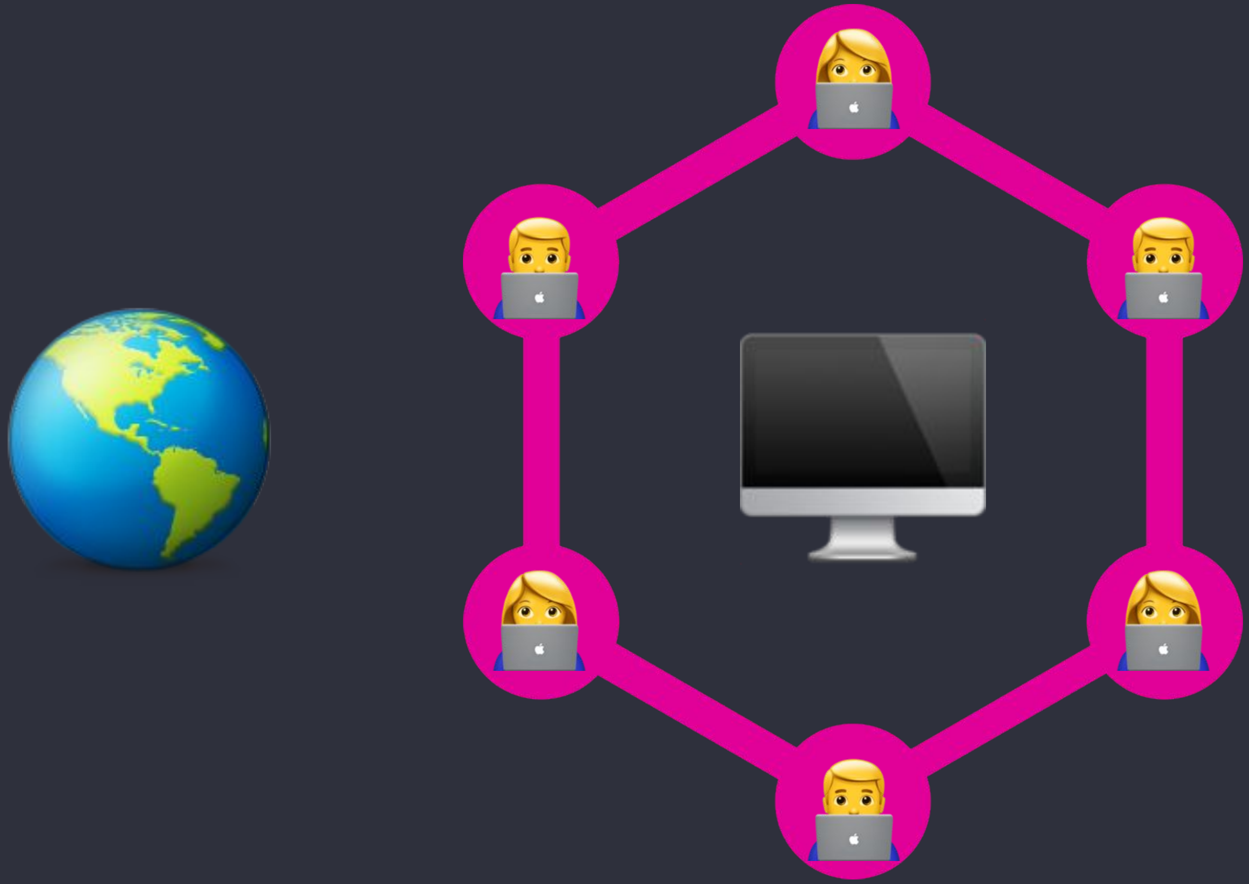
GraphQL on the Edge

A shield for your app

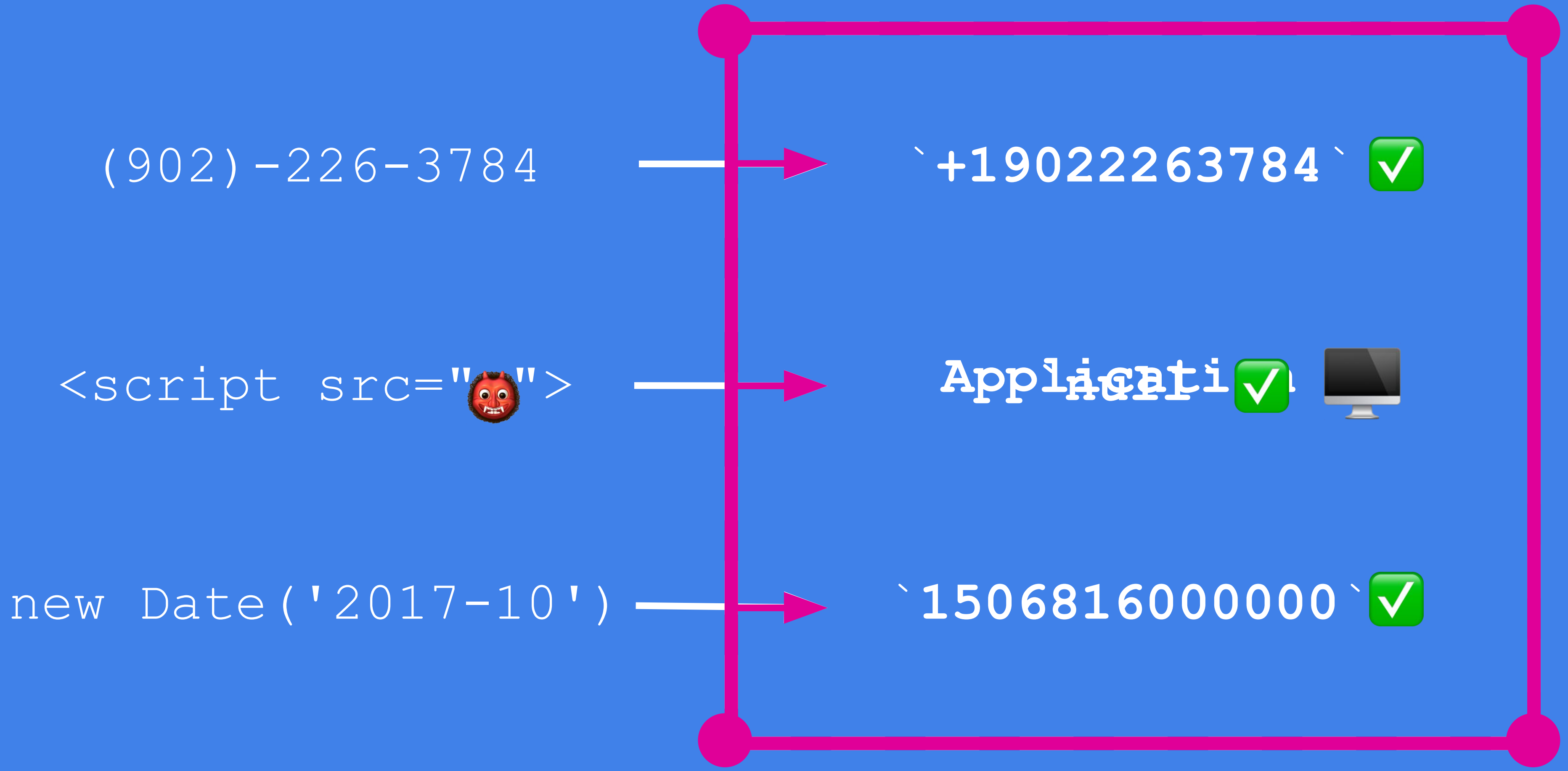
Mike Fix



Normalize

Sanitize

Verify


(902)-226-3784
`+19022263784` ✅
<script src="👹">
Applicati✅n 🖥️
null
new Date('2017-10')
`1506816000000`✅

GraphQL Normalized Types

✅ **Simple**

`npm i -S **gnt**`

✅ **Consistent**

✅ **Reliable**

The Future

☠️ **XSS** Safe Type

📛 Human Name

🌈 ...

The Future

Email	Good Password	Max String
CVV	**CSS Color**	SQL Safe
Cent	HTML Tag	**URL**
Address	Currencies	IP Address
Country	**MIME Type**	Glob String
Blood Type	File	**Airport**

