Module 3

Purwadhika
Digital Technology School

Fullstack Web Development

TDD in Backend Development

Job Connector Program

Intro

Test-driven development (TDD) is an evolutionary approach to development which combines test-first development where you write a test before you write just enough production code to fulfill that test and refactoring. In simple terms, test cases are created before code is written.

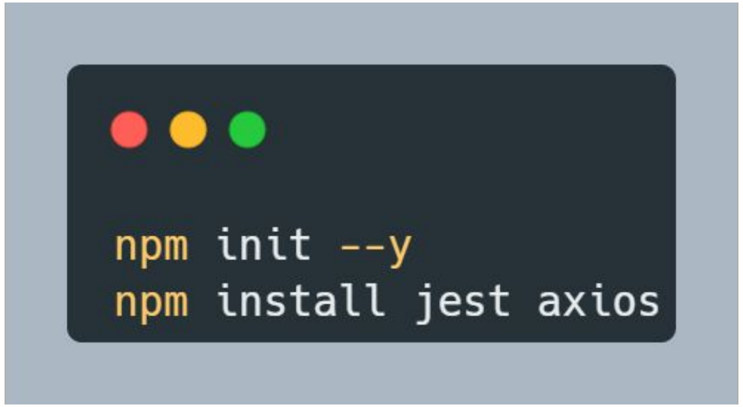
Mocking in Jest

Since you already know how to create unit testing in fundamental module, in this case we will implement mock function using jest into our projects.

Mock functions allow you to test the links between code by erasing the actual implementation of a function, capturing calls to the function (and the parameters passed in those calls), capturing instances of constructor functions when instantiated with new, and allowing test-time configuration of return values.

Mocking in Jest

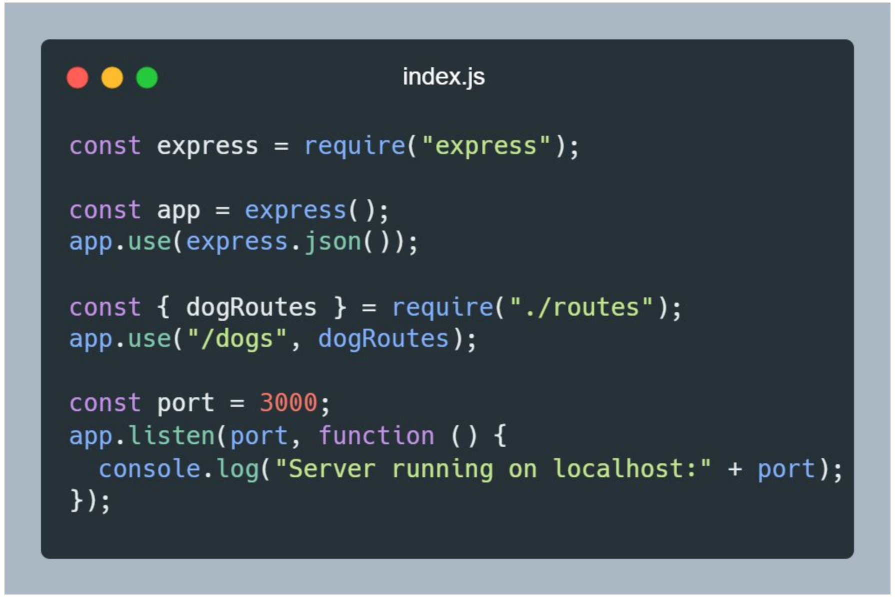
Lets create a new projects. In this case we will add axios package. We would mock result from function that use axios.



```
npm init --y
npm install jest axios
```

Mocking in Jest

Create index.js in the root of projects. And write down this code


index.js

```
const express = require("express");

const app = express();
app.use(express.json());

const { dogRoutes } = require("./routes");
app.use("/dogs", dogRoutes);

const port = 3000;
app.listen(port, function () {
  console.log("Server running on localhost:" + port);
});
```

Mocking in Jest

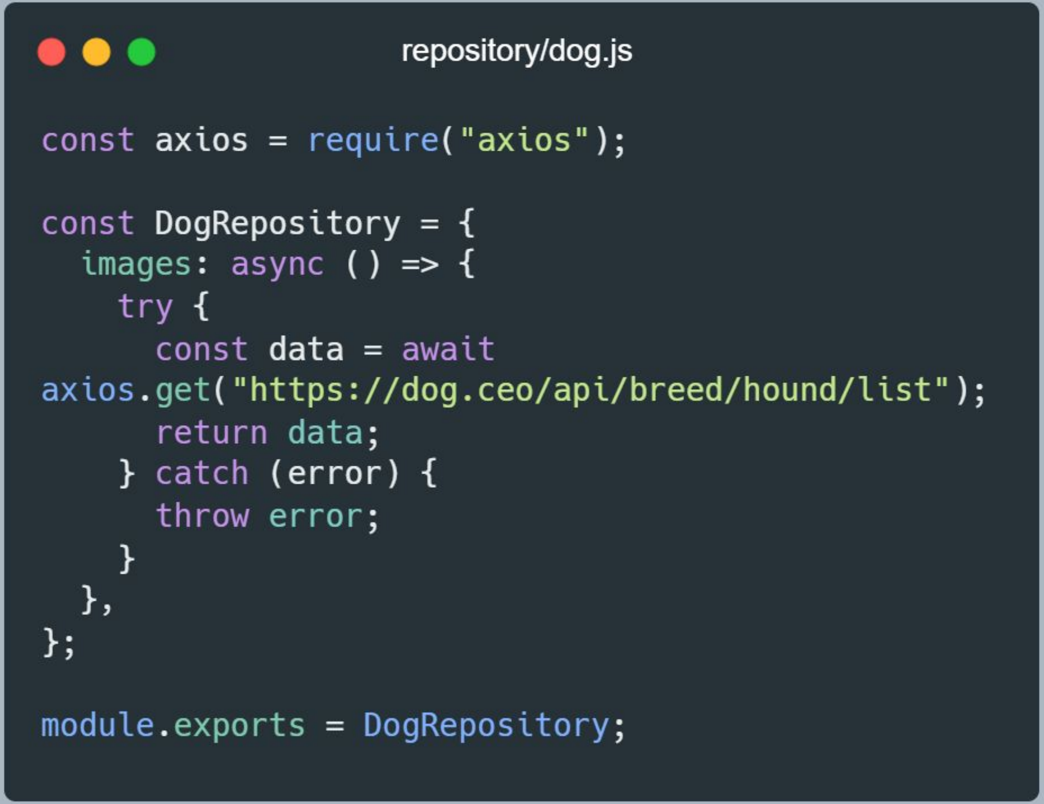
Create this directory in root of project:

- routes
  - index.js
  - dog.js
- controllers
  - index.js
  - dog.js
- repository
  - dog.js
- test
  - dog.test.js

Mocking in Jest

We would start modify our repository/dog.js

Write down this code!



