Deterministic Mocking with GraphQL and Apollo: Lunar Launch

ez cater

Goals

By the time this presentation is over, you should:

- Understand what GraphQL & Apollo are
- Be familiar with tooling to jumpstart your React/Apollo project
- Be able to write deterministic Unit Tests with Apollo components
- Explain what problem Lunar solves

Goals

By the time this presentation is over, you should:

- **Understand what GraphQL & Apollo are**
- Be familiar with tooling to jumpstart your React/Apollo project
- Be able to write deterministic Unit Tests with Apollo components
- Explain what problems Lunar solves

Goals

By the time this presentation is over, you should:

- Understand what GraphQL & Apollo are
- **Be familiar with tooling to jumpstart your React/Apollo project**
- Be able to write deterministic Unit Tests with Apollo components
- Explain what problems Lunar solves

Goals

By the time this presentation is over, you should:

- Understand what GraphQL & Apollo are
- Be familiar with tooling to jumpstart your React/Apollo project
- **Be able to write deterministic Unit Tests with Apollo components**
- Explain what problems Lunar solves

Goals

By the time this presentation is over, you should:

- Understand what GraphQL & Apollo are
- Be familiar with tooling to jumpstart your React/Apollo project
- Be able to write deterministic Unit Tests with Apollo components
- **Explain what problems Lunar solves**

What are GraphQL & Apollo?

What is GraphQL?

A query language for your API

```
query {
  user(id: 35) {
    id
    name
    eyeColor
  }
}
```

What is GraphQL?

A query language for your API

```
query {
  user(id: 35) {
    id
    name
    eyeColor
  }
}
```

```
{
  "data": {
    "user": {
      "id": 35,
      "name": "Mark Faga"
      "eyeColor": "BROWN"
    }
  }
}
```

What is GraphQL?

Mutation of data is supported as well

```
mutation AddUser($name: String!, $eyeColor: String) {
  addUser(name: $name, eyeColor: $eyeColor) {
    id
    name
    eyeColor
  }
}
```

What is GraphQL?

Mutation of data is supported as well

```
mutation AddUser($name: String!, $eyeColor: String) {
  addUser(name: $name, eyeColor: $eyeColor) {
    id
    name
    eyeColor
  }
}
```

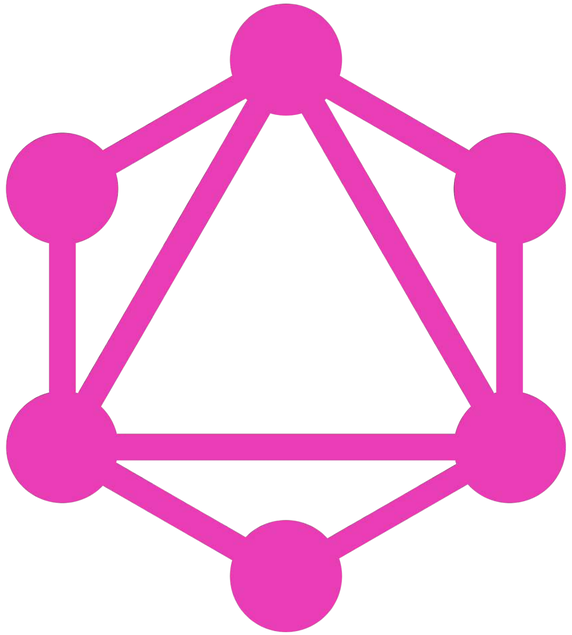
What is GraphQL?

Mutation of data is supported as well

```
mutation AddUser($name: String!, $eyeColor: String) {
  addUser(name: $name, eyeColor: $eyeColor) {
    id
    name
    eyeColor
  }
}
```

What is GraphQL?

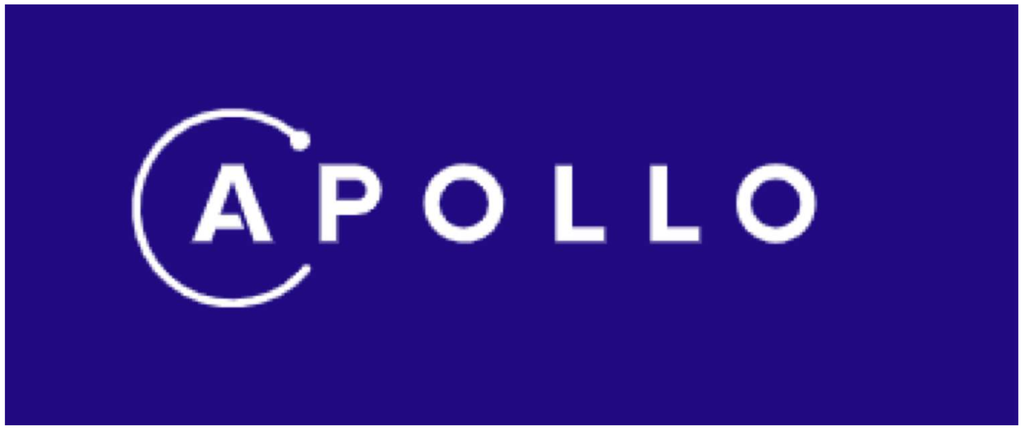
So much more!



https://graphql.org/

What is Apollo?

A family of technologies that allow you to integrate with, monitor and build out applications powered by GraphQL.



https://www.apollographql.com/

Tooling that helps jumpstart your React/GraphQL project:

Apollo Server

Skip Backup Slides

Our Schema

```
type Query {
  users: [User]
  user(id: ID!): User
}

type Mutation {
  addFavoriteFood(
    userId: ID!
    name: String!
    eatingFrequency: EatingFrequency!
  ): FavoriteFoodItem
}

type User {
  id: ID!
  name: String!
  favoriteFoods: [FavoriteFoodItem]
}
```

```
type FavoriteFoodItem {
  id: ID!
  foodItem: FoodItem!
  user: User!
  eatingFrequency: EatingFrequency!
}

type FoodItem {
  id: ID!
  name: String!
}

enum EatingFrequency {
  DAILY
  WEEKLY
  MONTHLY
}
```

Our Schema

```
type Query {
  users: [User]
  user(id: ID!): User
}

type Mutation {
  addFavoriteFood(
    userId: ID!
    name: String!
    eatingFrequency: EatingFrequency!
  ): FavoriteFoodItem
}

type User {
  id: ID!
  name: String!
  favoriteFoods: [FavoriteFoodItem]
}
```

```
type FavoriteFoodItem {
  id: ID!
  foodItem: FoodItem!
  user: User!
  eatingFrequency: EatingFrequency!
}

type FoodItem {
  id: ID!
  name: String!
}

enum EatingFrequency {
  DAILY
  WEEKLY
  MONTHLY
}
```

Our Schema

```
type Query {
  users: [User]
  user(id: ID!): User
}

type Mutation {
  addFavoriteFood(
    userID: ID!
    name: String!
    eatingFrequency: EatingFrequency!
  ): FavoriteFoodItem
}

type User {
  id: ID!
  name: String!
  favoriteFoods: [FavoriteFoodItem]
}
```

```
query {
  users {
    id
    name
  }
}
```

Our Schema

```
type Query {
  users: [User]
  user(id: ID!): User
}

type Mutation {
  addFavoriteFood(
    userId: ID!
    name: String!
    eatingFrequency: EatingFrequency!
  ): FavoriteFoodItem
}

type User {
  id: ID!
  name: String!
  favoriteFoods: [FavoriteFoodItem]
}
```

```
query {
  users {
    id
    name
  }
}
```