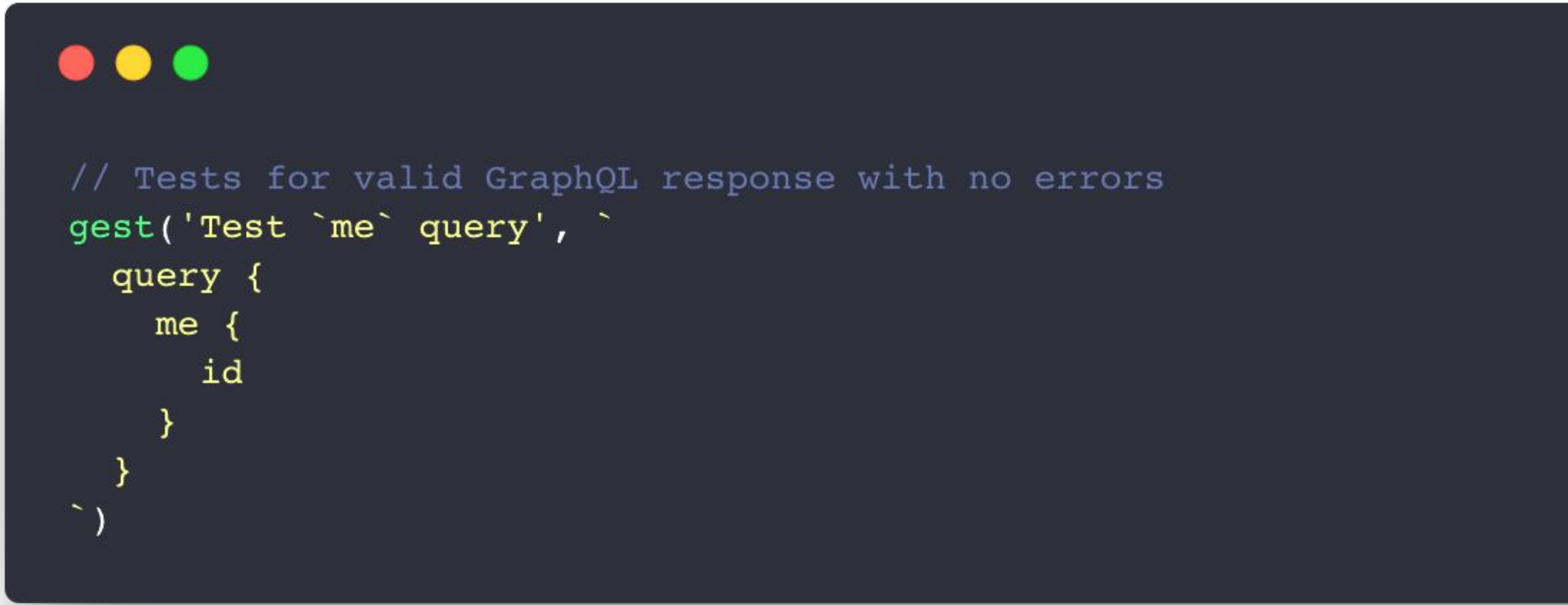
Testing at the Edge

"For libraries, **tests should ideally be written against the public APIs.** This goes against the common mantra of unit testing, but in our experience this helps both ensure that the right thing is being tested, and that it is easy to replace the underlying implementation"