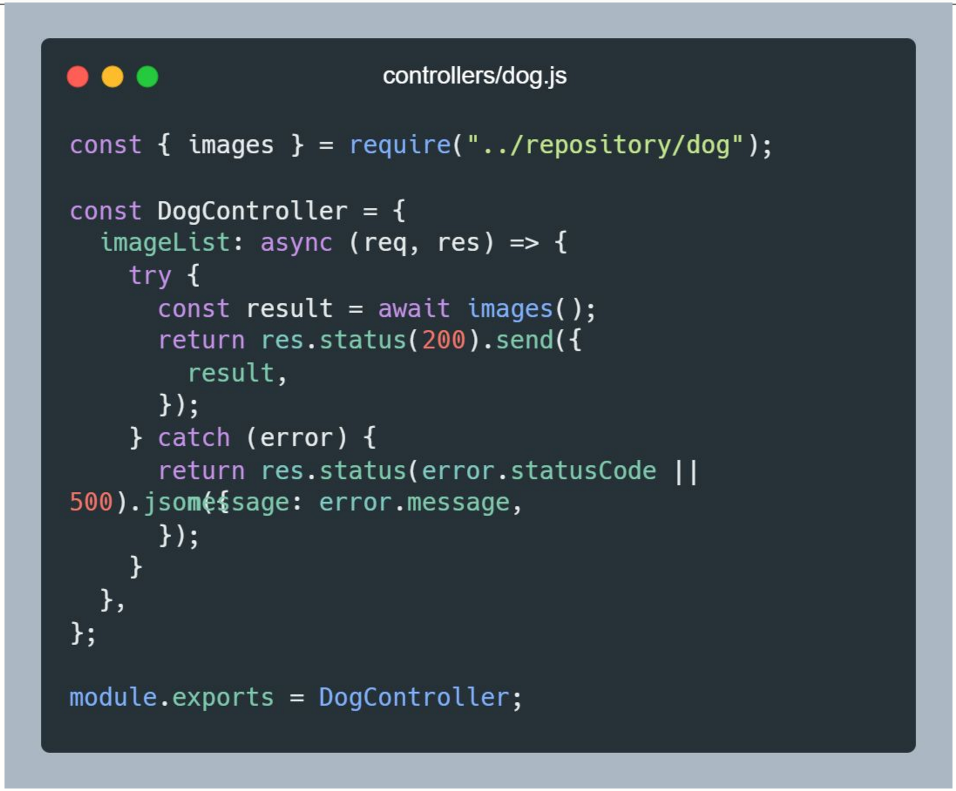
repository/dog.js

```
const axios = require("axios");

const DogRepository = {
  images: async () => {
    try {
      const data = await
axios.get("https://dog.ceo/api/breed/hound/list");
      return data;
    } catch (error) {
      throw error;
    }
  },
};

module.exports = DogRepository;
```

Mocking in Jest

We would modify our controllers/dog.js

Write down this code!

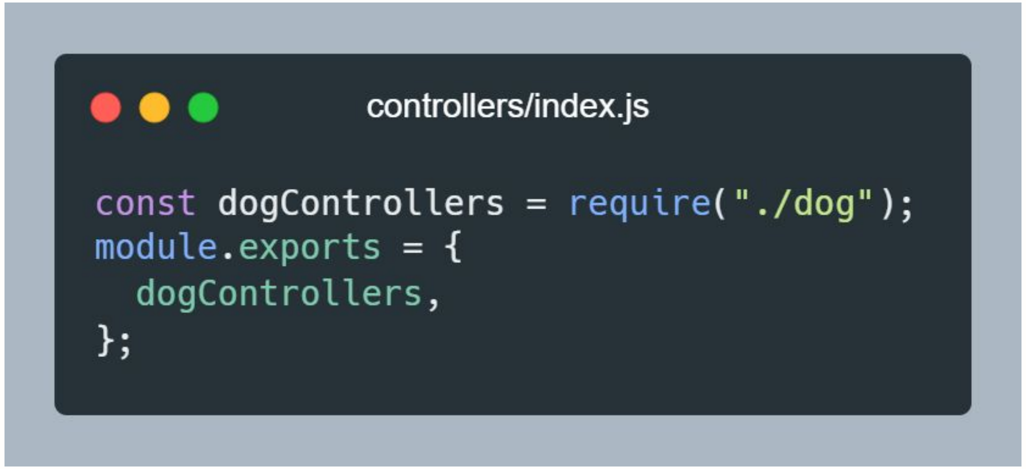

controllers/dog.js

```
const { images } = require("../repository/dog");

const DogController = {
  imageList: async (req, res) => {
    try {
      const result = await images();
      return res.status(200).send({
        result,
      });
    } catch (error) {
      return res.status(error.statusCode ||
500).json({ message: error.message,
      });
    }
  },
};

module.exports = DogController;
```

Mocking in Jest

We would modify our controllers/index.js

Write down this code!



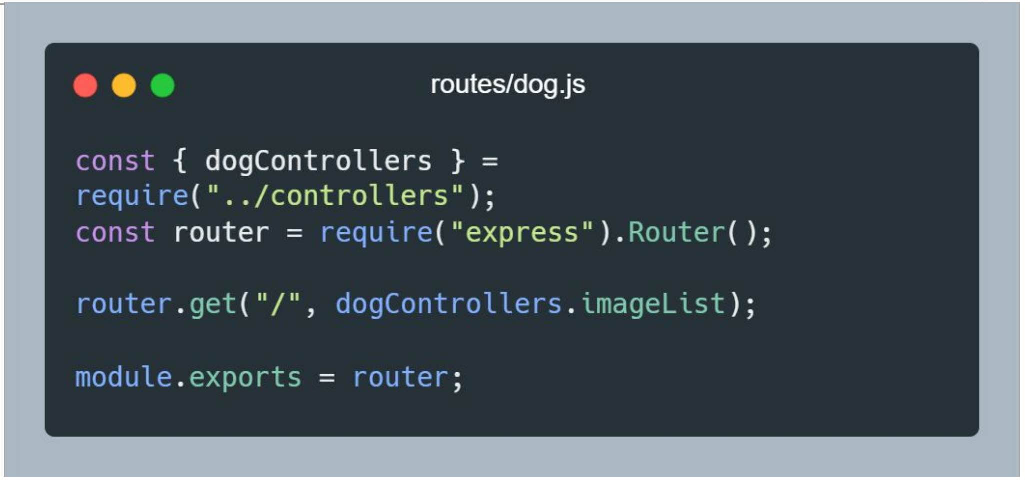
controllers/index.js

```javascript
const dogControllers = require("./dog");
module.exports = {
  dogControllers,
};
```

Mocking in Jest

We would modify our routes/dog.js

Write down this code!

