AlphaSense

Testing Boundary

May 2025

Tech Talk

Testing Boundary

AlphaSense

01 Levels of Testing

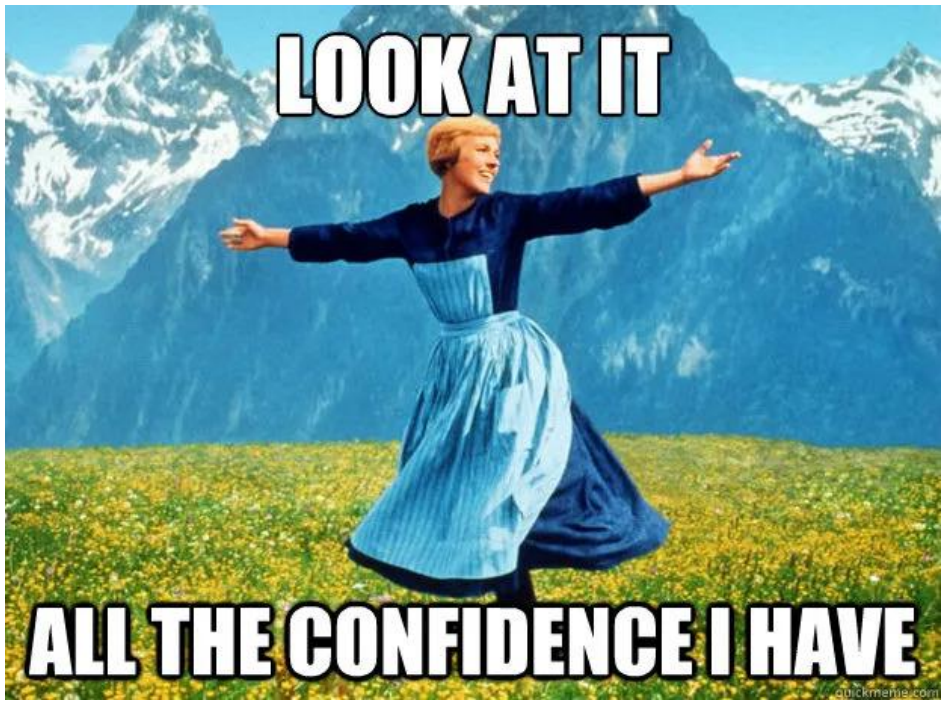
02 Testing Boundary

03 Jump into code

01.