- Dan Abramov

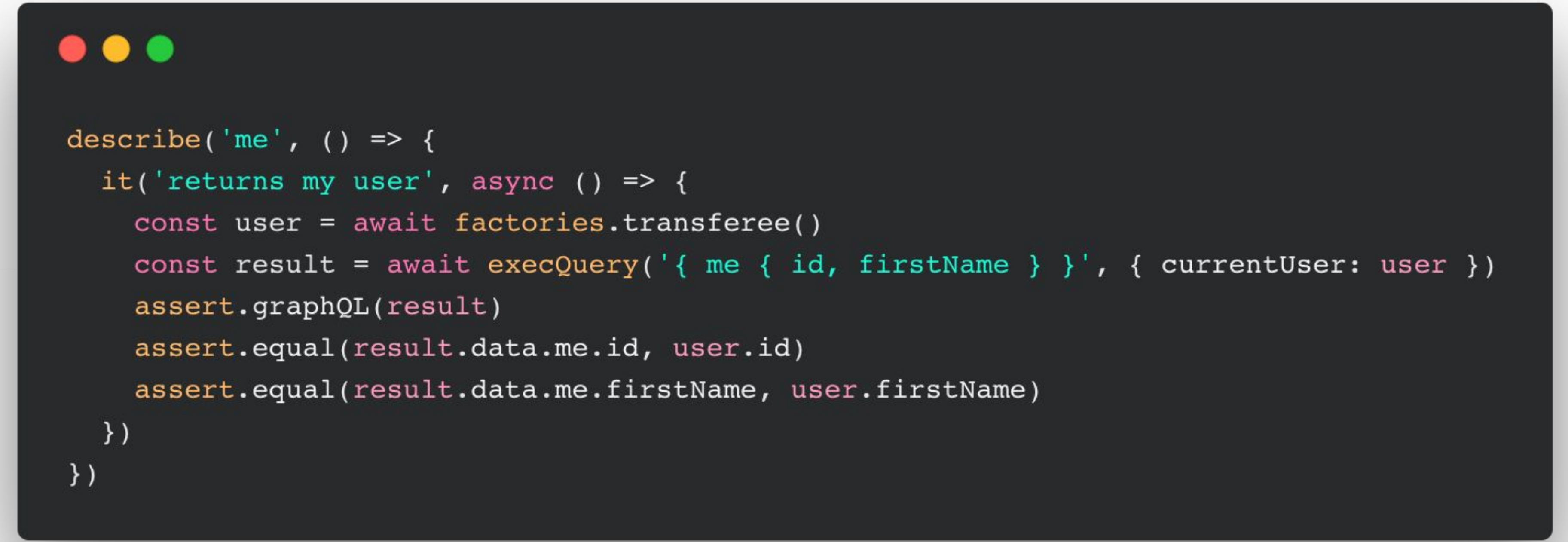
Source: https://hashnode.com/post/what-have-you-learned...

API Testing



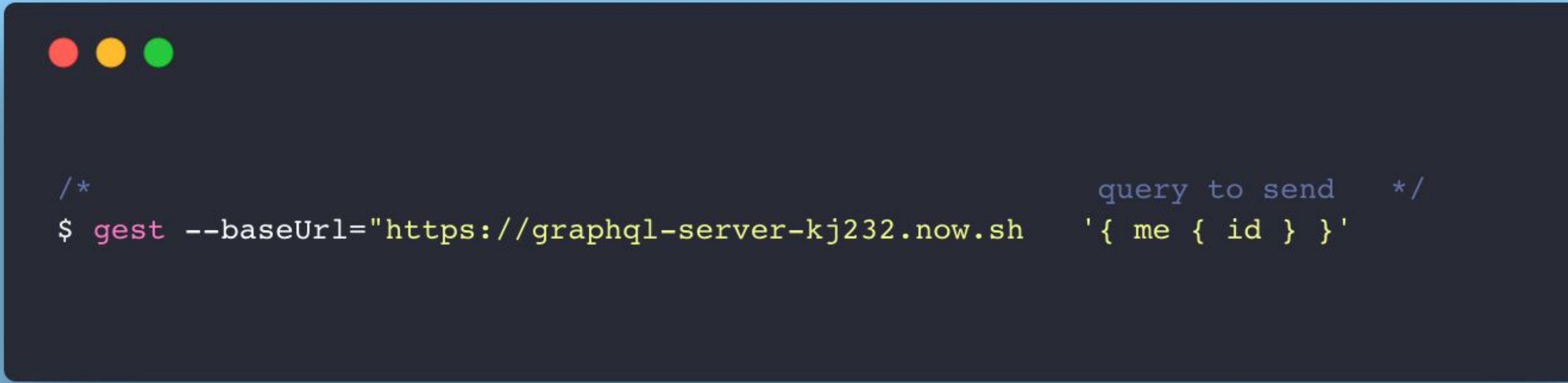
```
// Tests for valid GraphQL response with no errors
gest('Test `me` query', `
  query {
    me {
      id
    }
  }
`)
```

Unit Testing!