Our Schema

```
type User {
  id: ID!
  name: String!
  favoriteFoods: [FavoriteFoodItem]
}

type FavoriteFoodItem {
  id: ID!
  foodItem: FoodItem!
  user: User!
  eatingFrequency: EatingFrequency!
}

type FoodItem {
  id: ID!
  name: String!
}
```

```
query {
  users {
    id
    name
    favoriteFoods {
      id
      foodItem {
        id
        name
      }
    }
  }
}
```

Our Schema

```
query {
  users {
    id
    name
    favoriteFoods {
      id
      foodItem {
        id
        name
      }
    }
  }
}
```

Our Schema

```
query {
  users {
    id
    name
    favoriteFoods {
      id
      foodItem {
        id
        name
      }
    }
  }
}
```

```
"data": {
  "users": [
    {
      "id": "170a55ad-5436-4aee-9aaa-c76c5a0b1c58",
      "name": "Margret Reichert",
      "favoriteFoods": [
        {
          "id": "2a4a406c-8af5-40ee-a82b-25ee6d4897e0",
          "foodItem": {
            "id": "844e0d30-6e53-4231-bd56-33daf9e2fdf2",
            "name": "Hummus"
          }
        },
        ...
      ]
    },
    ...
  ]
}
```

Our Schema

```
query {
  users {
    id
    name
    favoriteFoods {
      id
      foodItem {
        id
        name
      }
    }
  }
}
```

```
"data": {
  "users": [
    {
      "id": "170a55ad-5436-4aee-9aaa-c76c5a0b1c58",
      "name": "Margret Reichert",
      "favoriteFoods": [
        {
          "id": "2a4a406c-8af5-40ee-a82b-25ee6d4897e0",
          "foodItem": {
            "id": "844e0d30-6e53-4231-bd56-33daf9e2fdf2",
            "name": "Hummus"
          }
        },
        ...
      ]
    },
    ...
  ]
}
```

Our Schema

```
query {
  users {
    id
    name
    favoriteFoods {
      id
      foodItem {
        id
        name
      }
    }
  }
}
```

```
"data": {
  "users": [
    {
      "id": "170a55ad-5436-4aee-9aaa-c76c5a0b1c58",
      "name": "Margret Reichert",
      "favoriteFoods": [
        {
          "id": "2a4a406c-8af5-40ee-a82b-25ee6d4897e0",
          "foodItem": {
            "id": "844e0d30-6e53-4231-bd56-33daf9e2fdf2",
            "name": "Hummus"
          }
        },
        ...
      ]
    },
    ...
  ]
}
```

My First Apollo Server

```
const {ApolloServer} = require('apollo-server');
const requireText = require('require-text');

const typeDefs = requireText('./schema.graphql', require);

const server = new ApolloServer({
  typeDefs,
  mocks: true,
});

server.listen({port: 3001}).then(({url}) => {
  // eslint-disable-next-line no-console
  console.log(`🚀 Server ready at ${url}`);
});
```

My First Apollo Server

```
const {ApolloServer} = require('apollo-server');
const requireText = require('require-text');

const typeDefs = requireText('./schema.graphql', require);

const server = new ApolloServer({
  typeDefs,
  mocks: true,
});

server.listen({port: 3001}).then(({url}) => {
  // eslint-disable-next-line no-console
  console.log(`🚀 Server ready at ${url}`);
});
```

My First Apollo Server

```
const {ApolloServer} = require('apollo-server');
const requireText = require('require-text');

const typeDefs = requireText('./schema.graphql', require);

const server = new ApolloServer({
  typeDefs,
  mocks: true,
});

server.listen({port: 3001}).then(({url}) => {
  // eslint-disable-next-line no-console
  console.log(`🚀 Server ready at ${url}`);
});
```

My First Apollo Server

```
const {ApolloServer} = require('apollo-server');
const requireText = require('require-text');

const typeDefs = requireText('./schema.graphql', require);

const server = new ApolloServer({
  typeDefs,
  mocks: true,
});

server.listen({port: 3001}).then(({url}) => {
  // eslint-disable-next-line no-console
  console.log(`🚀 Server ready at ${url}`);
});
```

mocks: true

```
const {ApolloServer} = require('apollo-server');
const requireText = require('require-text');

const typeDefs = requireText('./schema.graphql', require);

const server = new ApolloServer({
  typeDefs,
  mocks: true,
});

server.listen({port: 3001}).then(({url}) => {
  // eslint-disable-next-line no-console
  console.log(`🚀 Server ready at ${url}`);
});
```

My First Apollo Server

```
const {ApolloServer} = require('apollo-server');
const requireText = require('require-text');

const typeDefs = requireText('./schema.graphql', require);

const server = new ApolloServer({
  typeDefs,
  mocks: true,
});

server.listen({port: 3001}).then(({url}) => {
  // eslint-disable-next-line no-console
  console.log(`🚀 Server ready at ${url}`);
});
```

mocks: true


```
{
  "data": {
    "users": [
      {
        "id": "2b59b261-5c3a-4f7b-86c4-2270a4255749",
        "name": "Hello World",
        "favoriteFoods": [
          {
            "id": "f345504e-f8ed-42ed-a244-0485d8284226",
            "foodItem": {
              "id": "1cc35ecb-ed9e-4444-8726-2d38826a305f",
              "name": "Hello World"
            },
            "eatingFrequency": "DAILY"
          },
          {
            "id": "9a22ec1b-c6ca-47f3-afae-94df51f5c921",
            "foodItem": {
              "id": "7b3dd6a4-10a2-464b-b067-a85b4b240c19",
              "name": "Hello World"
            },
            "eatingFrequency": "MONTHLY"
          }
        ]
      },
      ...
    ]
  }
}
```

mocks: true


```
{
  "data": {
    "users": [
      {
        "id": "2b59b261-5c3a-4f7b-86c4-2270a4255749",
        "name": "Hello World",
        "favoriteFoods": [
          {
            "id": "f345504e-f8ed-42ed-a244-0485d8284226",
            "foodItem": {
              "id": "1cc35ecb-ed9e-4444-8726-2d38826a305f",
              "name": "Hello World"
            },
            "eatingFrequency": "DAILY"
          },
          {
            "id": "9a22ec1b-c6ca-47f3-afae-94df51f5c921",
            "foodItem": {
              "id": "7b3dd6a4-10a2-464b-b067-a85b4b240c19",
              "name": "Hello World"
            },
            "eatingFrequency": "MONTHLY"
          }
        ]
      },
      ...
    ]
  }
}
```