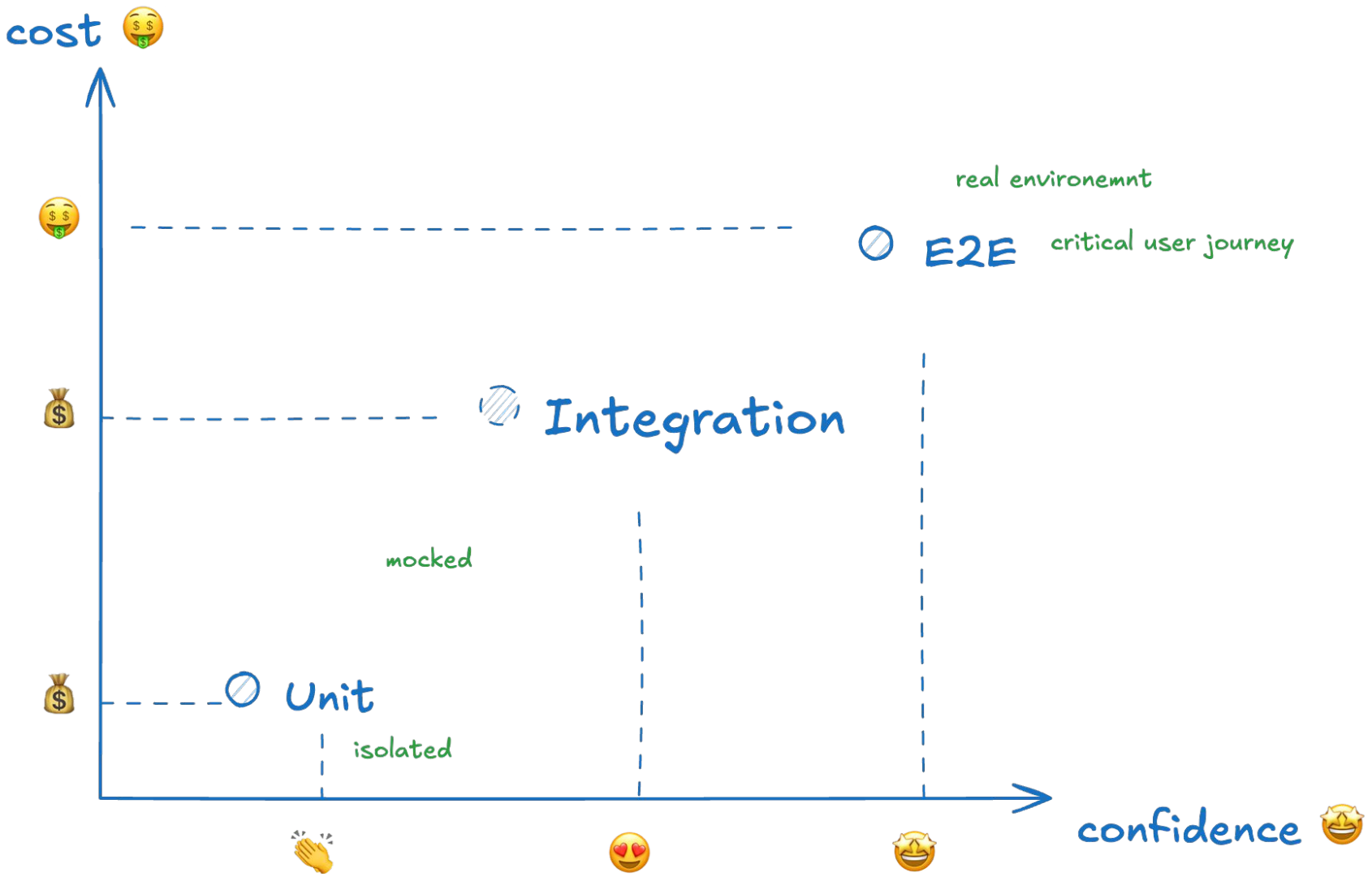
Levels Of Testing

Confidence

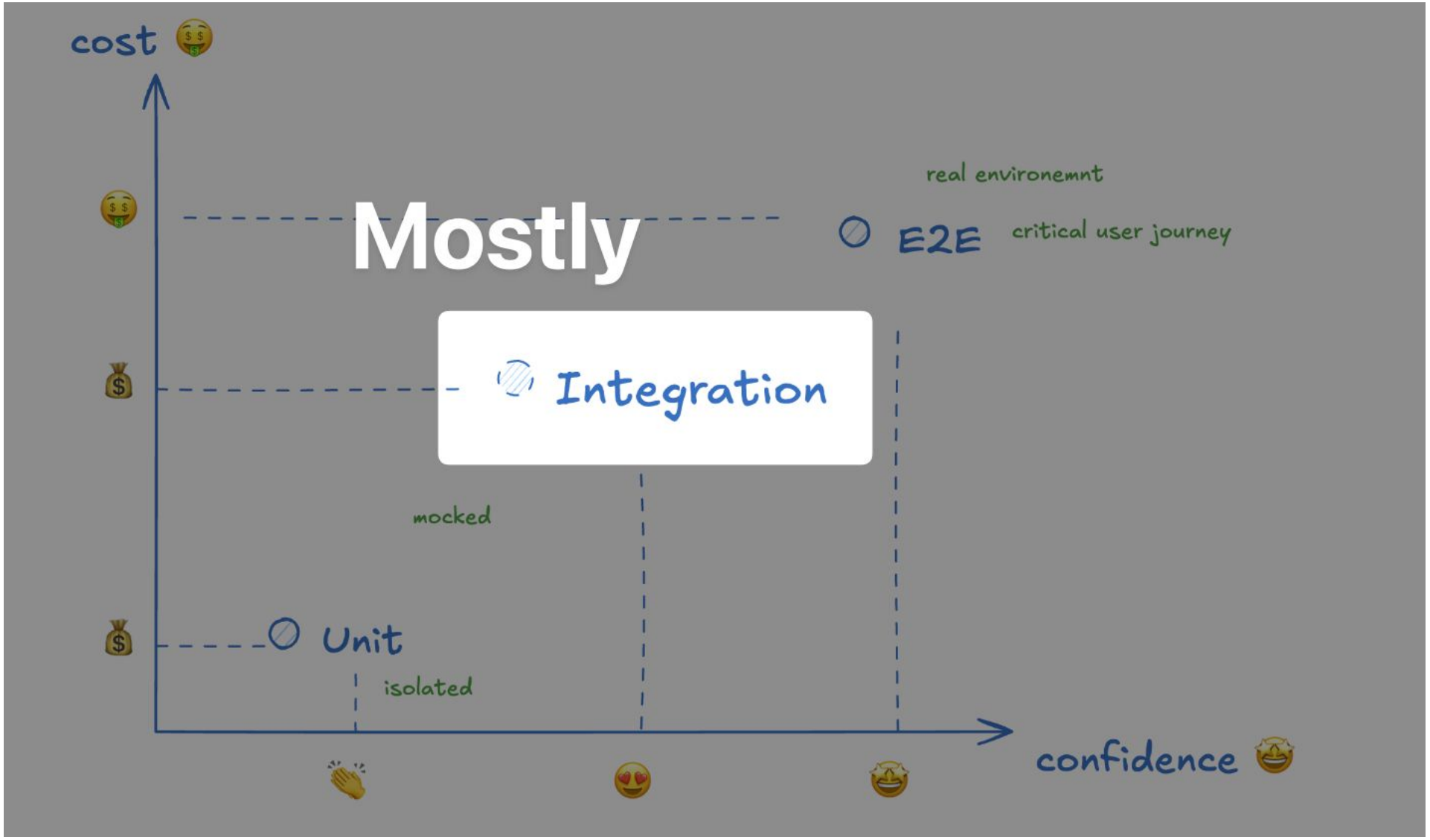


AlphaSense

Levels Of Testing



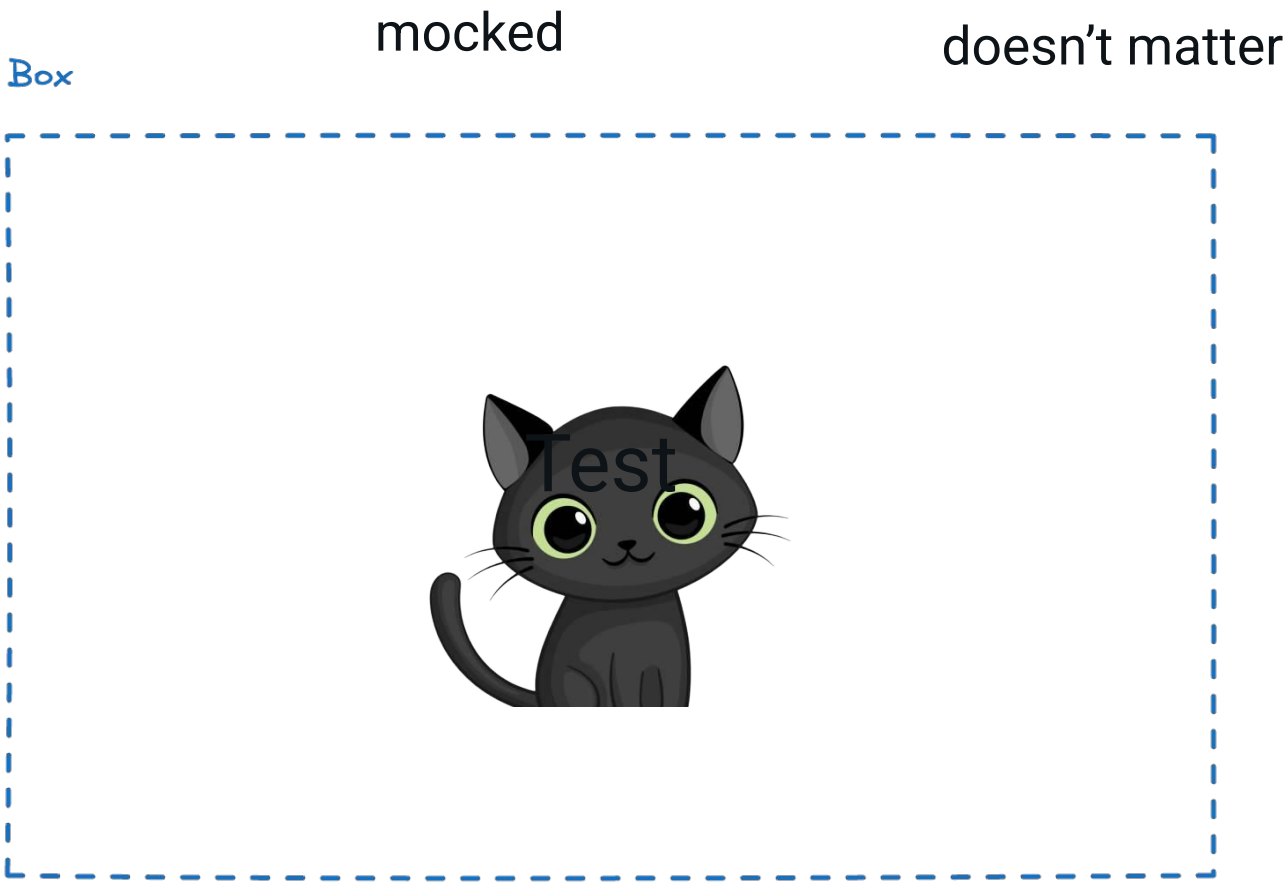
🚀 Tip 1


cost
real environemnt
Mostly
E2E
critical user journey
Integration
mocked
Unit
isolated
confidence

02

Testing Boundary

Testing Boundary



Example

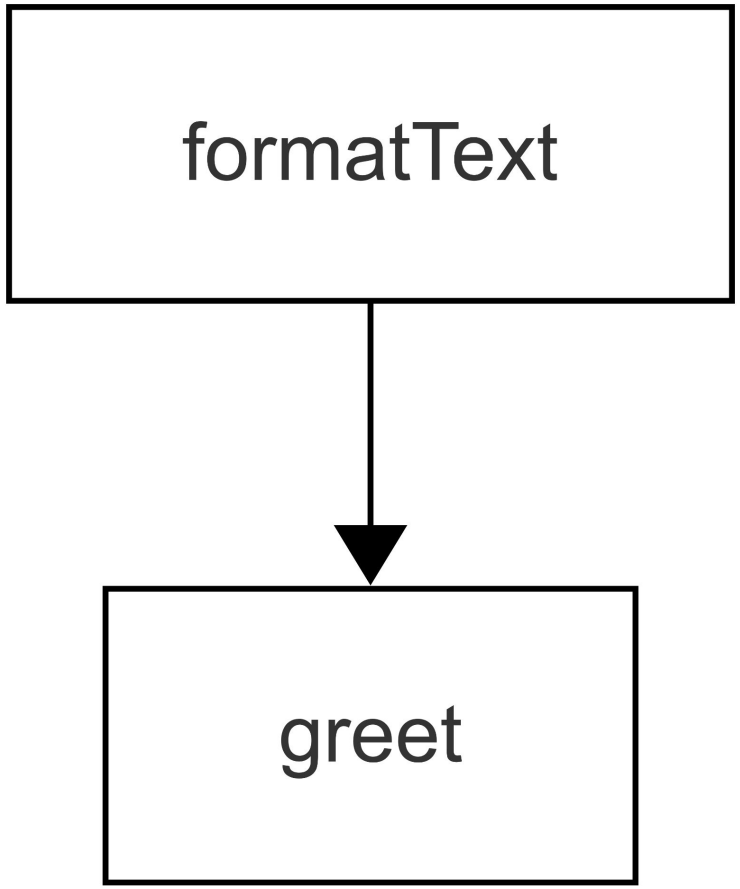
src > index.tsx > ...

```tsx
const formatText = (text: string) => text.trim().toUpperCase();

function greet(name: string) {
  const formattedName = formatText(name);
  return `Hello, ${formattedName}!`;
}

// Hello, Huy!
greet('huy');
```

❓ If I write a test for the **greet function**, is that **a unit test or an integration test**?