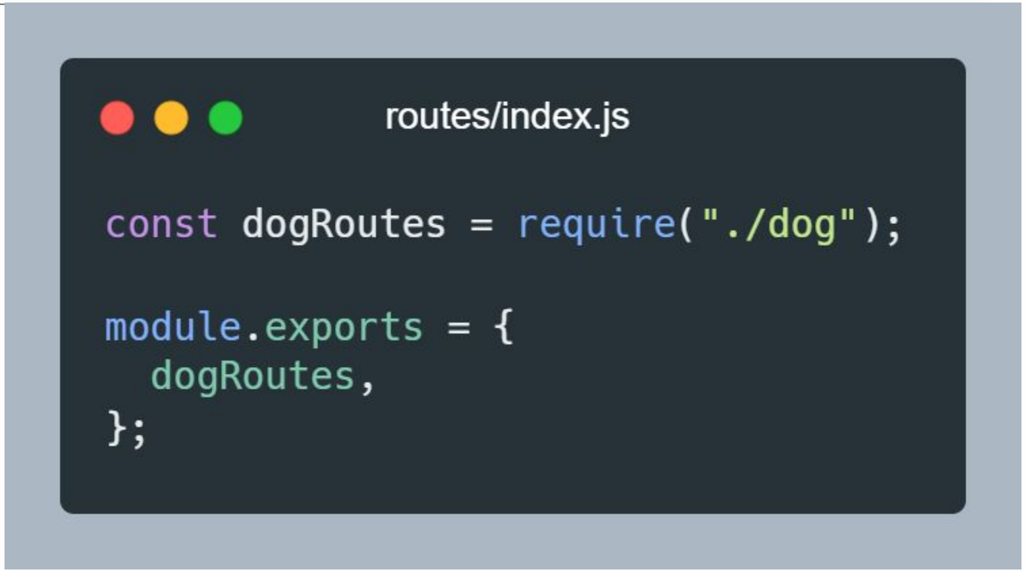

routes/dog.js

```
const { dogControllers } =
require("../controllers");
const router = require("express").Router();

router.get("/", dogControllers.imageList);

module.exports = router;
```

Mocking in Jest

We would modify our routes/index.js

Write down this code!



routes/index.js

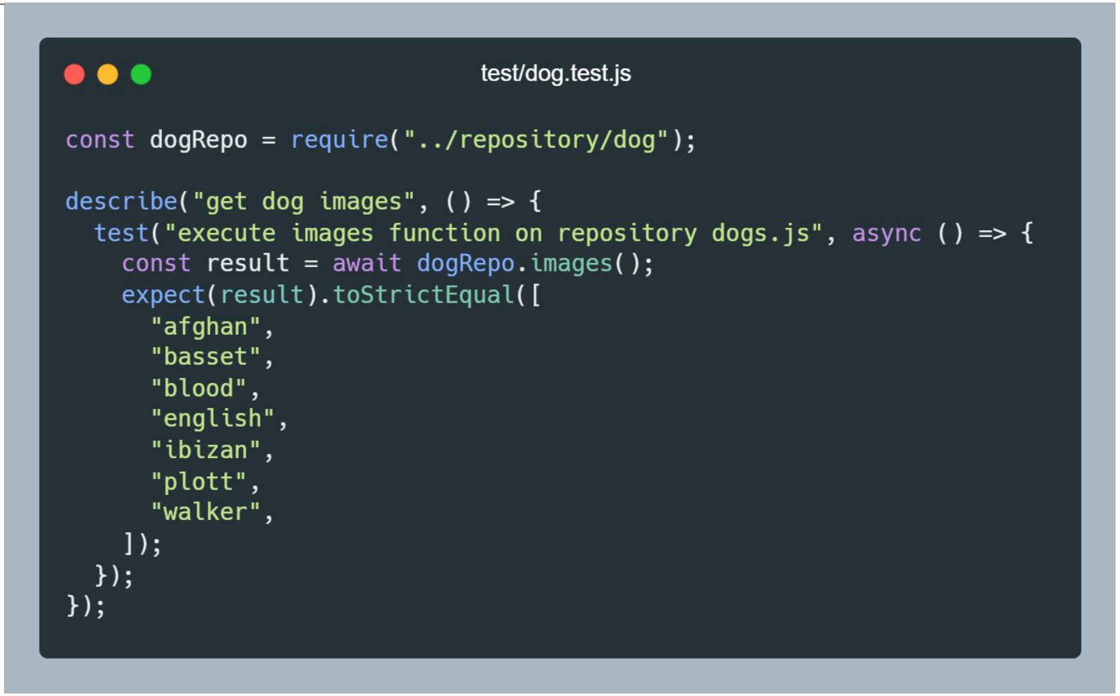
```
const dogRoutes = require("./dog");

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

Mocking in Jest

Now we already have simple application that calls list a dogs.

Lets write down the test case!

We would create a unit test based on repository/dog.js file.



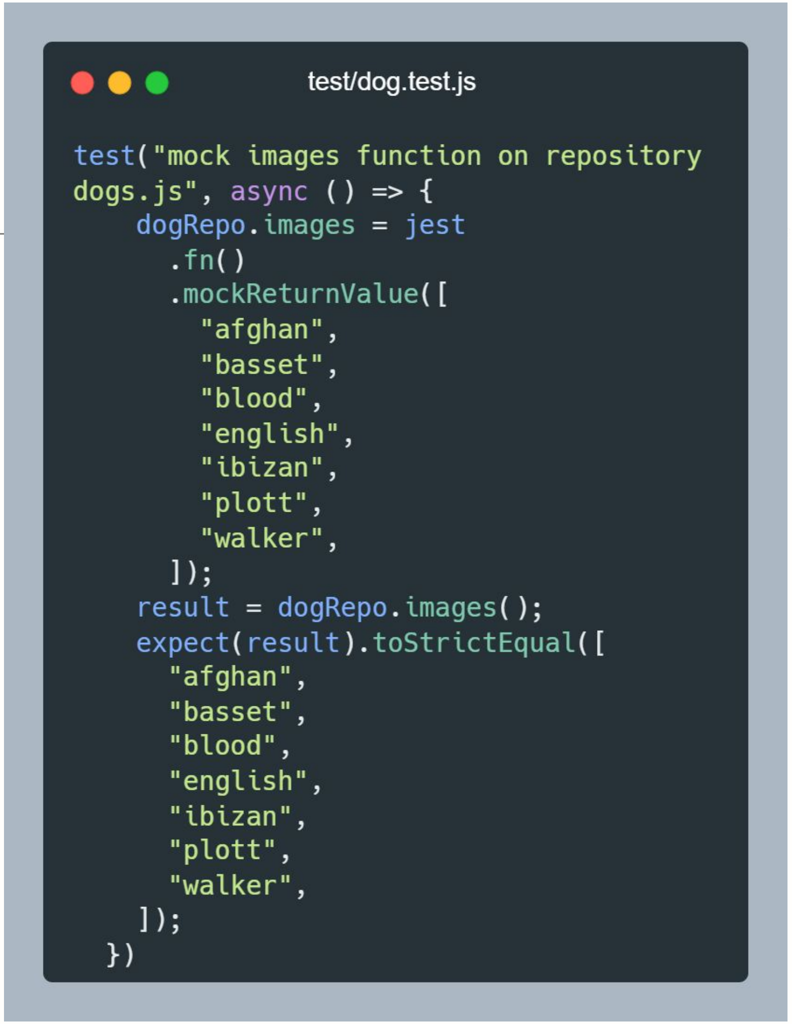
test/dog.test.js