mocks: true

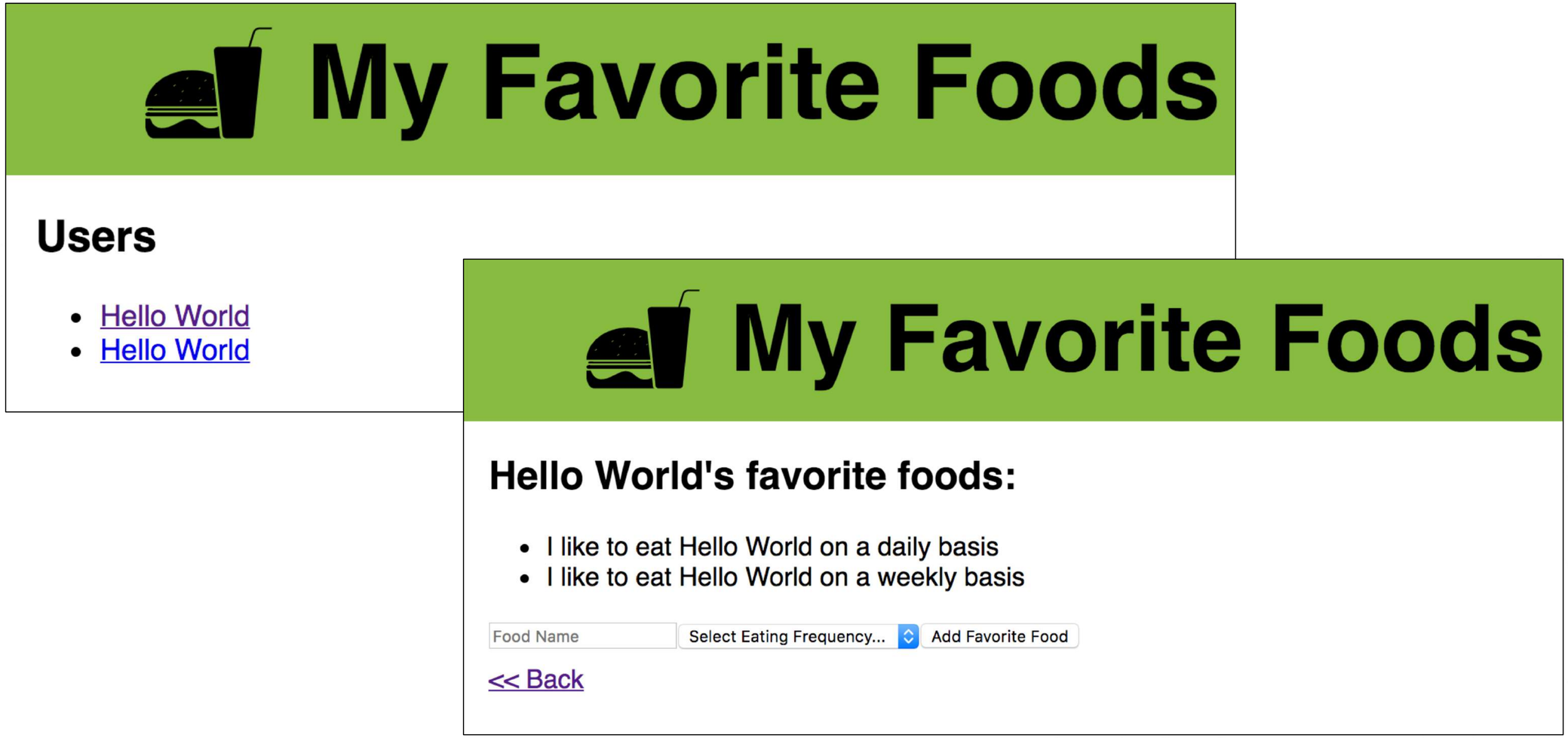

```
{
  "data": {
    "users": [
      {
        "id": "2b59b261-5c3a-4f7b-86c4-2270a4255749",
        "name": "Hello World",
        "favoriteFoods": [
          {
            "id": "f345504e-f8ed-42ed-a244-0485d8284226",
            "foodItem": {
              "id": "1cc35ecb-ed9e-4444-8726-2d38826a305f",
              "name": "Hello World"
            },
            "eatingFrequency": "DAILY"
          },
          {
            "id": "9a22ec1b-c6ca-47f3-afae-94df51f5c921",
            "foodItem": {
              "id": "7b3dd6a4-10a2-464b-b067-a85b4b240c19",
              "name": "Hello World"
            },
            "eatingFrequency": "MONTHLY"
          }
        ]
      },
      ...
    ]
  }
}
```

mocks: true

```
{
  "data": {
    "users": [
      {
        "id": "2b59b261-5c3a-4f7b-86c4-2270a4255749",
        "name": "Hello World",
        "favoriteFoods": [
          {
            "id": "f345504e-f8ed-42ed-a244-0485d8284226",
            "foodItem": {
              "id": "1cc35ecb-ed9e-4444-8726-2d38826a305f",
              "name": "Hello World"
            },
            "eatingFrequency": "DAILY"
          },
          {
            "id": "9a22ec1b-c6ca-47f3-afae-94df51f5c921",
            "foodItem": {
              "id": "7b3dd6a4-10a2-464b-b067-a85b4b240c19",
              "name": "Hello World"
            },
            "eatingFrequency": "MONTHLY"
          }
        ]
      },
      ...
    ]
  }
}
```

```
enum EatingFrequency {
  DAILY
  WEEKLY
  MONTHLY
}
```

mocks: true


My Favorite Foods
Users
Hello World
Hello World
My Favorite Foods
Hello World's favorite foods:
I like to eat Hello World on a daily basis
I like to eat Hello World on a weekly basis
Food Name
Select Eating Frequency...
Add Favorite Food
<< Back

My Second Apollo Server

```
const {ApolloServer} = require('apollo-server');
const requireText = require('require-text');
const mocks = require('./mocks');

const typeDefs = requireText('./schema.graphql', require);

const server = new ApolloServer({
  typeDefs,
  mocks,
});

server.listen({port: 3001}).then(({url}) => {
  // eslint-disable-next-line no-console
  console.log(`🚀 Server ready at ${url}`);
});
```

mocks: { … }

```
const {MockList} = require('apollo-server');
const faker = require('faker');

const mocks = {
  FoodItem: () => ({
    name: faker.helpers.randomize([
      'Chocolate Ice Cream',
      'Peppers',
      'Hummus',
      'Sushi',
      'Eggs Benedict',
      'Pad Se Ew',
    ]),
  }),
  User: () => ({
    name: faker.name.findName(),
    favoriteFoods: () => new MockList(5),
  }),
  Query: () => ({
    users: () => new MockList(3),
  }),
};

module.exports = mocks;
```

mocks: { … }

```
const {MockList} = require('apollo-server');
const faker = require('faker');

const mocks = {
  FoodItem: () => ({
    name: faker.helpers.randomize([
      'Chocolate Ice Cream',
      'Peppers',
      'Hummus',
      'Sushi',
      'Eggs Benedict',
      'Pad Se Ew',
    ]),
  }),
  User: () => ({
    name: faker.name.findName(),
    favoriteFoods: () => new MockList(5),
  }),
  Query: () => ({
    users: () => new MockList(3),
  }),
};

module.exports = mocks;
```

mocks: { … }

```
const {MockList} = require('apollo-server');
const faker = require('faker');

const mocks = {
  FoodItem: () => ({
    name: faker.helpers.randomize([
      'Chocolate Ice Cream',
      'Peppers',
      'Hummus',
      'Sushi',
      'Eggs Benedict',
      'Pad Se Ew',
    ]),
  }),
  User: () => ({
    name: faker.name.findName(),
    favoriteFoods: () => new MockList(5),
  }),
  Query: () => ({
    users: () => new MockList(3),
  }),
};

module.exports = mocks;
```

mocks: { … }

```
const {MockList} = require('apollo-server');
const faker = require('faker');

const mocks = {
  FoodItem: () => ({
    name: faker.helpers.randomize([
      'Chocolate Ice Cream',
      'Peppers',
      'Hummus',
      'Sushi',
      'Eggs Benedict',
      'Pad Se Ew',
    ]),
  }),
  User: () => ({
    name: faker.name.findName(),
    favoriteFoods: () => new MockList(5),
  }),
  Query: () => ({
    users: () => new MockList(3),
  }),
};

module.exports = mocks;
```

mocks: { … }

```
const {MockList} = require('apollo-server');
const faker = require('faker');

const mocks = {
  FoodItem: () => ({
    name: faker.helpers.randomize([
      'Chocolate Ice Cream',
      'Peppers',
      'Hummus',
      'Sushi',
      'Eggs Benedict',
      'Pad Se Ew',
    ]),
  }),
  User: () => ({
    name: faker.name.findName(),
    favoriteFoods: () => new MockList(5),
  }),
  Query: () => ({
    users: () => new MockList(3),
  }),
};

module.exports = mocks;
```