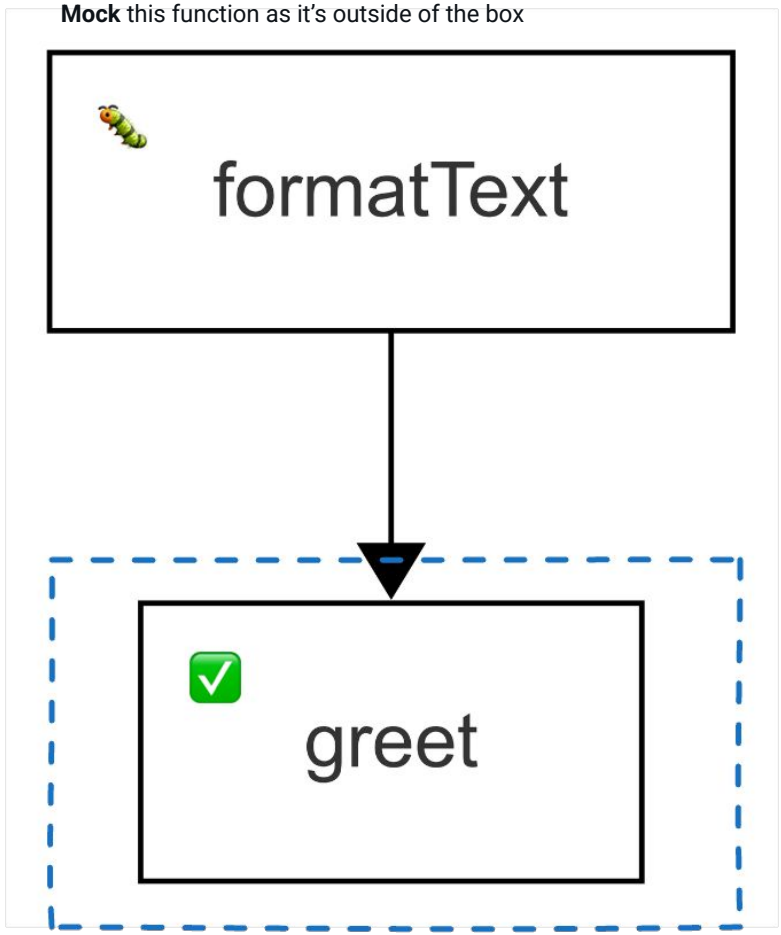
it depends on how we set the boundary

Dependency Chart



❓ How should we draw the boundary line?

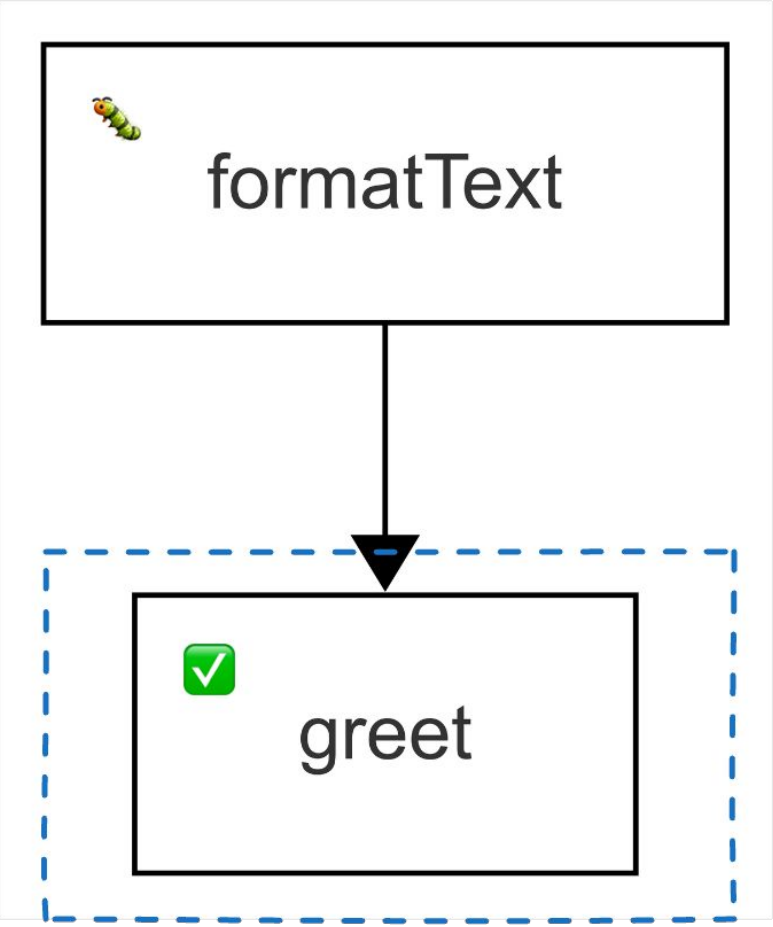
Unit Test



src > index.tsx > ...

```
1  const formatText = (text: string) => text.trim().toUpperCase();
2
3  function greet(name: string) {
4    const formattedName = formatText(name); (mocked)
5    return `Hello, ${formattedName}!`;
6  }                                          tested
7
8  // Hello, Huy!
9  greet('huy');
10
```

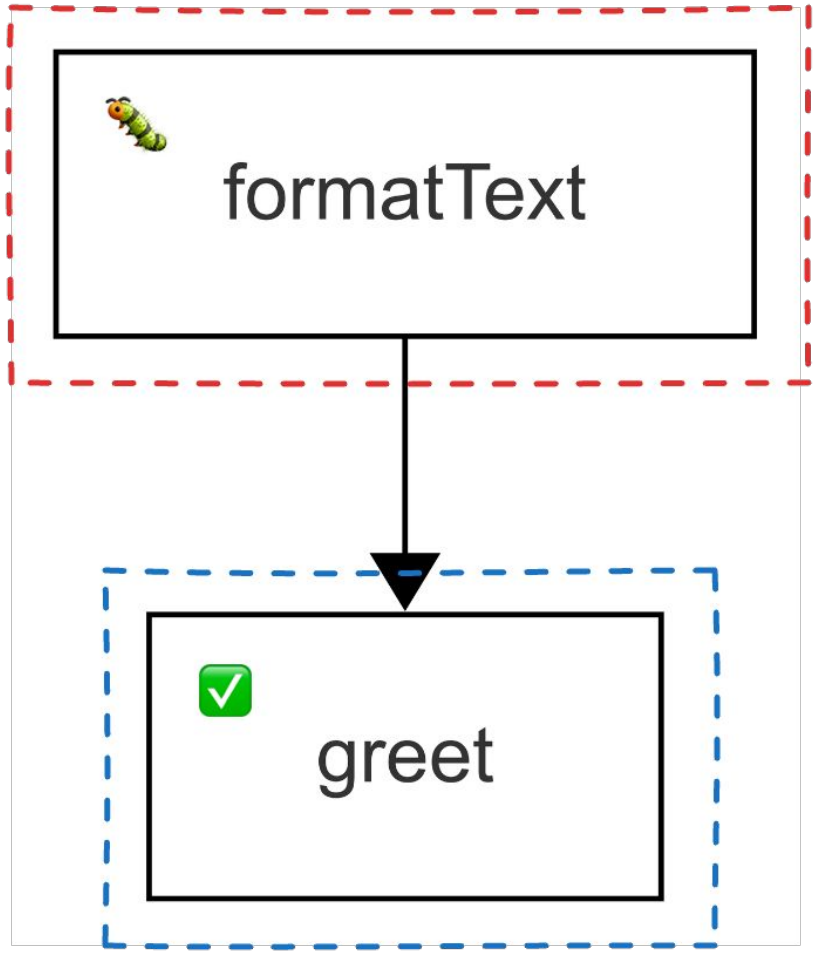
Unit Test

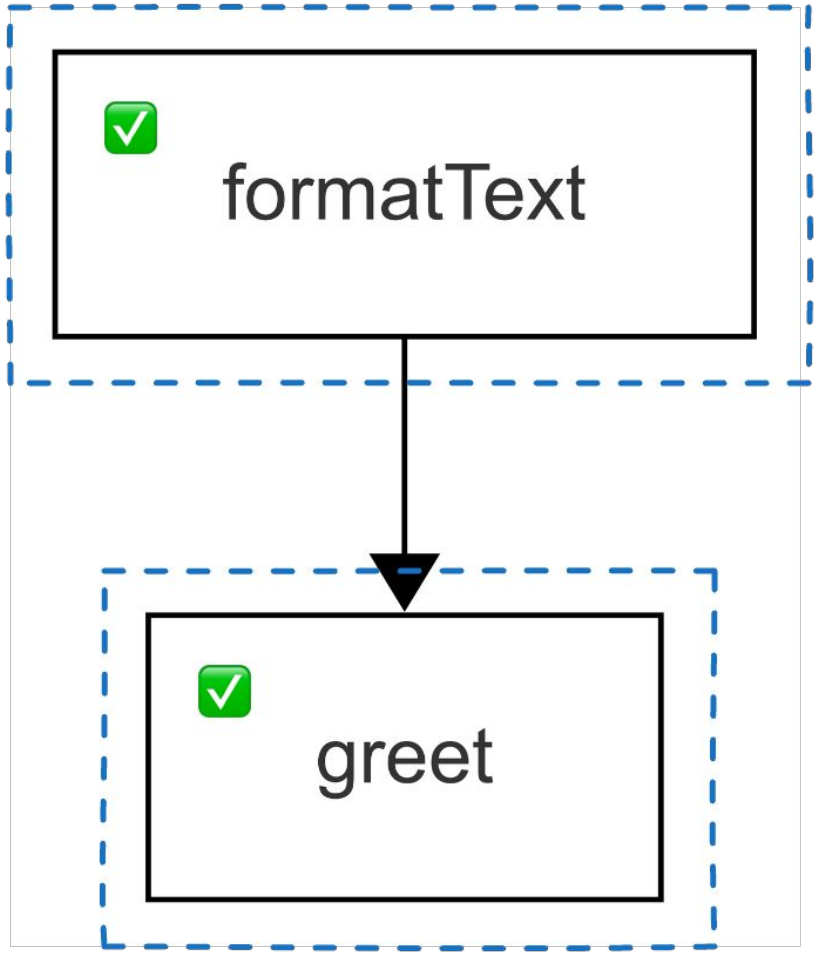



```
// 👇 Mock the formatText function
vi.mock('./index', () => ({
  formatText: vi.fn(),
}));

it('returns a greeting message', () => {
  // 👇 Mock return value
  vi.mocked(formatText).mockReturnValue('HUY');

  const result = greet('huy');
  expect(result).toBe('Hello, HUY!');
});
```

confident?