```
const dogRepo = require("../repository/dog");

describe("get dog images", () => {
  test("execute images function on repository dogs.js", async () => {
    const result = await dogRepo.images();
    expect(result).toStrictEqual([
      "afghan",
      "basset",
      "blood",
      "english",
      "ibizan",
      "plott",
      "walker",
    ]);
  });
});
```

Mocking in Jest

Lets mock images function result using jest.fn().mockResultValue()

We would overrides the function result, in this case, network call wont happen.



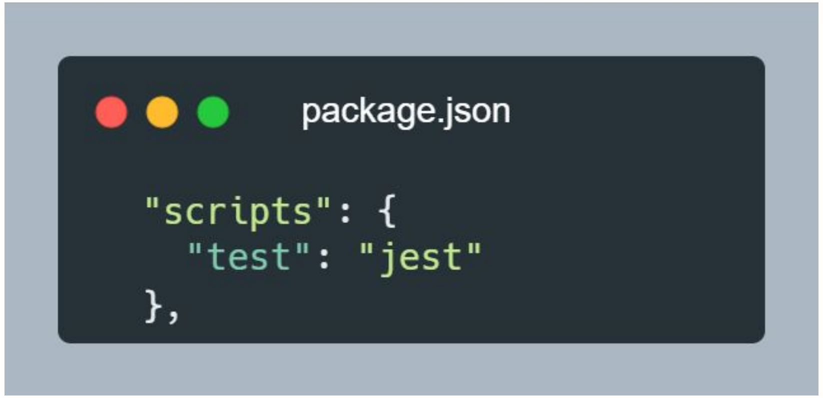
test/dog.test.js

```
test("mock images function on repository
dogs.js", async () => {
    dogRepo.images = jest
      .fn()
      .mockReturnValue([
        "afghan",
        "basset",
        "blood",
        "english",
        "ibizan",
        "plott",
        "walker",
      ]);
    result = dogRepo.images();
    expect(result).toStrictEqual([
      "afghan",
      "basset",
      "blood",
      "english",
      "ibizan",
      "plott",
      "walker",
    ]);
  })
```

Mocking in Jest

Modify your package.json and add test on scripts section.

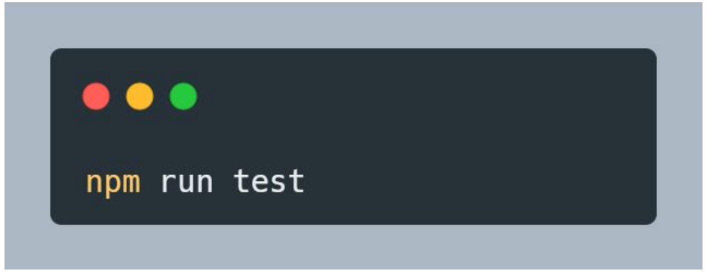
This would add test script into your projects



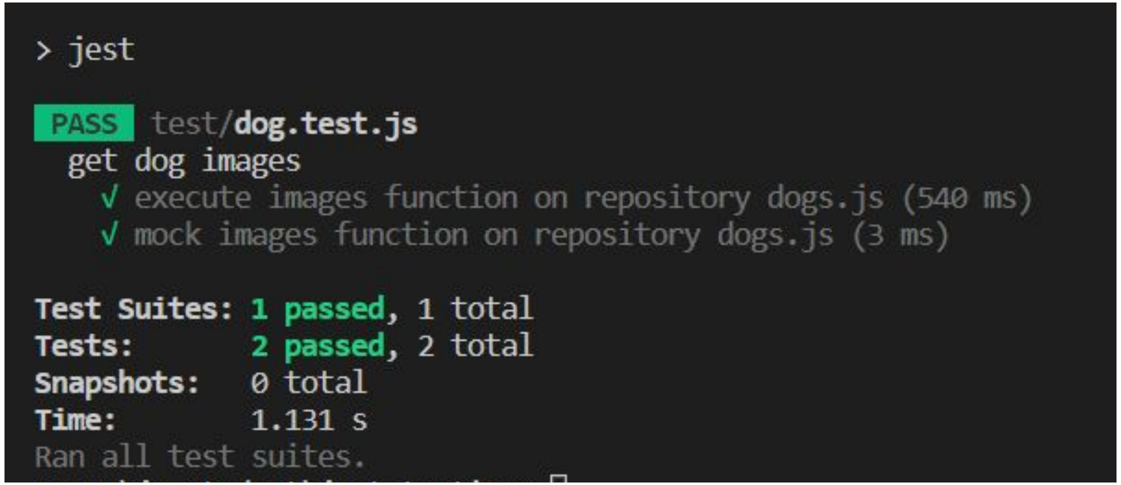
```
"scripts": {
  "test": "jest"
},
```

Mocking in Jest

Lets execute our unit test by running the test script that we just created before. Execute the command and you will see the testing result on terminal.



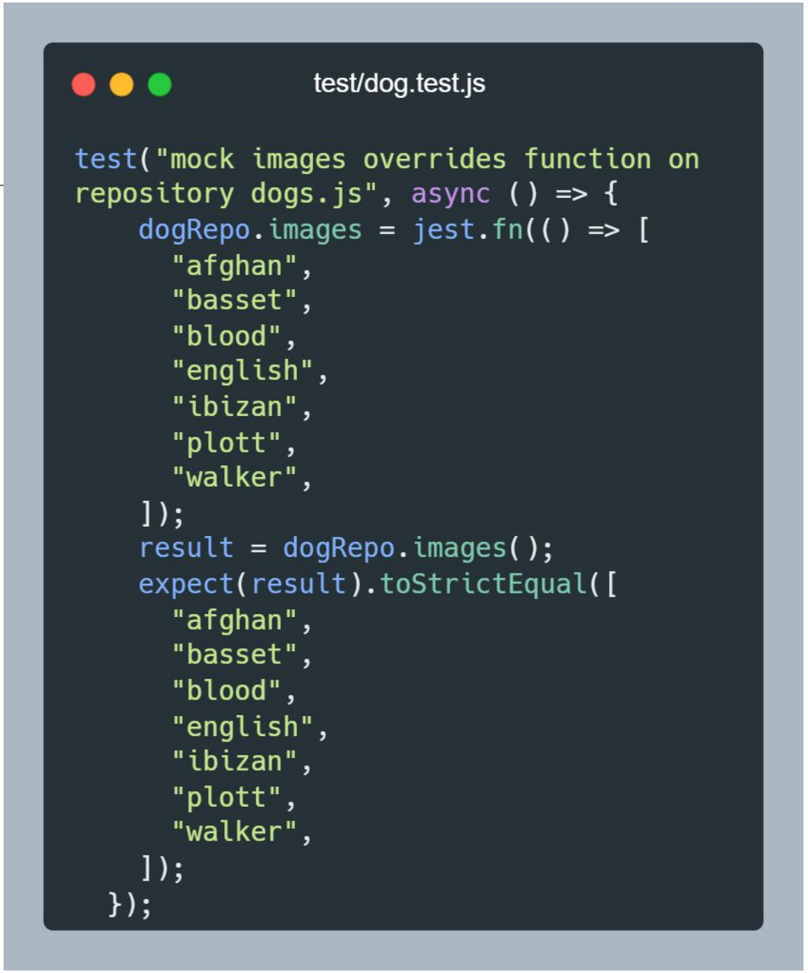
```
npm run test
```