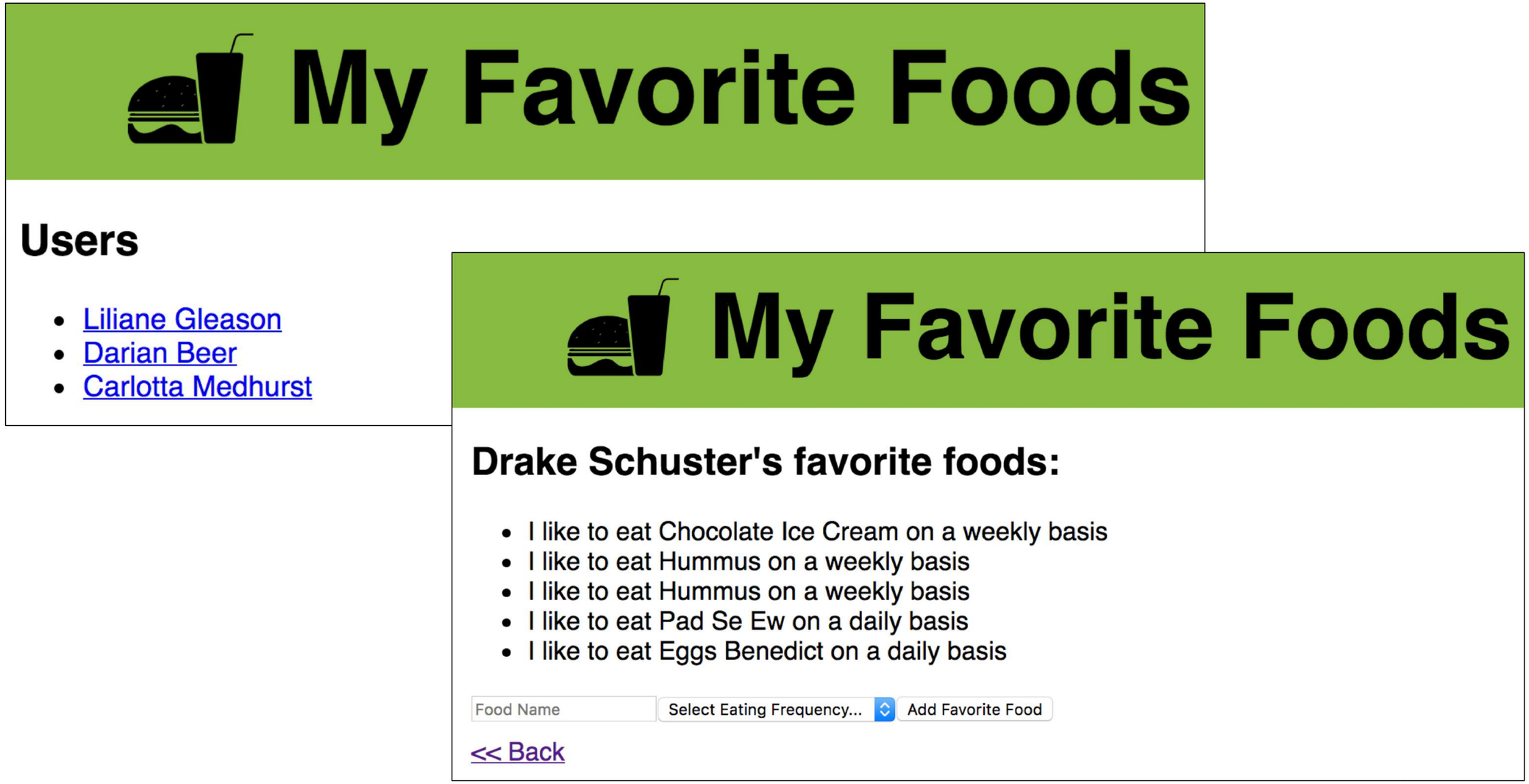
mocks: { … }

```
const {MockList} = require('apollo-server');
const faker = require('faker');

const mocks = {
  FoodItem: () => ({
    name: faker.helpers.randomize([
      'Chocolate Ice Cream',
      'Peppers',
      'Hummus',
      'Sushi',
      'Eggs Benedict',
      'Pad Se Ew',
    ]),
  }),
  User: () => ({
    name: faker.name.findName(),
    favoriteFoods: () => new MockList(5),
  }),
  Query: () => ({
    users: () => new MockList(3),
  }),
};

module.exports = mocks;
```

mocks: { … }

```
{
  "data": {
    "users": [
      {
        "id": "0afaa7c7-e109-4157-b1a8-d942e7c08731",
        "name": "Carroll Kerluke",
        "favoriteFoods": [
          {
            "id": "b3835714-340a-431f-bafe-7e1da7b2a430",
            "foodItem": {
              "id": "20b3e429-3b7b-41a6-bf1c-7d9009969466",
              "name": "Pad Se Ew"
            },
            "eatingFrequency": "DAILY"
          },
          ...
        ]
      },
      ...
    ]
  }
}
```

mocks: { … }


My Favorite Foods
Users
Liliane Gleason
Darian Beer
Carlotta Medhurst
My Favorite Foods
Drake Schuster's favorite foods:
I like to eat Chocolate Ice Cream on a weekly basis
I like to eat Hummus on a weekly basis
I like to eat Hummus on a weekly basis
I like to eat Pad Se Ew on a daily basis
I like to eat Eggs Benedict on a daily basis
Food Name
Select Eating Frequency...
Add Favorite Food
<< Back

How to write deterministic unit tests with GraphQL

A React Component that uses <Query />

```
const UserList = () => (
  <Query query={USER_LIST_QUERY}>
    {({data, error, loading}) => {
      if (loading) return <div>Loading...</div>;
      if (error && error.networkError) return <div>{error.message}</div>;
      if (error && error.graphQLErrors)
        return <div>{`GraphQL error: ${error.graphQLErrors[0].message}`}</div>;
      if (data.users.length === 0)
        return <div>You don&#39;t have any users yet!</div>;

      return (
        <ul className="user-list">
          {data.users.map((user, index) => (
            <UserListItem user={user} key={index} />
          ))}
        </ul>
      );
    }}
  </Query>
);
UserList.query = USER_LIST_QUERY;
```