Unit Test - Increasing confidence by adding more tests

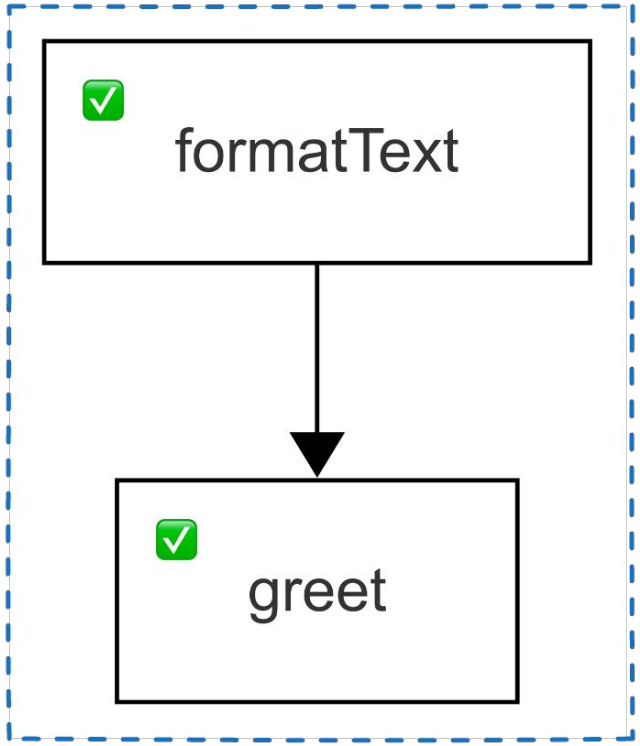




Isolated unit tests

AlphaSense

Integration Test



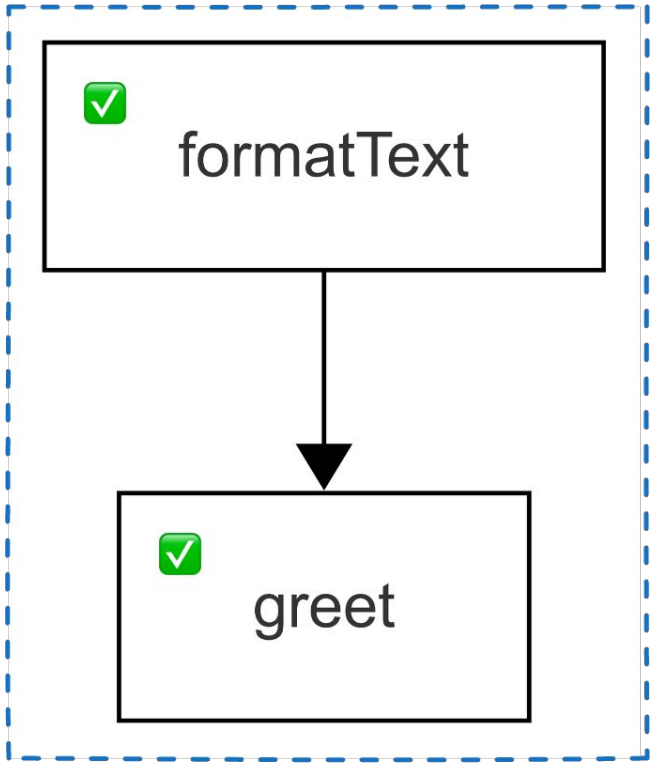
src > index.tsx > ...

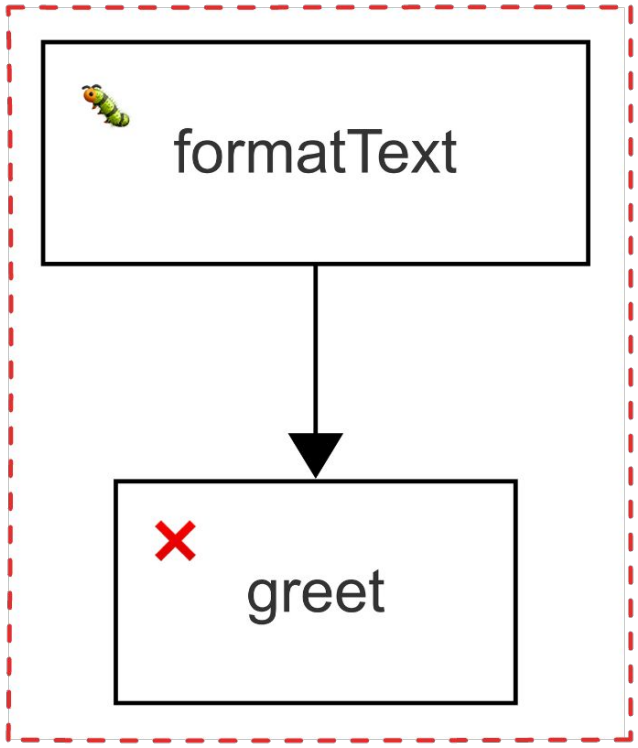
```
1  // Function 2
2  const formatText = (text: string) => text.trim().toUpperCase();
3
4  // Function 1
5  function greet(name: string) {
6    const formattedName = formatText(name);
7    return `Hello, ${formattedName}!`;
8  }
```

tested

AlphaSense

Integration Test






```
// Removed the mock
it('returns a greeting message', () => {
  const result = greet('huy');
  expect(result).toBe('Hello, HUY!');
});
```

simply remove the mocks

**Implicitly test the formatText* 🤩

Mocking

Not saying mocking is bad

It’s a very useful tool. We need it!

We need to be aware of its limitations and find the balance

The more we mock, the more our test **diverges** from the real application

🚀 Tip 2

Just mock less

🚀 Tip 3

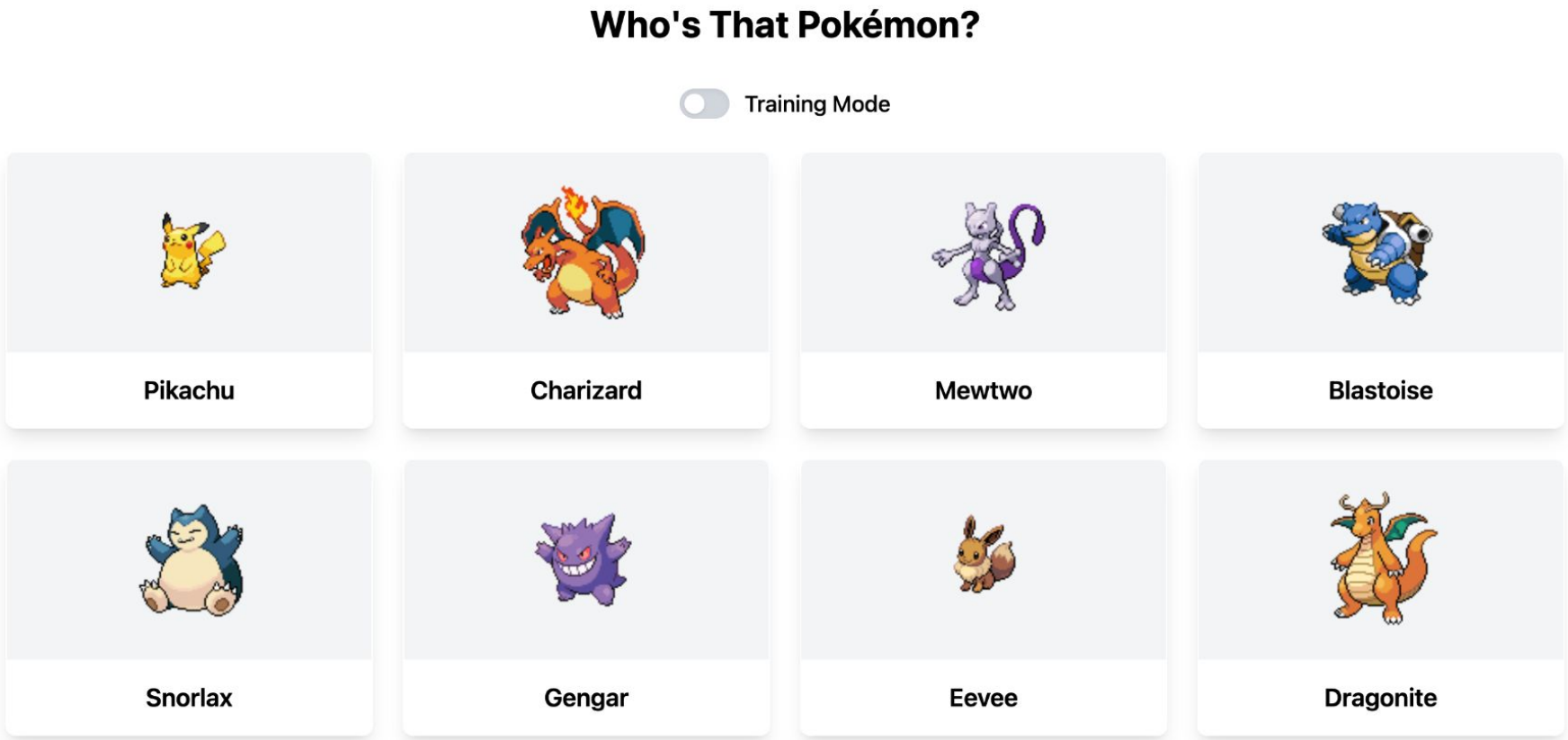
A **test must fail only when** an expectation is not met