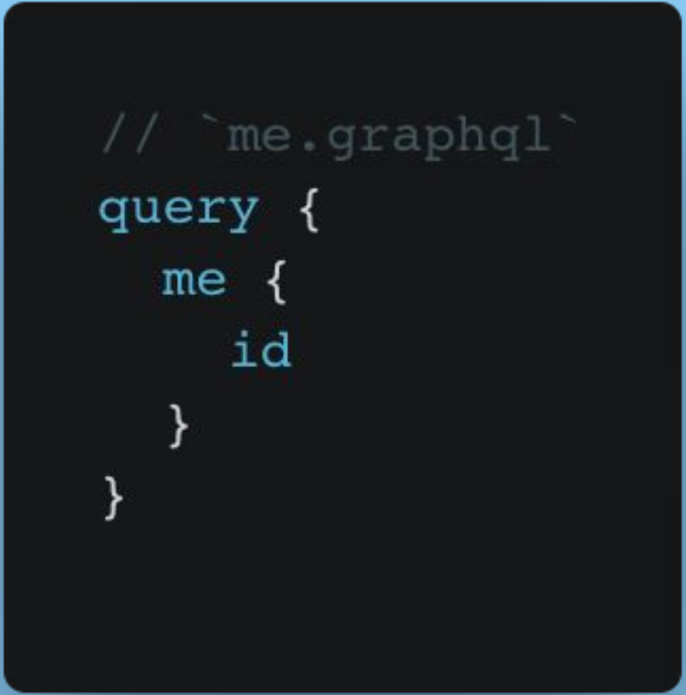
```
describe('me', () => {
  it('returns my user', async () => {
    const user = await factories.transferee()
    const result = await execQuery('{ me { id, firstName } }', { currentUser: user })
    assert.graphQL(result)
    assert.equal(result.data.me.id, user.id)
    assert.equal(result.data.me.firstName, user.firstName)
  })
})
```

Deployment, Smoke, Regression...

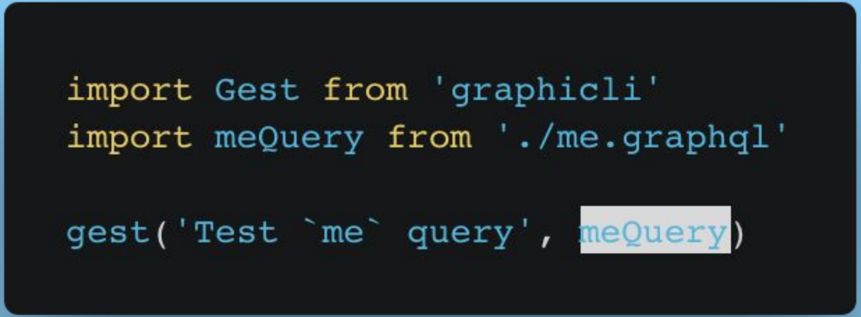


```
/*                                                    query to send   */
$ gest --baseUrl="https://graphql-server-kj232.now.sh   '{ me { id } }'
```

Reusability

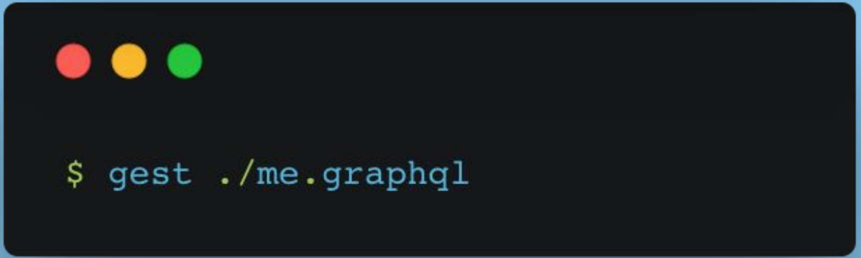


```graphql
// `me.graphql`
query {
  me {
    id
  }
}
```