A React Component that uses <Query />

```
const UserList = () => (
  <Query query={USER_LIST_QUERY}>
    {({data, error, loading}) => {
      if (loading) return <div>Loading...</div>;
      if (error && error.networkError) return <div>{error.message}</div>;
      if (error && error.graphQLErrors)
        return <div>{`GraphQL error: ${error.graphQLErrors[0].message}`}</div>;
      if (data.users.length === 0)
        return <div>You don&#39;t have any users yet!</div>;

      return (
        <ul className="user-list">
          {data.users.map((user, index) => (
            <UserListItem user={user} key={index} />
          ))}
        </ul>
      );
    }}
  </Query>
);
UserList.query = USER_LIST_QUERY;
```

A React Component that uses <Query />

```
const UserList = () => (
  <Query query={USER_LIST_QUERY}>
    {({data, error, loading}) => {
      if (loading) return <div>Loading...</div>;
      if (error && error.networkError) return <div>{error.message}</div>;
      if (error && error.graphQLErrors)
        return <div>{`GraphQL error: ${error.graphQLErrors[0].message}`}</div>;
      if (data.users.length === 0)
        return <div>You don&#39;t have any users yet!</div>;

      return (
        <ul className="user-list">
          {data.users.map((user, index) => (
            <UserListItem user={user} key={index} />
          ))}
        </ul>
      );
    }}
  </Query>
);
UserList.query = USER_LIST_QUERY;
```

Testing with <Query />

```
import React from 'react';
import wait from 'waait';
import renderer from 'react-test-renderer';
import {MockedProvider} from 'react-apollo/test-utils';
import {MemoryRouter} from 'react-router-dom';
import UserList from '../UserList';
import UserListItem from '../UserListItem';

const renderComponent = mocks =>
  renderer.create(
    <MemoryRouter>
      <MockedProvider mocks={mocks} addTypename={false}>
        <UserList />
      </MockedProvider>
    </MemoryRouter>
  );
```

Testing with <Query />

```
import React from 'react';
import wait from 'waait';
import renderer from 'react-test-renderer';
import {MockedProvider} from 'react-apollo/test-utils';
import {MemoryRouter} from 'react-router-dom';
import UserList from '../UserList';
import UserListItem from '../UserListItem';

const renderComponent = mocks =>
  renderer.create(
    <MemoryRouter>
      <MockedProvider mocks={mocks} addTypename={false}>
        <UserList />
      </MockedProvider>
    </MemoryRouter>
  );
```

Testing with <Query />

```
import React from 'react';
import wait from 'waait';
import renderer from 'react-test-renderer';
import {MockedProvider} from 'react-apollo/test-utils';
import {MemoryRouter} from 'react-router-dom';
import UserList from '../UserList';
import UserListItem from '../UserListItem';

const renderComponent = mocks =>
  renderer.create(
    <MemoryRouter>
      <MockedProvider mocks={mocks} addTypename={false}>
        <UserList />
      </MockedProvider>
    </MemoryRouter>
  );
```

Testing with <Query />

```
import React from 'react';
import wait from 'waait';
import renderer from 'react-test-renderer';
import {MockedProvider} from 'react-apollo/test-utils';
import {MemoryRouter} from 'react-router-dom';
import UserList from '../UserList';
import UserListItem from '../UserListItem';

const renderComponent = mocks =>
  renderer.create(
    <MemoryRouter>
      <MockedProvider mocks={mocks} addTypename={false}>
        <UserList />
      </MockedProvider>
    </MemoryRouter>
  );
```

Testing with <Query />

```
const UserList = () => (
  <Query query={USER_LIST_QUERY}>
    {({data, error, loading}) => {
      if (loading) return <div>Loading...</div>;
```

Testing with <Query />

```
const UserList = () => (
  <Query query={USER_LIST_QUERY}>
    {({data, error, loading}) => {
      if (loading) return <div>Loading...</div>;
```

```
describe('when the user list is loading', () => {
  it('displays loading indicator', () => {
    const component = renderComponent([{request: {query: UserList.query}}]);

    expect(component.toJSON().children.join('')).toEqual('Loading...');
  });
});
```

Testing with <Query />

```
const UserList = () => (
  <Query query={USER_LIST_QUERY}>
    {({data, error, loading}) => {
      if (loading) return <div>Loading...</div>;
```

```
describe('when the user list is loading', () => {
  it('displays loading indicator', () => {
    const component = renderComponent([{request: {query: UserList.query}}]);

    expect(component.toJSON().children.join('')).toEqual('Loading...');
  });
});
```

Testing with <Query />

```
const UserList = () => (
  <Query query={USER_LIST_QUERY}>
    {({data, error, loading}) => {
      if (loading) return <div>Loading...</div>;
```

```
describe('when the user list is loading', () => {
  it('displays loading indicator', () => {
    const component = renderComponent([{request: {query: UserList.query}}]);

    expect(component.toJSON().children.join('')).toEqual('Loading...');
  });
});
```

Testing with <Query />

```
if (data.users.length === 0)
  return <div>You don&#39;t have any users yet!</div>;
```

```
describe('when no users are returned', () => {
  it('display missing users message', async () => {
    const component = renderComponent([
      {request: {query: UserList.query}, result: {data: {users: []}}},
    ]);

    await wait(0);

    expect(component.toJSON().children.join('')).toEqual(
      "You don't have any users yet!"
    );
  });
});
```

Testing with <Query />

```
if (data.users.length === 0)
  return <div>You don&#39;t have any users yet!</div>;
```

```
describe('when no users are returned', () => {
  it('display missing users message', async () => {
    const component = renderComponent([
      {request: {query: UserList.query}, result: {data: {users: []}}},
    ]);

    await wait(0);

    expect(component.toJSON().children.join('')).toEqual(
      "You don't have any users yet!"
    );
  });
});
```

Testing with <Query />

```
return (
  <ul className="user-list">
    {data.users.map((user, index) => (
      <UserListItem user={user} key={index} />
    ))}
  </ul>
);
```

Testing with <Query />

```
describe('when users are returned', () => {
  it('renders those users in a list', async () => {
    const mocks = [
      {
        request: {
          query: UserList.query,
        },
        result: {
          data: {
            users: [{id: 1, name: 'Mark'}, {id: 2, name: 'Hillary'}],
          },
        },
      },
    ];

    const component = renderComponent(mocks);

    await wait(0);
```

Testing with <Query />

```
const userListItems = component.root.findAllByType(UserListItem);
expect(userListItems).toHaveLength(2);
expect(userListItems[0].props).toEqual({
  user: {
    id: 1,
    name: 'Mark',
  },
});
expect(userListItems[1].props).toEqual({
  user: {
    id: 2,
    name: 'Hillary',
  },
});
```

Testing with <Query />

```
const mocks = [
  {
    request: {
      query: UserList.query,
    },
    result: {
      data: {
        users: [{name: 'Mark'}, {id: 2, name: 'Hillary'}],
      },
    },
  },
];
```

```
console.warn node_modules/apollo-cach
  Missing field id in {
    "name": "Mark"
  }
```

Testing with <Query />

```
describe('there is a network error', () => {
  it('displays the error', async () => {
    const component = renderComponent([
      {
        request: {query: UserList.query},
        error: new Error('I broke'),
      },
    ]);
```

```
describe('there is a GraphQL error', () => {
  it('displays the error', async () => {
    const component = renderComponent([
      {
        request: {query: UserList.query},
        result: {
          errors: [{message: 'Whoops!'}],
        },
      },
    ]);
```