A **test must pass** when an expectation is met

03

Real-world example

🚀 Pokemon app

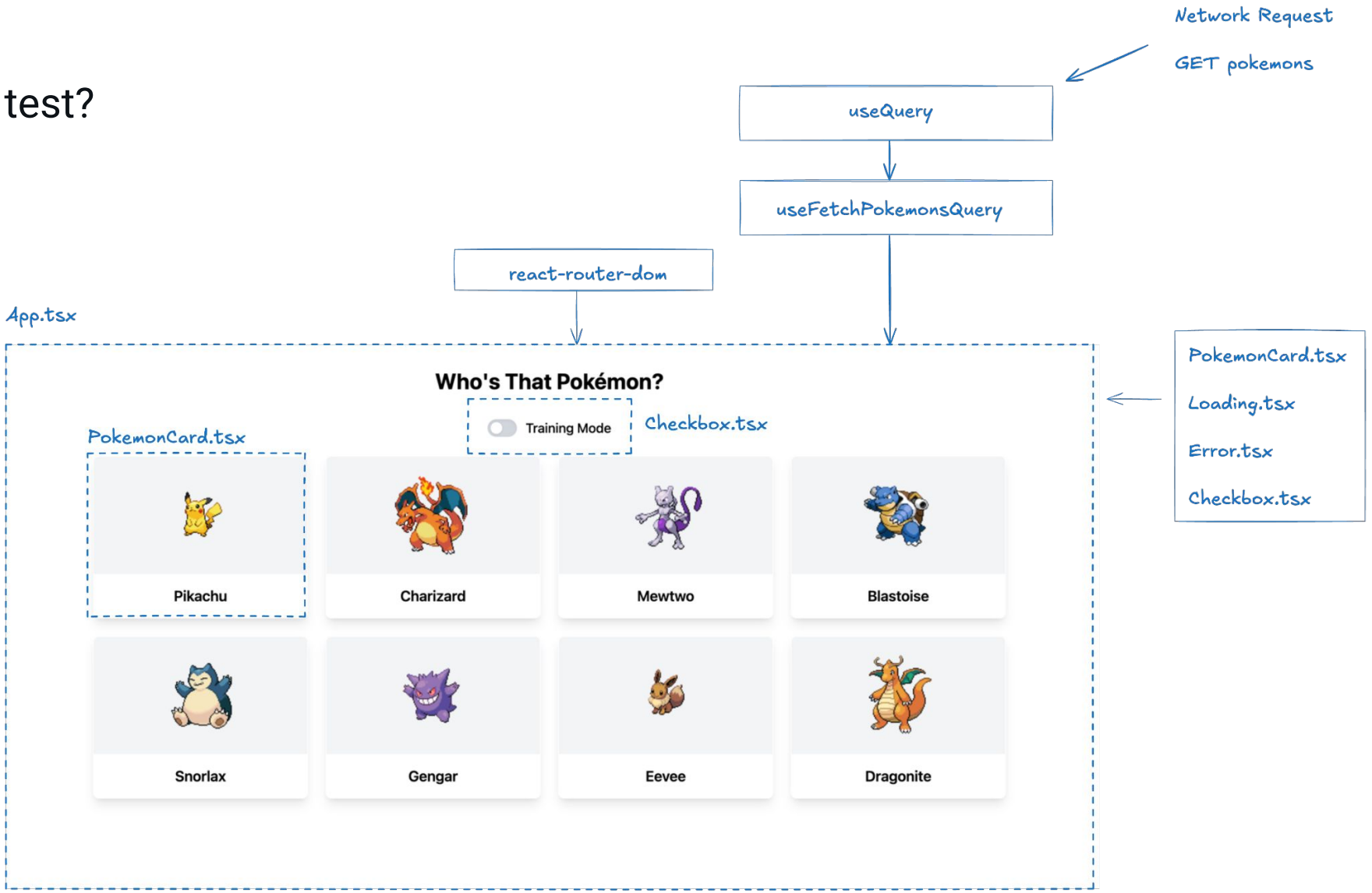


source-code

AlphaSense

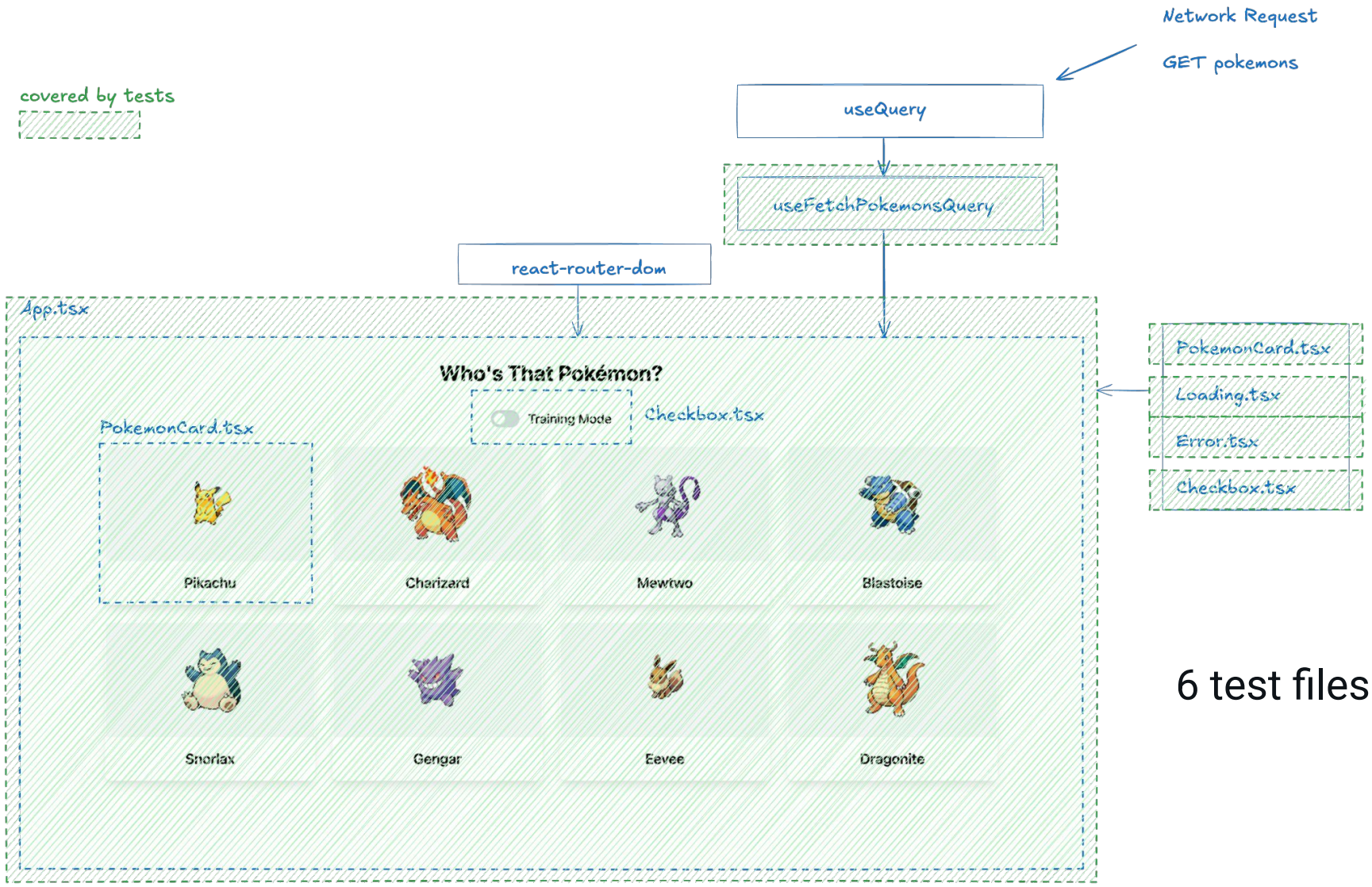
Dependency Tree

❓ What to test?