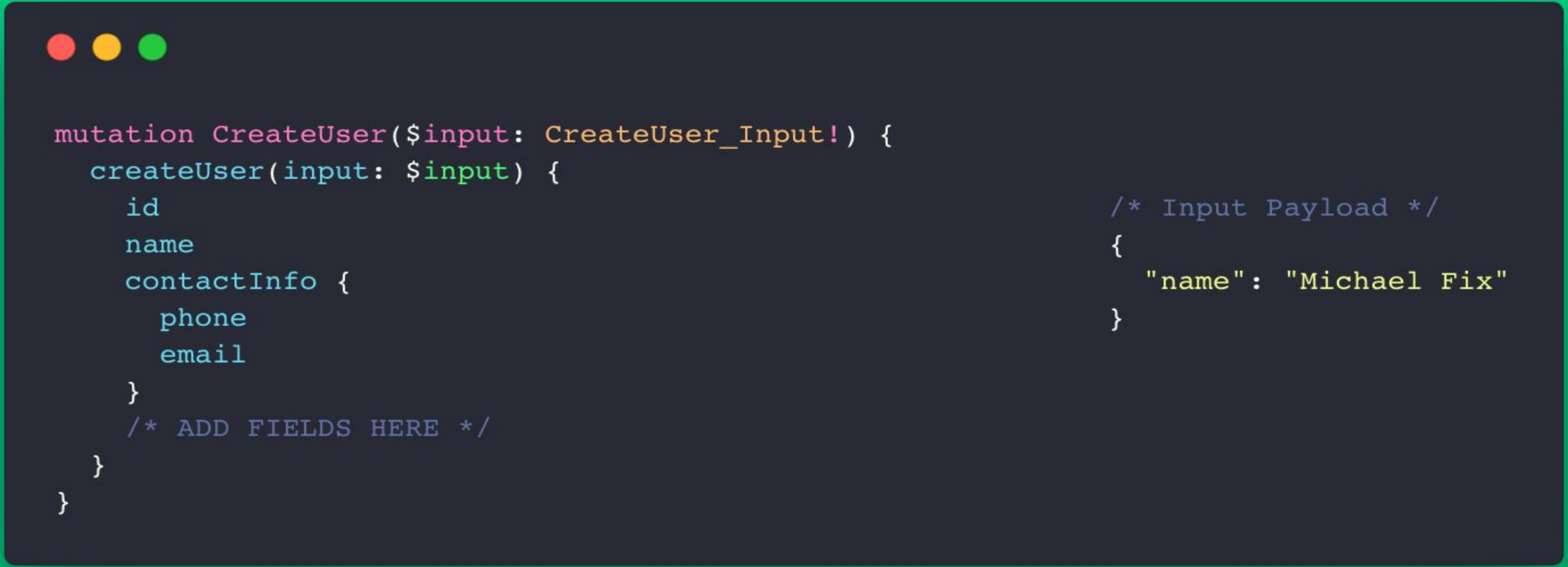
```js
import Gest from 'graphicli'
import meQuery from './me.graphql'

gest('Test `me` query', meQuery)
```



```
$ gest ./me.graphql
```

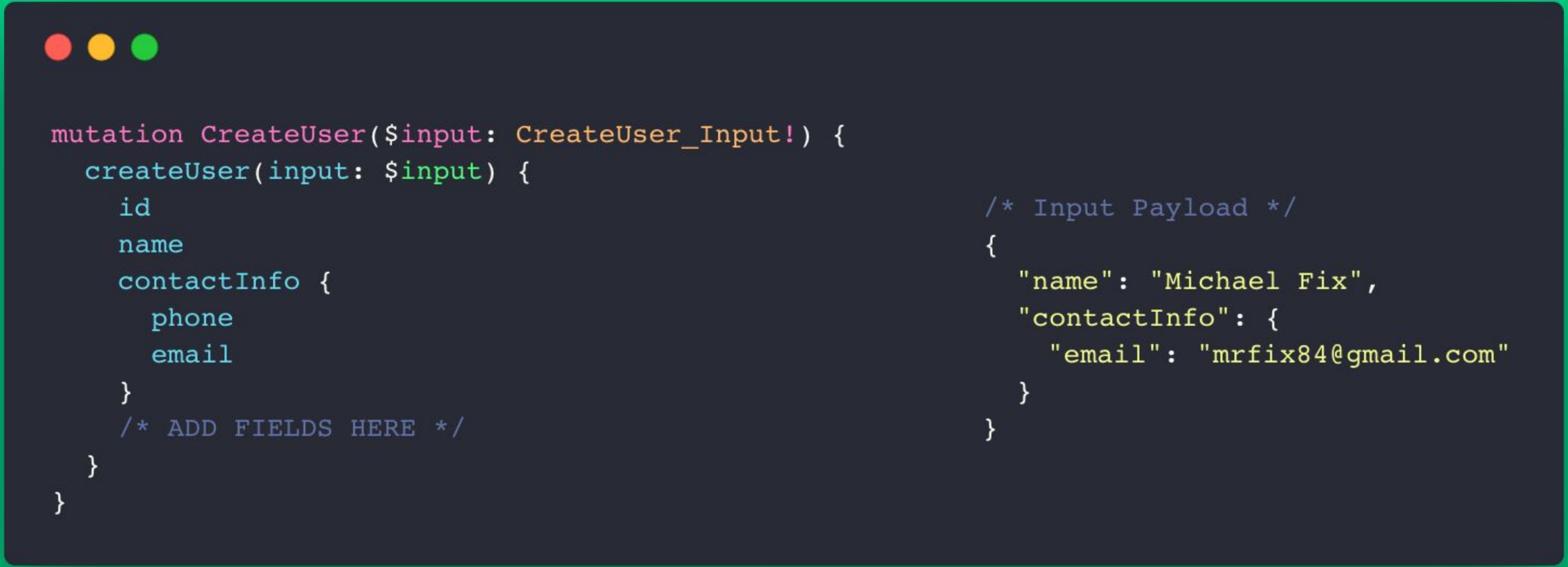
Data-driven Testing



```
mutation CreateUser($input: CreateUser_Input!) {
  createUser(input: $input) {
    id
    name
    contactInfo {
      phone
      email
    }
    /* ADD FIELDS HERE */
  }
}
```

```
/* Input Payload */
{
  "name": "Michael Fix"
}
```