```
> jest

PASS  test/dog.test.js
  get dog images
    √ execute images function on repository dogs.js (540 ms)
    √ mock images function on repository dogs.js (3 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.131 s
Ran all test suites.
```

Mocking in Jest

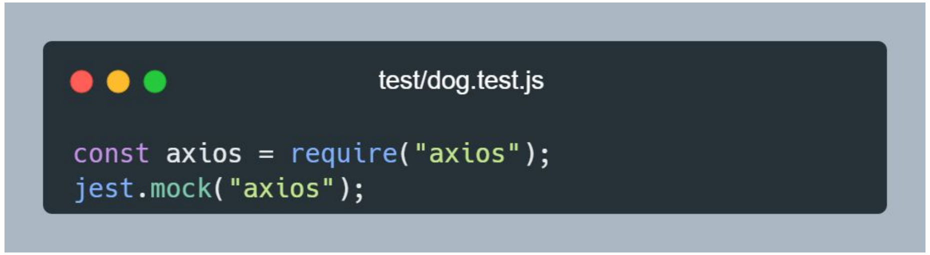
You could even mock a function by overrides the existing function using jest.fn().


test/dog.test.js

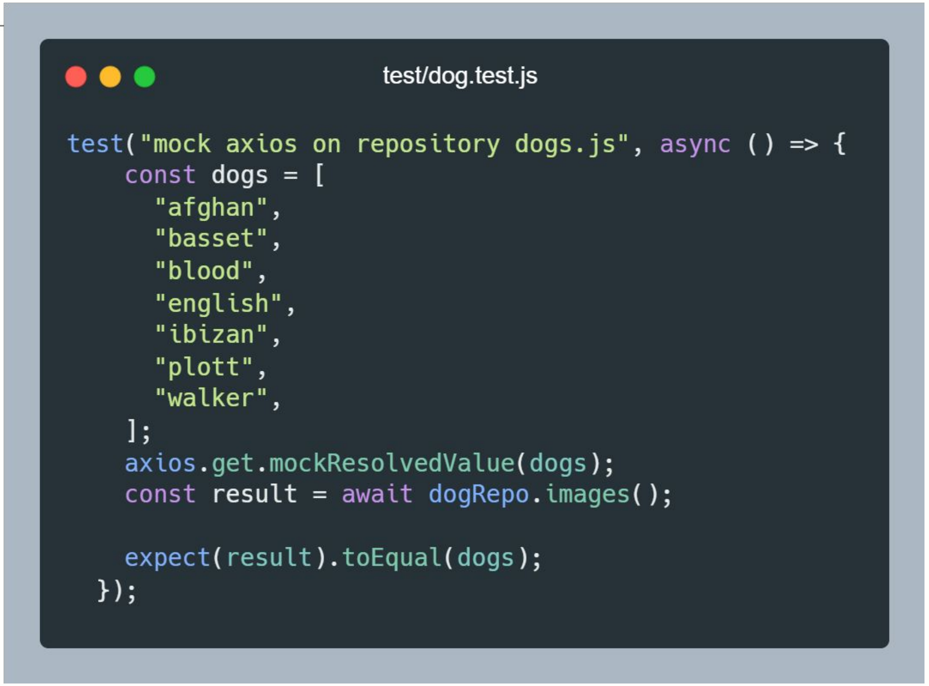
```
test("mock images overrides function on
repository dogs.js", async () => {
    dogRepo.images = jest.fn(() => [
      "afghan",
      "basset",
      "blood",
      "english",
      "ibizan",
      "plott",
      "walker",
    ]);
    result = dogRepo.images();
    expect(result).toStrictEqual([
      "afghan",
      "basset",
      "blood",
      "english",
      "ibizan",
      "plott",
      "walker",
    ]);
  });
```

Mocking in Jest

This time, lets mock the axios packages and create the unit test


test/dog.test.js