AlphaSense

Unit test? - Bottom up



6 test files

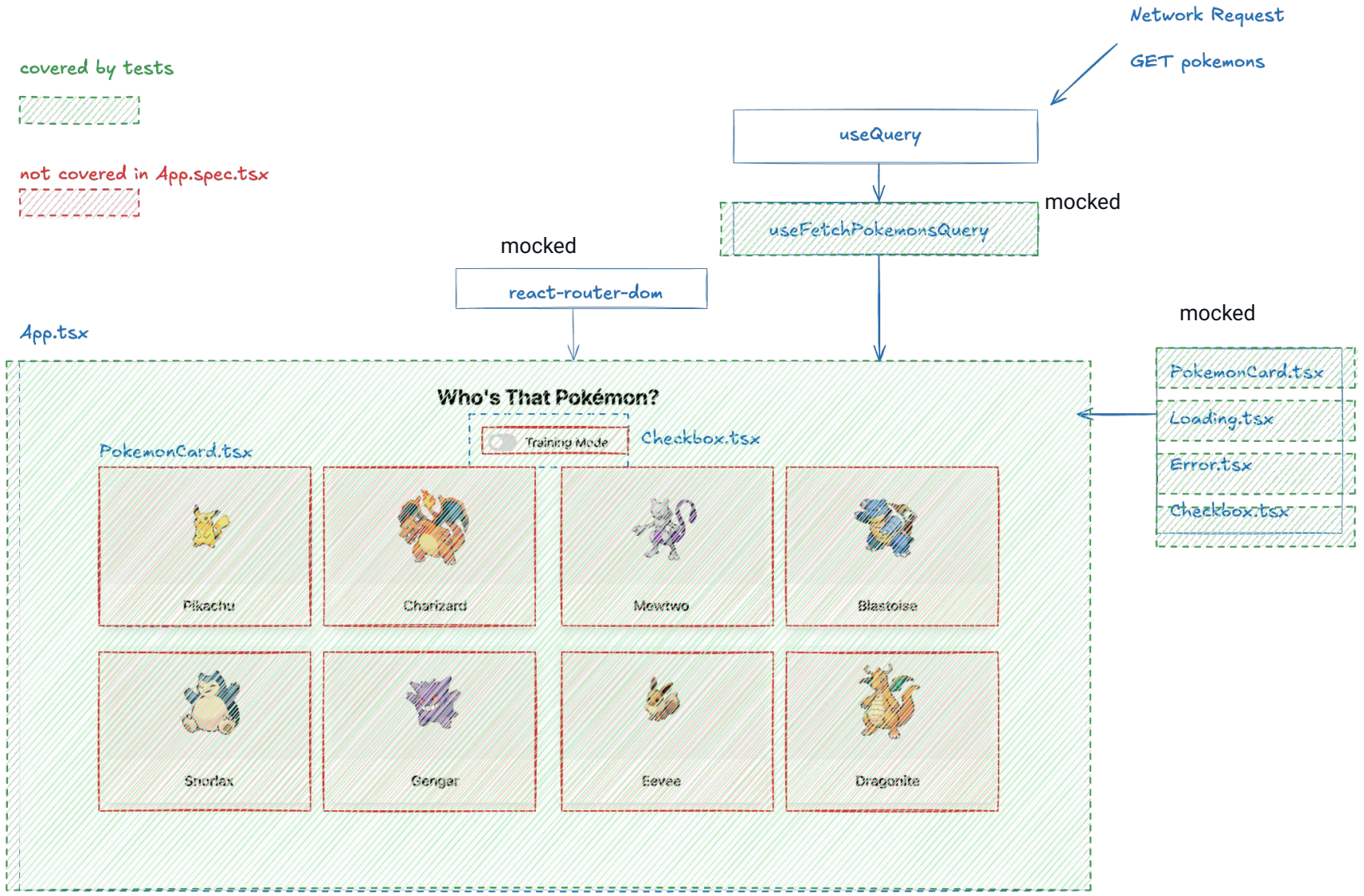
App is well covered with tests

Mostly unit tests

However, **are we confident** if the app actually works? 🤔

AlphaSense

We are “poking holes” around the App



📝 Test Plan

Test cases
1. Should **see the list of Pokémon cards** with all expected elements.
2. Should display a **loading message** while Pokémon data is being fetched
3. Should show an **error message** if fetching Pokémon data fails
4. Should hide pokemon details in the **training mode** and reveal on hover
5. Should start and **persist training mode based on the URL parameter** so the state can be shared and resumed.

the fetching hook should work

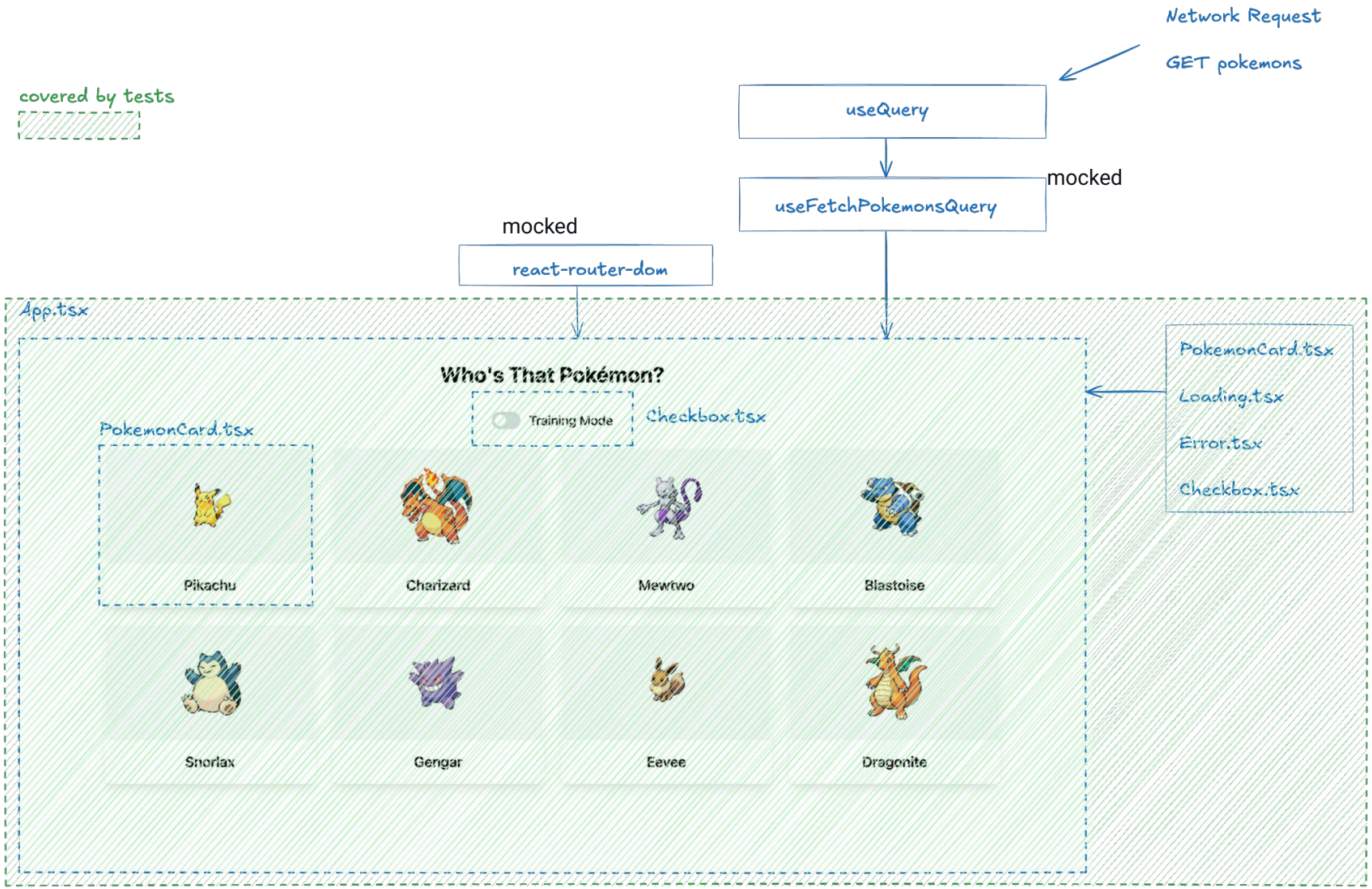
react router pushes location

testing implementation details

In fact, these are already covered in the Integration test (implicitly)

AlphaSense

Focus on Integration tests



✅ Better

We’re mocking
- react-router
- useFetchPokemonQuery

App.spec.tsx

Is that necessary to mock here?