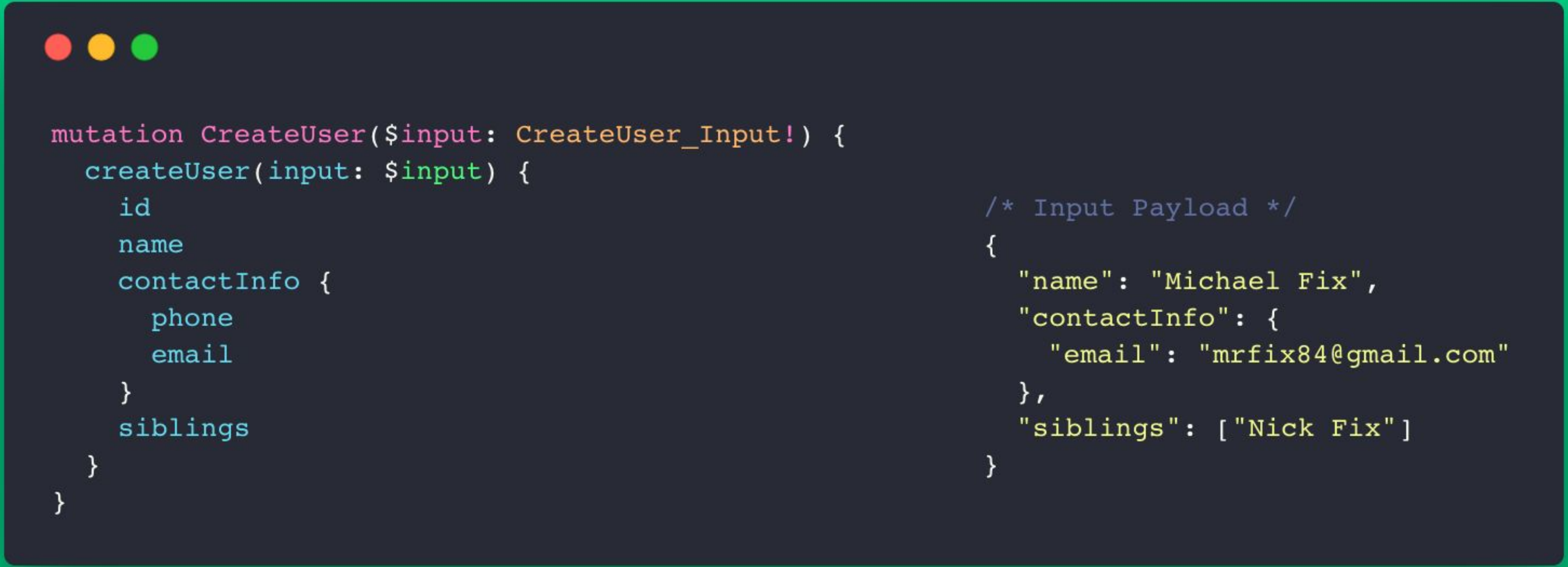
...only your data changes



```graphql
mutation CreateUser($input: CreateUser_Input!) {
  createUser(input: $input) {
    id
    name
    contactInfo {
      phone
      email
    }
    /* ADD FIELDS HERE */
  }
}
```

```
/* Input Payload */
{
  "name": "Michael Fix",
  "contactInfo": {
    "email": "mrfix84@gmail.com"
  }
}
```

...only your data changes



```
mutation CreateUser($input: CreateUser_Input!) {
  createUser(input: $input) {
    id
    name
    contactInfo {
      phone
      email
    }
    siblings
  }
}
```

```
/* Input Payload */
{
  "name": "Michael Fix",
  "contactInfo": {
    "email": "mrfix84@gmail.com"
  },
  "siblings": ["Nick Fix"]
}
```

"For very complex code, fuzz testing may work better than unit tests. If the code you're **testing has to handle many unexpected cases** that are hard to predict, it might be worth writing tests in a non-deterministic manner that **randomly generates inputs** and asserts that the outputs satisfy certain conditions."