```javascript
const axios = require("axios");
jest.mock("axios");
```

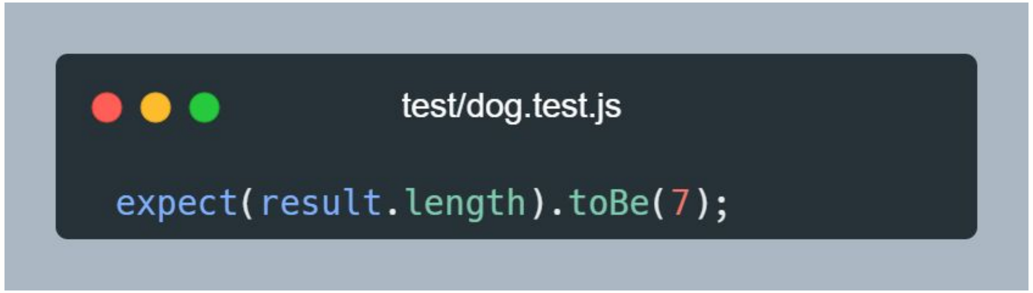

test/dog.test.js

```javascript
test("mock axios on repository dogs.js", async () => {
    const dogs = [
      "afghan",
      "basset",
      "blood",
      "english",
      "ibizan",
      "plott",
      "walker",
    ];
    axios.get.mockResolvedValue(dogs);
    const result = await dogRepo.images();

    expect(result).toEqual(dogs);
  });
```

Mocking in Jest

You could also use toBe method after except as a test case.

This method usually used for type of string or number.

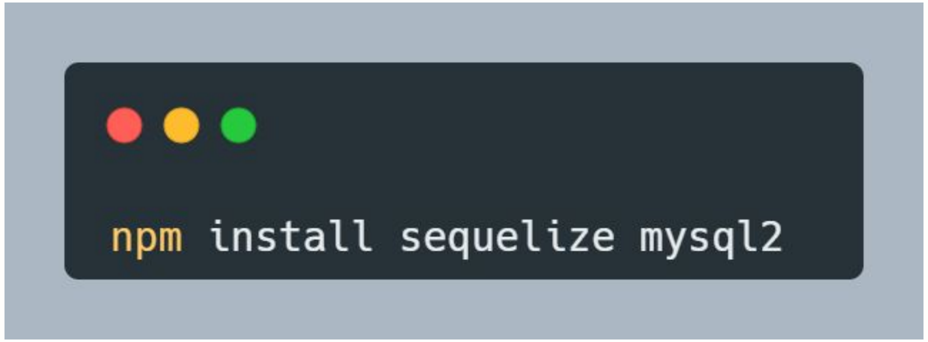

test/dog.test.js

```
expect(result.length).toBe(7);
```

Mocking in Jest

Lets implement mock into database connection. This time we will implement sequelize as orm and mock that function in unit test.

Starting by install mysql2, and sequelize package



```
npm install sequelize mysql2
```

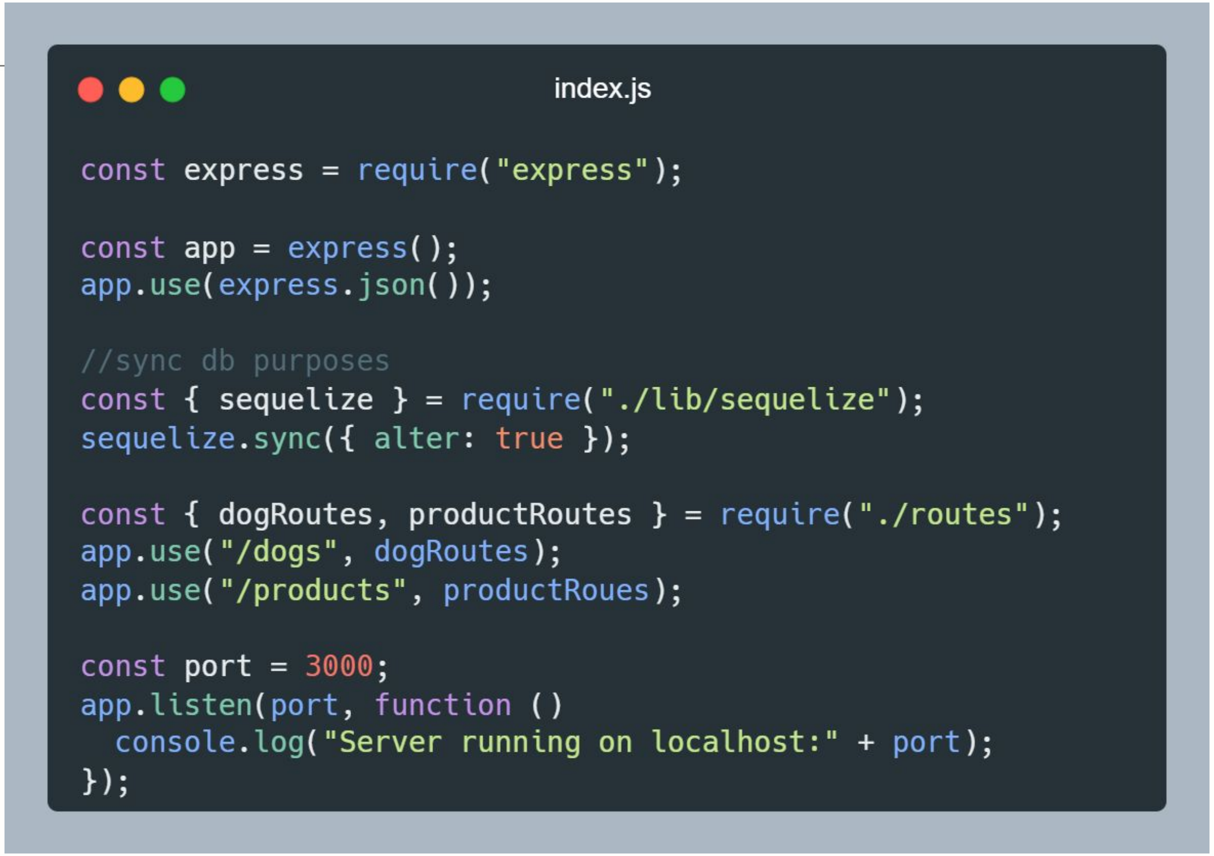
Mocking in Jest

In this part we will create several folder and file.

- lib
  - sequelize.js
- controllers
  - product.js
- routes
  - product.js
- repository
  - product.js
- models
  - product.js
- test
  - product.test.js

Mocking in Jest

We will use sequelize sync approach through our database. Lets modify a bit our index.js on root of projects.



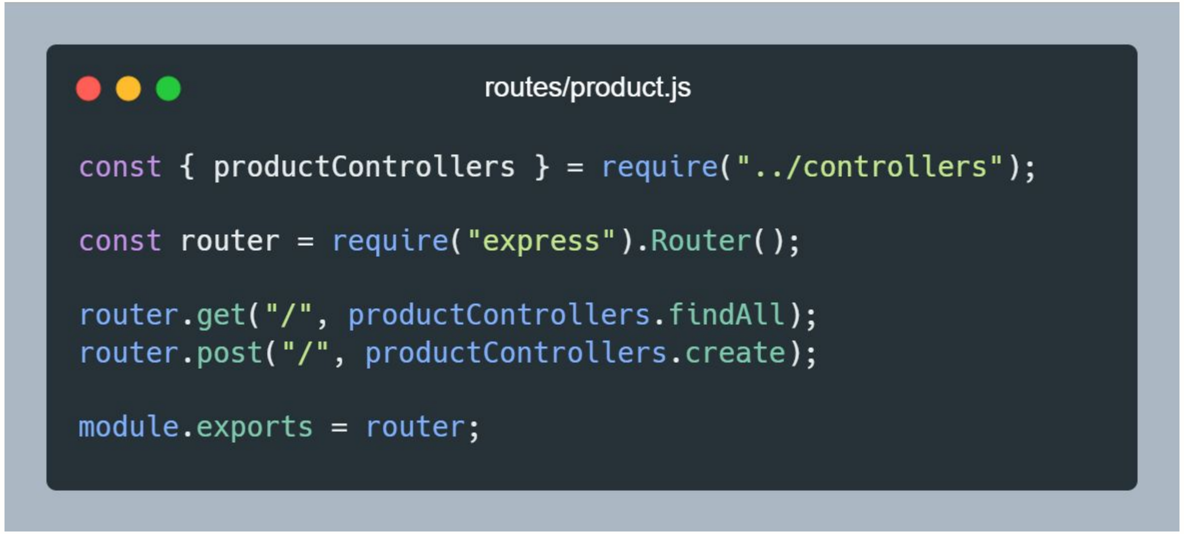
index.js