Can we get rid of these mocks?

But....

What is the issue here?

```
vi.mock('react-router-dom', () => ({
  ...vi.importActual('react-router-dom'),
  useLocation: vi.fn().mockReturnValue({
    pathname: '/',
    search: '',
  }),
  useHistory: vi.fn(),
}));

vi.mock('./graphql/pokemons.graphql', () => ({
  useFetchPokemonsQuery: vi.fn(),
}));
```

Issue 1: Wrong mocked data

```
it('loading state', () => {
  vi.mocked(useFetchPokemonsQuery, {partial: true}).mockReturnValueOnce({
    data: {
      pokemons: MOCKED_POKEMONS,
    },
    loading: true,
    error: undefined,
  });

  renderComponent();

  screen.getByText(/loading.../i);
});
```

Bug

why do we have data here while loading is true?

What we're assuming while mocking is **not correct**

Thus, we have less confidence here - compared to not mocking this hook

AlphaSense

Issue 2: Incomplete mocked data

```
it('loading state', () => {
  vi.mocked(useFetchPokemonsQuery, {partial: true}).mockReturnValueOnce({
    data: {
      pokemons: MOCKED_POKEMONS,
    },
    loading: true,
    error: undefined,
  });

  renderComponent();

  screen.getByText(/loading.../i);
});
```

only 3 properties

Here we assume the hooks return only 3 properties

```
export interface QueryResult<TData = any, TVariables extends O
    /**…
    client: ApolloClient<any>;
    /**…
    observable: ObservableQuery<TData, TVariables>;
    /**…
    data: MaybeMasked<TData> | undefined;
    /**…
    previousData?: MaybeMasked<TData>;
    /**…
    error?: ApolloError;
    /**…
    errors?: ReadonlyArray<GraphQLFormattedError>;
    /**…
    loading: boolean;
    /**…
    networkStatus: NetworkStatus;
    /**…
    called: boolean;
}
```

In fact

Issue 3: Race condition + complexity

```
it('loading state', () => {
  vi.mocked(query1, {partial: true}).mockReturnValueOnce({
    data: {
      pokemons: MOCKED_POKEMONS,
    },
    loading: true,
    error: undefined,
  });

  vi.mocked(query2, {partial: true}).mockReturnValueOnce({
    data: {
      pokemons: MOCKED_POKEMONS,
    },
    loading: false,
    error: undefined,
  });

  renderComponent();

  screen.getByText(/loading.../i);
});
```

If we have 2 queries, **how should we mock their loading states?**

Should both be true or one true and one false?

Issue 4: Testing flow

```
> it('renders list of pokemons', () => {…
  });

> it('loading state', () => {…
  });

> it('error state', () => {…
  });
```

The 3 states **are tested in isolation**, but the user experiences them as a single flow:

1. Sees loading indicator
2. Loading indicator disappears
3. Sees list of Pokémon (or error)
4. Interacts with the app

How do we verify that the loading indicator appears before the main UI and then disappears?

AlphaSense

Issue 5

A **test must fail only when** an expectation is not met

A **test must pass** when an expectation is met

```
// const {data: pokemonData, loading, error} = useFetchPokemonsQuery();

const {data: pokemonData, loading, error} = useQuery<{pokemons: Pokemon[]}>(GET_POKEMONS);

function handleTrainingModeChange(checked: boolean) {
}

if (loading) {
  return <Loading message="Loading Pokémon..." />;
}
```

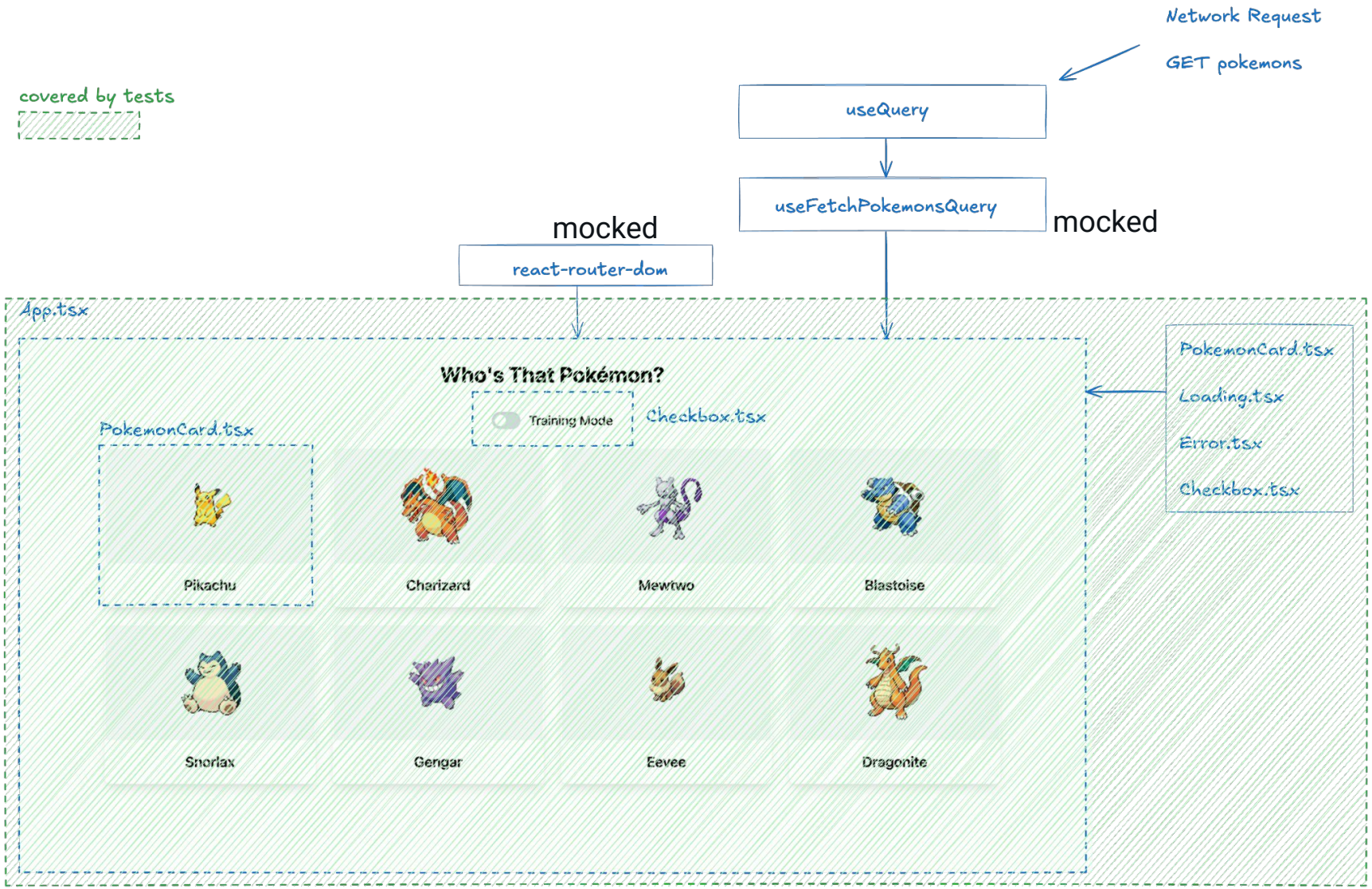
Why do all tests fail when using `useQuery` even though the two lines do the same?

```
// if (loading) {
//   return <Loading message="Loading Pokémon..." />;
// }

if (error) {
  return <ErrorMessage errorMessage={error.message} />;
}

return (
  <div className="container mx-auto px-4 py-8">
    <h1 className="text-3xl font-bold text-center mb-8">Who's That Pokémon?</h1>

    <Loading message="Loading Pokémon..." />

    <div className="flex justify-center mb-6">
      <Checkbox
        label="Training Mode"
```