- Dan Abramov

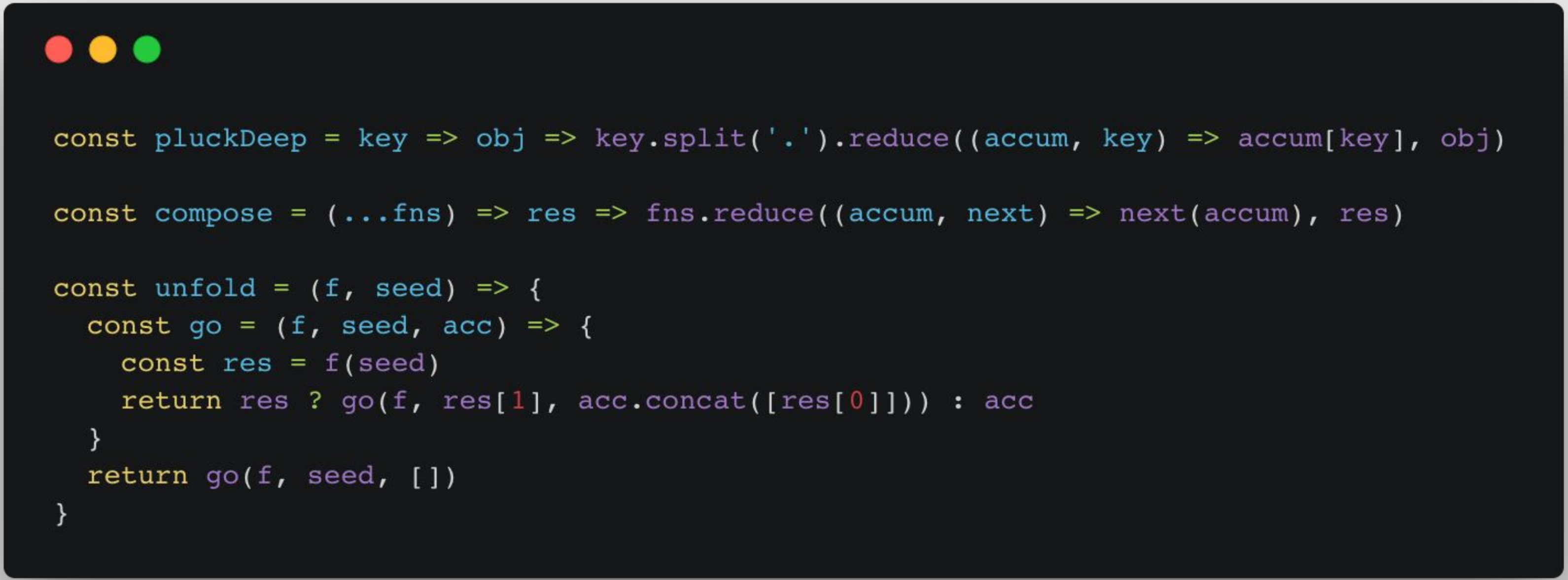
Source: https://hashnode.com/post/what-have-you-learned...

Contact Me 😄

```
{
  data: {
    me: {
      displayName: "Michael Fix",
      email: "mrfix84@gmail.com",
      website: "mfix22.github.io",
      socials: {
        twitter: "@fixitup2",
        github: "@mfix22"
      }
    }
  }
}
```

Bonus

carbon.now.sh



```
const pluckDeep = key => obj => key.split('.').reduce((accum, key) => accum[key], obj)

const compose = (...fns) => res => fns.reduce((accum, next) => next(accum), res)

const unfold = (f, seed) => {
  const go = (f, seed, acc) => {
    const res = f(seed)
    return res ? go(f, res[1], acc.concat([res[0]])) : acc
  }
  return go(f, seed, [])
}
```

"Write tests. Not too many. **Mostly integration.**"

- Guillermo Rauch

Source: https://twitter.com/rauchg/status/807635649813618688