Testing with <Query />

```
describe('there is a network error', () => {
  it('displays the error', async () => {
    const component = renderComponent([
      {
        request: {query: UserList.query},
        error: new Error('I broke'),
      },
    ]);
```

```
describe('there is a GraphQL error', () => {
  it('displays the error', async () => {
    const component = renderComponent([
      {
        request: {query: UserList.query},
        result: {
          errors: [{message: 'Whoops!'}],
        },
      },
    ]);
```

Testing with <Mutation />

```
<Mutation
  mutation={ADD_FAVORITE_FOOD_MUTATION}
  update={updateWrapper(userId)}
>
  {addFavoriteFood => (
    // We render our form here
  )}
</Mutation>
```

Testing with <Mutation />

```
const newFoodName = 'Delicious Food';
const newEatingFrequency = 'DAILY';
const userId = 123;
const mocks = [
  {
    request: {
      query: AddFavoriteFoodToUser.mutation,
      variables: {
        userId,
        name: newFoodName,
        eatingFrequency: newEatingFrequency,
      },
    },
    result: {
      data: {
        addFavoriteFood: {
          id: 456,
          foodItem: {
            id: 789,
            name: newFoodName,
          },
          eatingFrequency: newEatingFrequency,
        },
      },
    },
  },
];
```

Testing with <Mutation />

```
const newFoodName = 'Delicious Food';
const newEatingFrequency = 'DAILY';
const userId = 123;
const mocks = [
  {
    request: {
      query: AddFavoriteFoodToUser.mutation,
      variables: {
        userId,
        name: newFoodName,
        eatingFrequency: newEatingFrequency,
      },
    },
    result: {
      data: {
        addFavoriteFood: {
          id: 456,
          foodItem: {
            id: 789,
            name: newFoodName,
          },
          eatingFrequency: newEatingFrequency,
        },
      },
    },
  },
];
```

Testing with <Mutation />

```
const newFoodName = 'Delicious Food';
const newEatingFrequency = 'DAILY';
const userId = 123;
const mocks = [
  {
    request: {
      query: AddFavoriteFoodToUser.mutation,
      variables: {
        userId,
        name: newFoodName,
        eatingFrequency: newEatingFrequency,
      },
    },
    result: {
      data: {
        addFavoriteFood: {
          id: 456,
          foodItem: {
            id: 789,
            name: newFoodName,
          },
          eatingFrequency: newEatingFrequency,
        },
      },
    },
  },
];
```

Testing with <Mutation />

```
const newFoodName = 'Delicious Food';
const newEatingFrequency = 'DAILY';
const userId = 123;
const mocks = [
  {
    request: {
      query: AddFavoriteFoodToUser.mutation,
      variables: {
        userId,
        name: newFoodName,
        eatingFrequency: newEatingFrequency,
      },
    },
    result: {
      data: {
        addFavoriteFood: {
          id: 456,
          foodItem: {
            id: 789,
            name: newFoodName,
          },
          eatingFrequency: newEatingFrequency,
        },
      },
    },
  },
];
```

Testing with <Mutation />

```
const component = renderComponent(userId, updateWrapper, mocks);

// Enter new food name
component.root.findByType('input').props.onChange({
  target: {value: newFoodName},
});

// Select eating frequency
component.root.findByType('select').props.onChange({
  target: {value: newEatingFrequency},
});

// Submit form
component.root.findByType('form').props.onSubmit({
  preventDefault: jest.fn(),
  target: {reset: jest.fn()},
});

// Make assertions
expect(updateSpy).toHaveBeenCalled();
expect(component.root.findByProps({className: "success-message"}).children).toEqual(['You did it!']);
```

Testing with <Mutation />

```
const component = renderComponent(userId, updateWrapper, mocks);

// Enter new food name
component.root.findByType('input').props.onChange({
  target: {value: newFoodName},
});

// Select eating frequency
component.root.findByType('select').props.onChange({
  target: {value: newEatingFrequency},
});

// Submit form
component.root.findByType('form').props.onSubmit({
  preventDefault: jest.fn(),
  target: {reset: jest.fn()},
});

// Make assertions
expect(updateSpy).toHaveBeenCalled();
expect(component.root.findByProps({className: "success-message"}).children).toEqual(['You did it!']);
```

Testing with <Mutation />

```
const component = renderComponent(userId, updateWrapper, mocks);

// Enter new food name
component.root.findByType('input').props.onChange({
  target: {value: newFoodName},
});

// Select eating frequency
component.root.findByType('select').props.onChange({
  target: {value: newEatingFrequency},
});

// Submit form
component.root.findByType('form').props.onSubmit({
  preventDefault: jest.fn(),
  target: {reset: jest.fn()},
});

// Make assertions
expect(updateSpy).toHaveBeenCalled();
expect(component.root.findByProps({className: "success-message"}).children).toEqual(['You did it!']);
```

Testing with <Mutation />

```
const component = renderComponent(userId, updateWrapper, mocks);

// Enter new food name
component.root.findByType('input').props.onChange({
  target: {value: newFoodName},
});

// Select eating frequency
component.root.findByType('select').props.onChange({
  target: {value: newEatingFrequency},
});

// Submit form
component.root.findByType('form').props.onSubmit({
  preventDefault: jest.fn(),
  target: {reset: jest.fn()},
});

// Make assertions
expect(updateSpy).toHaveBeenCalled();
expect(component.root.findByProps({className: "success-message"}).children).toEqual(['You did it!']);
```

Integration Tests and Lunar

Quick Intro - Cypress

A complete end-to-end testing experience

https://www.cypress.io/

- Fast, easy and reliable testing for anything that runs in a browser.

A simple integration test

```
it('renders the correct number of food items for the user', () => {
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 5);
});
```

Adding Favorite Foods to a User

✔ renders the correct number of food items for the user

BEFORE EACH

1	REQUEST	POST 200 http://localhost:3001/store/…

TEST

1	VISIT	http://localhost:3000/user/1
2	GET	h2
3	- CONTAINS	favorite foods:
	(XHR)	GET 200 /sockjs-node/info?t=153775462…
4	GET	li 5
5	- ASSERT	expected [<li>, 4 more...] to have a length of 5

A simple integration test

```
it('renders the correct number of food items for the user', () => {
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 5);
});
```

Adding Favorite Foods to a User

✔ renders the correct number of food items for the user

BEFORE EACH

1	REQUEST	POST 200 http://localhost:3001/store/…

TEST

1	VISIT	http://localhost:3000/user/1
2	GET	h2
3	- CONTAINS	favorite foods:
	(XHR)	GET 200 /sockjs-node/info?t=153775462…
4	GET	li 5
5	- ASSERT	expected [<li>, 4 more...] to have a length of 5

A simple integration test

```
it('renders the correct number of food items for the user', () => {
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 5);
});
```

```
User: () => ({
  name: faker.name.findName(),
  favoriteFoods: () => new MockList(5),
}),
```

A simple integration test

```
it('renders the correct number of food items for the user', () => {
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 5);
});
```

```
User: () => ({
  name: faker.name.findName(),
  favoriteFoods: () => new MockList(5),
}),
```

The Problem

What issues did we just see in our simple integration test?

- Coupling of our tests with a mocked backend
- Inability to test variations of a components interaction with an API

Introducing Lunar

by ezcater

What problem does Lunar solve?