All tests are passed when the **loading doesn't work** as expected

AlphaSense

Before

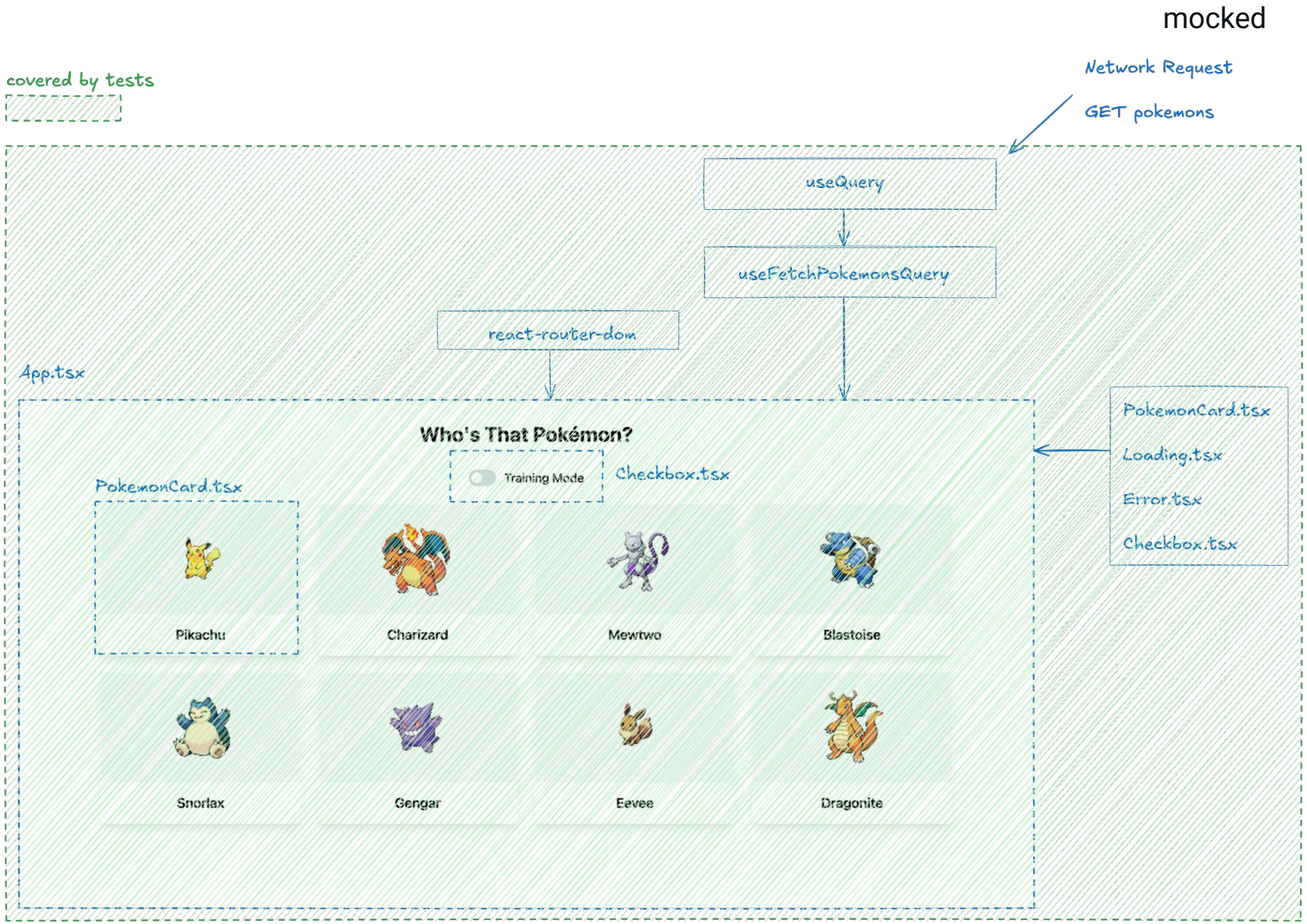


We're mocking
- react-router
- useFetchPokemonQuery

The three states—success, loading, and error—are tested in isolation.

AlphaSense

After: Expanding the boundary



✅ Fixed by **removing mocks** for:

- react-router
- useFetchPokemonQuery

[See changes]

Docs

- [MockedProvider]
- [Testing React Router]

Code coverage

with only 1 test file

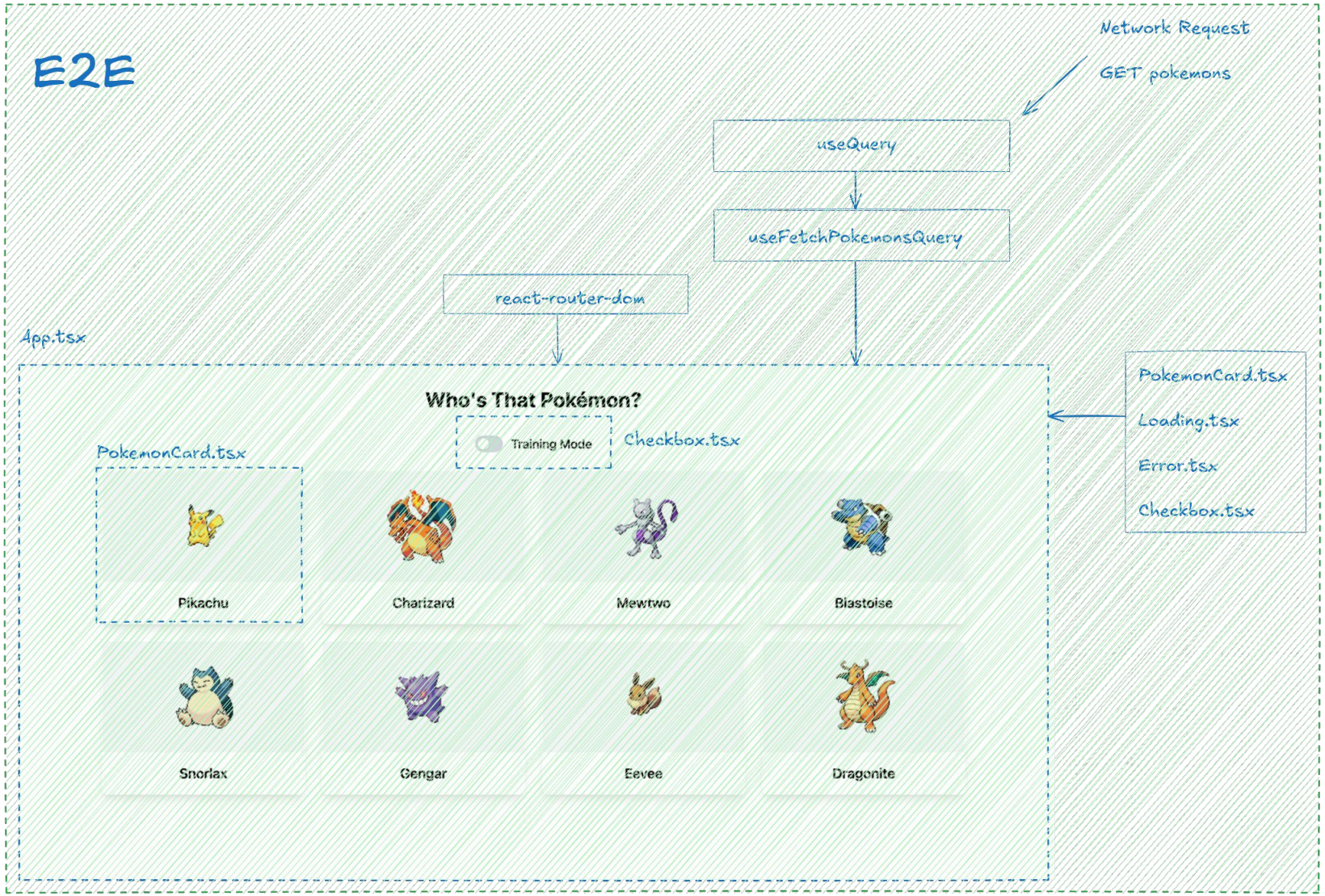
👉 **Before**

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	96.79	94.44	90	96.79	
src	100	100	100	100	
App.tsx	100	100	100	100	
src/components	100	96.42	100	100	
Checkbox.tsx	100	100	100	100	
ErrorMessage.tsx	100	100	100	100	
Loading.tsx	100	100	100	100	
PokemonCard.tsx	100	95	100	100	49
src/graphql	0	0	0	0	
pokemons.graphql.ts	0	0	0	0	1-16

👉 **After**

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	97.22	100	100	
src	100	100	100	100	
App.tsx	100	100	100	100	
src/components	100	96.42	100	100	
Checkbox.tsx	100	100	100	100	
ErrorMessage.tsx	100	100	100	100	
Loading.tsx	100	100	100	100	
PokemonCard.tsx	100	95	100	100	49
src/graphql	100	100	100	100	
pokemons.graphql.ts	100	100	100	100	

E2E


covered by tests
E2E
Network Request
GET pokemons
useQuery
useFetchPokemonsQuery
react-router-dom
App.tsx
PokemonCard.tsx
Loading.tsx
Error.tsx
Checkbox.tsx
Who's That Pokémon?
Training Mode
Checkbox.tsx
PokemonCard.tsx
Pikachu
Charizard
Mewtwo
Blastoise
Snorlax
Gengar
Eevee
Dragonite

Recap...

Thank you!