```
const express = require("express");

const app = express();
app.use(express.json());

//sync db purposes
const { sequelize } = require("./lib/sequelize");
sequelize.sync({ alter: true });

const { dogRoutes, productRoutes } = require("./routes");
app.use("/dogs", dogRoutes);
app.use("/products", productRoues);

const port = 3000;
app.listen(port, function ()
  console.log("Server running on localhost:" + port);
});
```

Mocking in Jest

Create routes/product.js file

Write down this code!

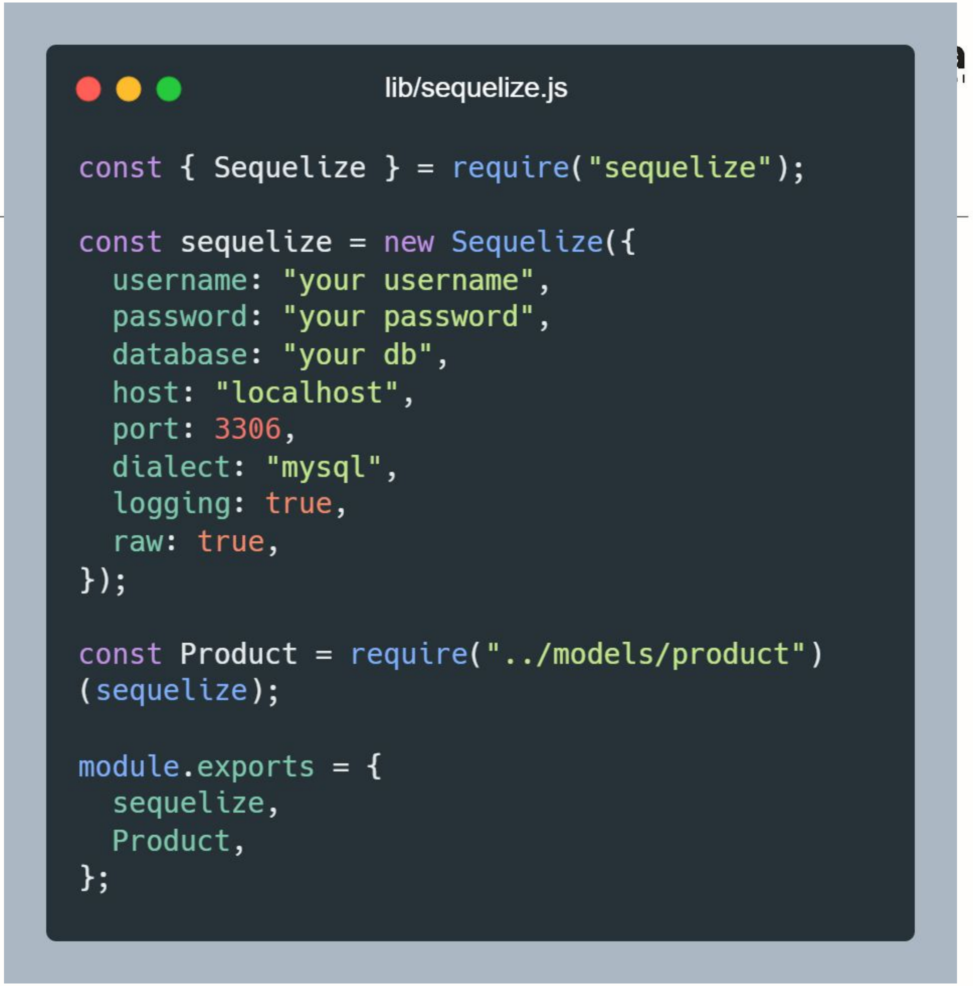

routes/product.js