Decouple your integration tests from your mocked server implementation

https://github.com/ezcater/lunar-core

Lunar in Action: Integration Testing Demo

Skip Backup Slides

Integration tests with Lunar

```
it('renders the correct number of food items for the user', () => {
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 2);
});
```

Integration tests with Lunar

```
it('renders the correct number of food items for the user', () => {
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 2);
});
```

```
  User: () => ({
    name: faker.name.findName(),
    favoriteFoods: () => new MockList(5),
  }),
```

Integration tests with Lunar

```
it('renders the correct number of food items for the user', () => {
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 2);
});
```

Adding Favorite Foods to a User

✖ renders the correct number of food items for the user

BEFORE EACH

1	REQUEST	POST 200 http://localhost:3001/store/…

TEST

1	VISIT	http://localhost:3000/user/1
2	GET	h2
3	- CONTAINS	favorite foods:
	(XHR)	GET 200 /sockjs-node/info?t=153775593…
4	GET	li 5
5	- ASSERT	expected [<li>, 4 more...] to have a length of 2 but got 5

CypressError: Timed out retrying: Too many elements found. Found '5', expected '2'.

Integration tests with Lunar

```
it('renders the correct number of food items for the user', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [{}, {}],
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 2);
});
```

Integration tests with Lunar

```
it('renders the correct number of food items for the user', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [{}, {}],
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 2);
});
```

Integration tests with Lunar

```
it('renders the correct number of food items for the user', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [{}, {}],
      }),
    }),
  });
  cy.visit('http://localhost:3000/user
  cy.get('h2').contains('favorite foo
  cy.get('li').should('have.length', 2);
});
```

Adding Favorite Foods to a User

✔ renders the correct number of food items for the user

BEFORE EACH

1	REQUEST	POST 200 http://localhost:3001/store/…

TEST

1	REQUEST	POST 200 http://localhost:3001/store/…
2	VISIT	http://localhost:3000/user/1
3	GET	h2
4	- CONTAINS	favorite foods:
	(XHR)	GET 200 /sockjs-node/info?t=153775651…
5	GET	li 2
6	- ASSERT	expected [<li>, 1 more...] to have a length of 2

Integration tests with Lunar

```
cy.contains('li', 'I like to eat Spaghetti on a weekly basis');
cy.contains('li', 'I like to eat Coconut Water on a daily basis');
```

Integration tests with Lunar

```
it('renders the correct number of food items for the user', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [
          {foodItem: {name: 'Spaghetti'}, eatingFrequency: 'WEEKLY'},
          {foodItem: {name: 'Coconut Water'}, eatingFrequency: 'DAILY'},
        ],
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 2);
  cy.contains('li', 'I like to eat Spaghetti on a weekly basis');
  cy.contains('li', 'I like to eat Coconut Water on a daily basis');
});
```

Integration tests with Lunar

```
it('renders the correct number of food items for the user', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [
          {foodItem: {name: 'Spaghetti
          {foodItem: {name: 'Coconut W
        ],
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/
  cy.get('h2').contains('favorite food
  cy.get('li').should('have.length', 2
  cy.contains('li', 'I like to eat Spa
  cy.contains('li', 'I like to eat Coc
});
```

Adding Favorite Foods to a User

✔ renders the correct number of food items for the user

BEFORE EACH

1	REQUEST	POST 200 http://localhost:3001/store/…

TEST

1	REQUEST	POST 200 http://localhost:3001/store/…
2	VISIT	http://localhost:3000/user/1
3	GET	h2
4	- CONTAINS	favorite foods:
	(XHR)	*GET 200 /sockjs-node/info?t=153775690…*
5	GET	li 2
6	- ASSERT	expected [<li>, 1 more...] to have a length of 2
7	CONTAINS	li, I like to eat Spaghetti on a weekly…
8	CONTAINS	li, I like to eat Coconut Water on a da…

Mutation integration tests with Lunar

```
it('Adds the new food item to the list and blanks out the form', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [],
      }),
    }),
    Mutation: () => ({
      addFavoriteFood: () => ({
        foodItem: {
          name: 'Mac & Cheese',
        },
        eatingFrequency: 'WEEKLY',
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.contains('div', 'No favorites yet!');
  cy.get('input').type('Mac & Cheese');
  cy.get('select').select('WEEKLY');
  cy.get('button').click();
  cy.contains('li', 'I like to eat Mac & Cheese on a weekly basis');
  cy.get('input').should('have.value', '');
  cy.get('select').should('have.value', null);
  cy.contains('div', 'You did it!');
});
```

Mutation integration tests with Lunar

```
it('Adds the new food item to the list and blanks out the form', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [],
      }),
    }),
    Mutation: () => ({
      addFavoriteFood: () => ({
        foodItem: {
          name: 'Mac & Cheese',
        },
        eatingFrequency: 'WEEKLY',
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.contains('div', 'No favorites yet!');
  cy.get('input').type('Mac & Cheese');
  cy.get('select').select('WEEKLY');
  cy.get('button').click();
  cy.contains('li', 'I like to eat Mac & Cheese on a weekly basis');
  cy.get('input').should('have.value', '');
  cy.get('select').should('have.value', null);
  cy.contains('div', 'You did it!');
});
```

Mutation integration tests with Lunar

```
it('Adds the new food item to the list and blanks out the form', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [],
      }),
    }),
    Mutation: () => ({
      addFavoriteFood: () => ({
        foodItem: {
          name: 'Mac & Cheese',
        },
        eatingFrequency: 'WEEKLY',
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.contains('div', 'No favorites yet!');
  cy.get('input').type('Mac & Cheese');
  cy.get('select').select('WEEKLY');
  cy.get('button').click();
  cy.contains('li', 'I like to eat Mac & Cheese on a weekly basis');
  cy.get('input').should('have.value', '');
  cy.get('select').should('have.value', null);
  cy.contains('div', 'You did it!');
});
```

Mutation integration tests with Lunar

```
it('Adds the new food item to the list and blanks out the form', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [],
      }),
    }),
    Mutation: () => ({
      addFavoriteFood: () => ({
        foodItem: {
          name: 'Mac & Cheese',
        },
        eatingFrequency: 'WEEKLY',
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.contains('div', 'No favorites yet!');
  cy.get('input').type('Mac & Cheese');
  cy.get('select').select('WEEKLY');
  cy.get('button').click();
  cy.contains('li', 'I like to eat Mac & Cheese on a weekly basis');
  cy.get('input').should('have.value', '');
  cy.get('select').should('have.value', null);
  cy.contains('div', 'You did it!');
});
```

Mutation integration tests with Lunar

```
it('Adds the new food item to the list and blanks out the form', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [],
      }),
    }),
    Mutation: () => ({
      addFavoriteFood: () => ({
        foodItem: {
          name: 'Mac & Cheese',
        },
        eatingFrequency: 'WEEKLY',
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.contains('div', 'No favorites yet!');
  cy.get('input').type('Mac & Cheese');
  cy.get('select').select('WEEKLY');
  cy.get('button').click();
  cy.contains('li', 'I like to eat Mac & Cheese on a weekly basis');
  cy.get('input').should('have.value', '');
  cy.get('select').should('have.value', null);
  cy.contains('div', 'You did it!');
});
```

