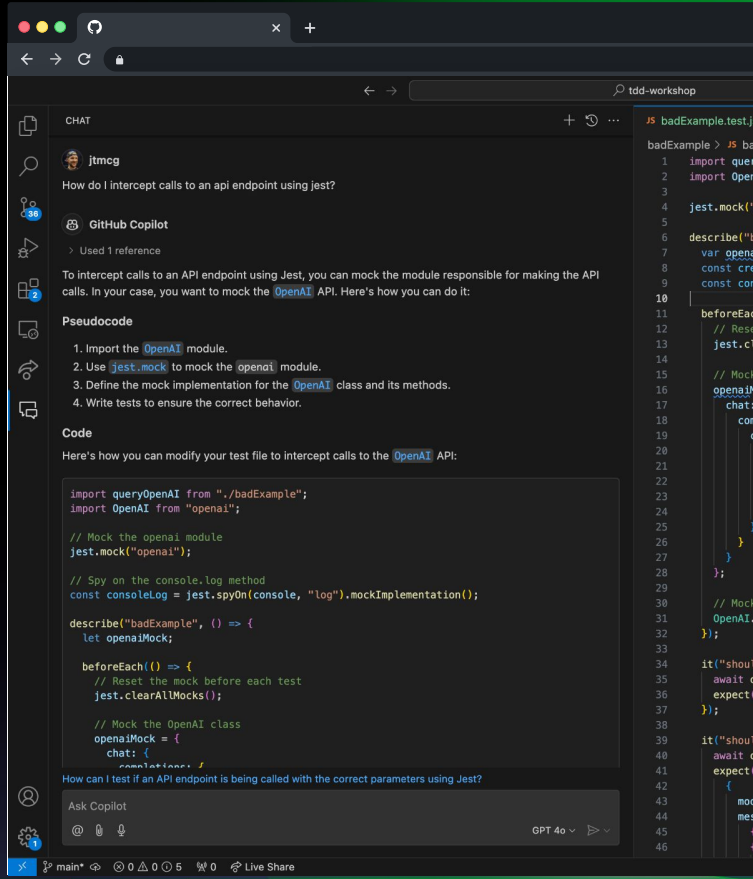
🤖 GitHub Copilot and Tests

Testing frameworks can be hard to learn

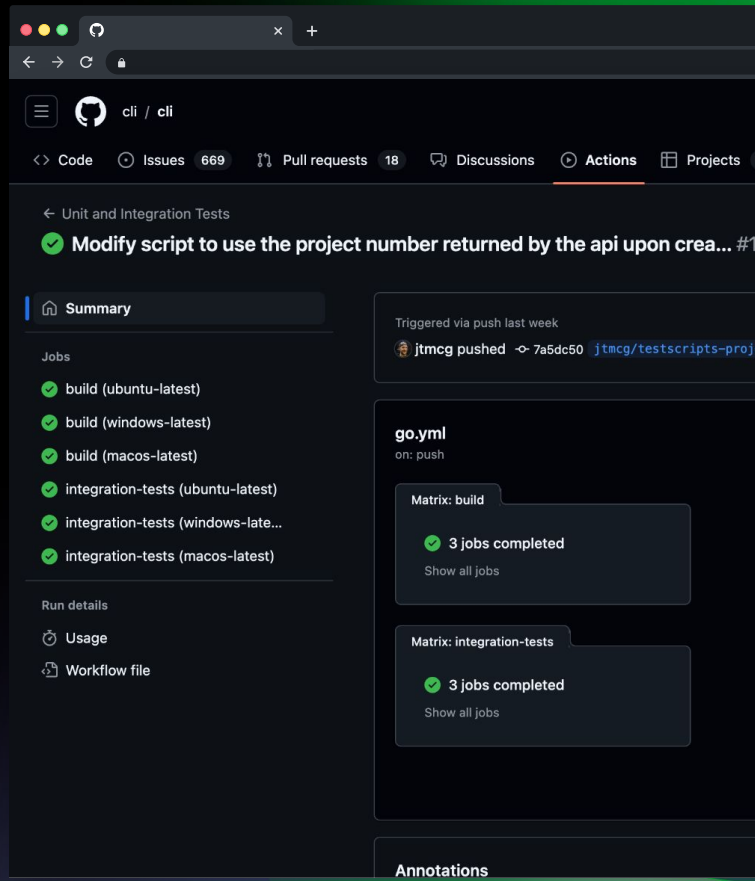
Copilot can help accelerate this learning curve

Tests are logically simpler than code

Copilot is really good at writing tests!



GitHub Actions & CI/CD



Continuous Integration

Automate tests and checks before merging code

Continuous Delivery

Automate deployments of new code

Resources

Workshop Repo
github.com/jtmcg/tdd-workshop

Jest testing framework for JS

jestjs.io/docs/getting-started

OpenAI Docs

platform.openai.com/docs/overview

GitHub Actions

docs.github.com/en/actions

GitHub Copilot

github.com/features/copilot


🤔 Q&A


Thank you