Mutation integration tests with Lunar

```
it('Adds the new food item to the list and blanks out the form', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [],
      }),
    }),
    Mutation: () => ({
      addFavoriteFood: () => ({
        foodItem: {
          name: 'Mac & Cheese',
        },
        eatingFrequency: 'WEEKLY',
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.contains('div', 'No favorites yet!');
  cy.get('input').type('Mac & Cheese');
  cy.get('select').select('WEEKLY');
  cy.get('button').click();
  cy.contains('li', 'I like to eat Mac & Cheese on a weekly basis');
  cy.get('input').should('have.value', '');
  cy.get('select').should('have.value', null);
  cy.contains('div', 'You did it!');
});
```

Mutation integration tests with Lunar

```
it('Adds the new food item to the list and blanks out the form', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [],
      }),
    }),
    Mutation: () => ({
      addFavoriteFood: () => ({
        foodItem: {
          name: 'Mac & Cheese',
        },
        eatingFrequency: 'WEEKLY',
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.contains('div', 'No favorites yet!');
  cy.get('input').type('Mac & Cheese');
  cy.get('select').select('WEEKLY');
  cy.get('button').click();
  cy.contains('li', 'I like to eat Mac & Cheese on a weekly basis');
  cy.get('input').should('have.value', '');
  cy.get('select').should('have.value', null);
  cy.contains('div', 'You did it!');
});
```

Mutation integration tests with Lunar

```
it('Adds the new food item to the list and blanks out the form', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [],
      }),
    }),
    Mutation: () => ({
      addFavoriteFood: () => ({
        foodItem: {
          name: 'Mac & Cheese',
        },
        eatingFrequency: 'WEEKLY',
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.contains('div', 'No favorites yet!');
  cy.get('input').type('Fluffernutter Sandwich');
  cy.get('select').select('DAILY');
  cy.get('button').click();
  cy.contains('li', 'I like to eat Mac & Cheese on a weekly basis');
  cy.get('input').should('have.value', '');
  cy.get('select').should('have.value', null);
  cy.contains('div', 'You did it!');
});
```

Mutation integration tests with Lunar

```
it('Adds the new food item to the list and blanks out the form', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [],
      }),
    }),
    Mutation: () => ({
      addFavoriteFood: (parent, args) => ({
        foodItem: {
          name: args.name,
        },
        eatingFrequency: args.eatingFrequency,
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.contains('div', 'No favorites yet!');
  cy.get('input').type('Fluffernutter Sandwich');
  cy.get('select').select('DAILY');
  cy.get('button').click();
  cy.contains('li', 'I like to eat Fluffernutter Sandwich on a daily basis');
  cy.get('input').should('have.value', '');
  cy.get('select').should('have.value', null);
  cy.contains('div', 'You did it!');
});
```

Mutation integration tests with Lunar

```
it('Adds the new food item to the list and blanks out the form', () => {
  cy.mock({
    Query: () => ({
      user: () => ({
        favoriteFoods: [],
      }),
    }),
    Mutation: () => ({
      addFavoriteFood: (parent, args) => ({
        foodItem: {
          name: args.name,
        },
        eatingFrequency: args.eatingFrequency,
      }),
    }),
  });
  cy.visit('http://localhost:3000/user/1');
  cy.contains('div', 'No favorites yet!');
  cy.get('input').type('Fluffernutter Sandwich');
  cy.get('select').select('DAILY');
  cy.get('button').click();
  cy.contains('li', 'I like to eat Fluffernutter Sandwich on a daily basi
  cy.get('input').should('have.value', '');
  cy.get('select').should('have.value', null);
  cy.contains('div', 'You did it!');
});
```

✔ Adds the new food item to the list and blanks out the form

BEFORE EACH

1	REQUEST	POST 200 http://localhost:3001/store…

TEST

1	REQUEST	POST 200 http://localhost:3001/store…
2	VISIT	http://localhost:3000/user/1
3	CONTAINS	div, No favorites yet!
4	GET	input
5	- TYPE	Fluffernutter Sandwich
	(XHR)	*GET 200 /sockjs-node/info?t=15377588…*
6	GET	select
7	- SELECT	DAILY
8	GET	button
9	- CLICK	
10	CONTAINS	li, I like to eat Fluffernutter Sandwi…
11	GET	input
12	- ASSERT	expected <input#name> to have value '…
13	GET	select
14	- ASSERT	expected <select#eatingFrequency> to have value null
15	CONTAINS	div, You did it!

Recap: without Lunar

```
it('renders the correct number of food items for the user', () => {
  cy.visit('http://localhost:3000/user/1');
  cy.get('h2').contains('favorite foods:');
  cy.get('li').should('have.length', 5);
});
```

```
User: () => ({
  name: faker.name.findName(),
  favoriteFoods: () => new MockList(5),
}),
```

Recap: without Lunar

```
updateAddress: (_obj, address) => {
  let response;

  if (/multiple/i.test(address.city)) {
    response = {
      __typename: 'AddressGeocodeMultiplePayload',
      addresses: [fakeAddress(), fakeAddress()],
    };
  } else if (/error/i.test(address.city)) {
    response = {};
  } else {
    response = {
      __typename: 'AddressSaveSuccessPayload',
      address: fakeAddress(),
    };
  }

  return response;
},
```

Recap: without Lunar

```
updateAddress: (_obj, address) => {
  let response;

  if (/multiple/i.test(address.city)) {
    response = {
      __typename: 'AddressGeocodeMultiplePayload',
      addresses: [fakeAddress(), fakeAddress()],
    };
  } else if (/error/i.test(address.city)) {
    response = {};
  } else {
    response = {
      __typename: 'AddressSaveSuccessPayload',
      address: fakeAddress(),
    };
  }

  return response;
},
```

Recap: without Lunar

```
updateAddress: (_obj, address) => {
  let response;

  if (/multiple
    response
      __typ
      add
    };
  } else
    resp
  } else
    respo
      __typ
      address
    };
  }

  return response;
},
```

Recap: with Lunar

```
cy.mock({
  Query: () => ({
    user: () => ({
      favoriteFoods: [],
    }),
  }),
  Mutation: () => ({
    addFavoriteFood: (parent, args) => ({
      foodItem: {
        name: args.name,
      },
      eatingFrequency: args.eatingFrequency,
    }),
  }),
});
```

Recap: with Lunar

```
cy.mock({
  Query: () => ({
    user: () => ({
      favoriteFoods: [],
    }),
  }),
  Mutation: () => ({
    addFavoriteFood: (parent, args) => ({
      foodItem: {
        name: args.name,
      },
      eatingFrequency: args.eatingFrequency,
    }),
  }),
});
```

```
// Navigate to page
cy.visit('http://localhost:3000/user/1');

// Validate page content
cy.contains('div', 'No favorites yet!');

// Fill in and submit form (triggers mutation)
cy.get('input').type('Fluffernutter Sandwich');
cy.get('select').select('DAILY');
cy.get('button').click();

// Validate that mutation result displays on screen
cy.contains(
  'li',
  'I like to eat Fluffernutter Sandwich on a daily basis'
);

// Validate form is reset
cy.get('input').should('have.value', '');
cy.get('select').should('have.value', null);

// Validate success message
cy.contains('div', 'You did it!');
```

Recap: with Lunar

Special Thanks

Craig Cavalier

Special Thanks

Craig Cavalier

https://ezcater.github.io/recipe/

Special Thanks

We’re Hiring!

https://www.ezcater.com/company/apply/

Thank you!

https://github.com/mjfaga/react-boston-2018-lunar-launch
https://github.com/ezcater/lunar-core

Hillary Bauer

bauer-hillary-c
@mama_bau

Mark Faga

mjfaga
@markjfaga