```javascript
const { productControllers } = require("../controllers");

const router = require("express").Router();

router.get("/", productControllers.findAll);
router.post("/", productControllers.create);

module.exports = router;
```

Mocking in Jest

Create lib/sequelize.js file

Write down this code!

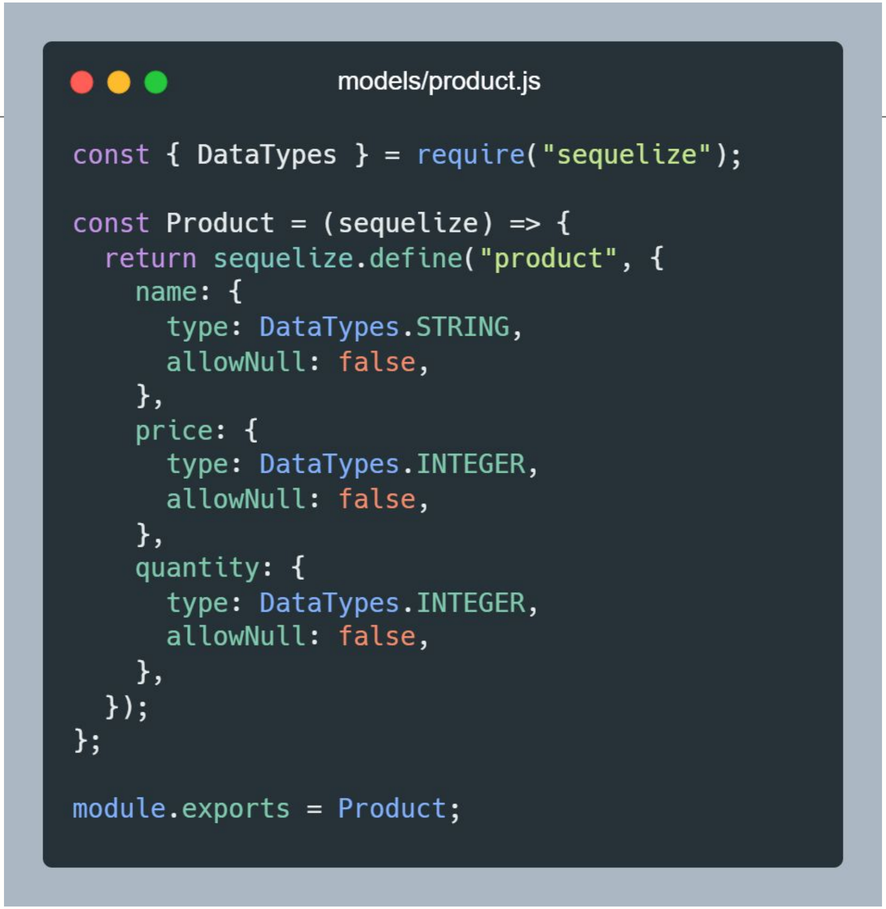

lib/sequelize.js

```
const { Sequelize } = require("sequelize");

const sequelize = new Sequelize({
  username: "your username",
  password: "your password",
  database: "your db",
  host: "localhost",
  port: 3306,
  dialect: "mysql",
  logging: true,
  raw: true,
});

const Product = require("../models/product")
(sequelize);

module.exports = {
  sequelize,
  Product,
};
```

Mocking in Jest

Create models/product.js file

Write down this code!



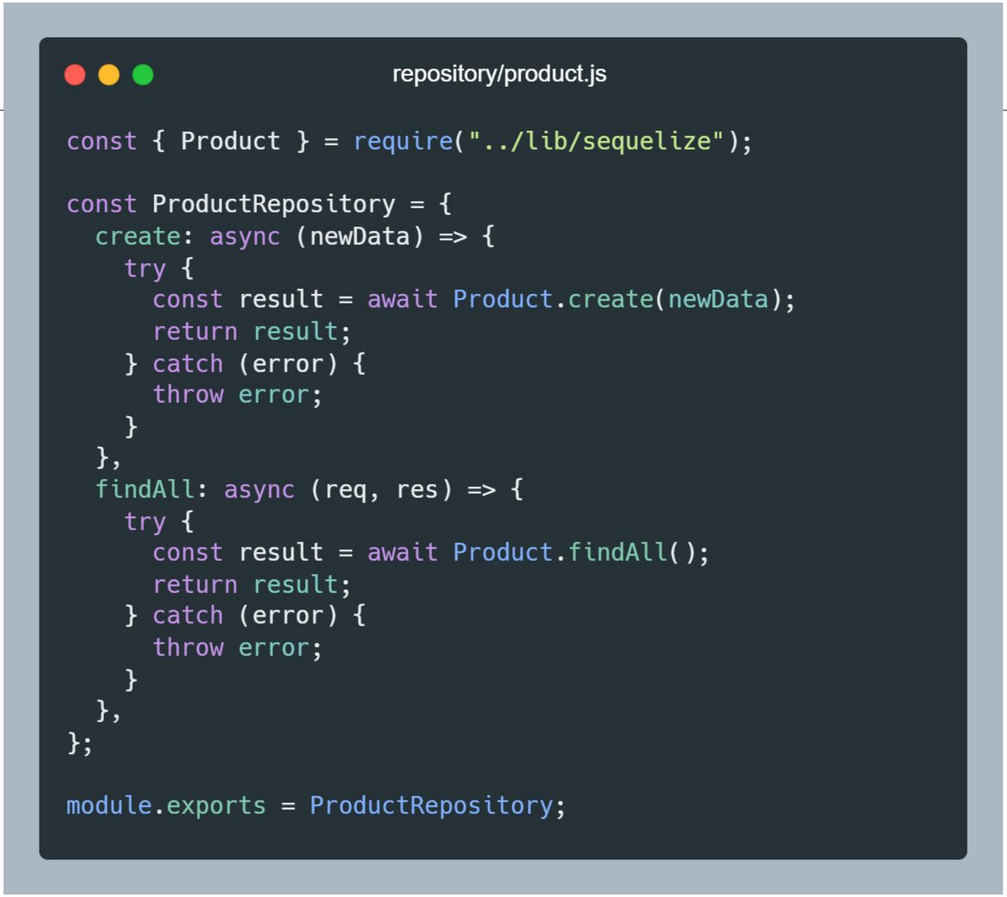
models/product.js

```
const { DataTypes } = require("sequelize");

const Product = (sequelize) => {
  return sequelize.define("product", {
    name: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    price: {
      type: DataTypes.INTEGER,
      allowNull: false,
    },
    quantity: {
      type: DataTypes.INTEGER,
      allowNull: false,
    },
  });
};

module.exports = Product;
```

Mocking in Jest

Create repository/product.js file

Write down this code!

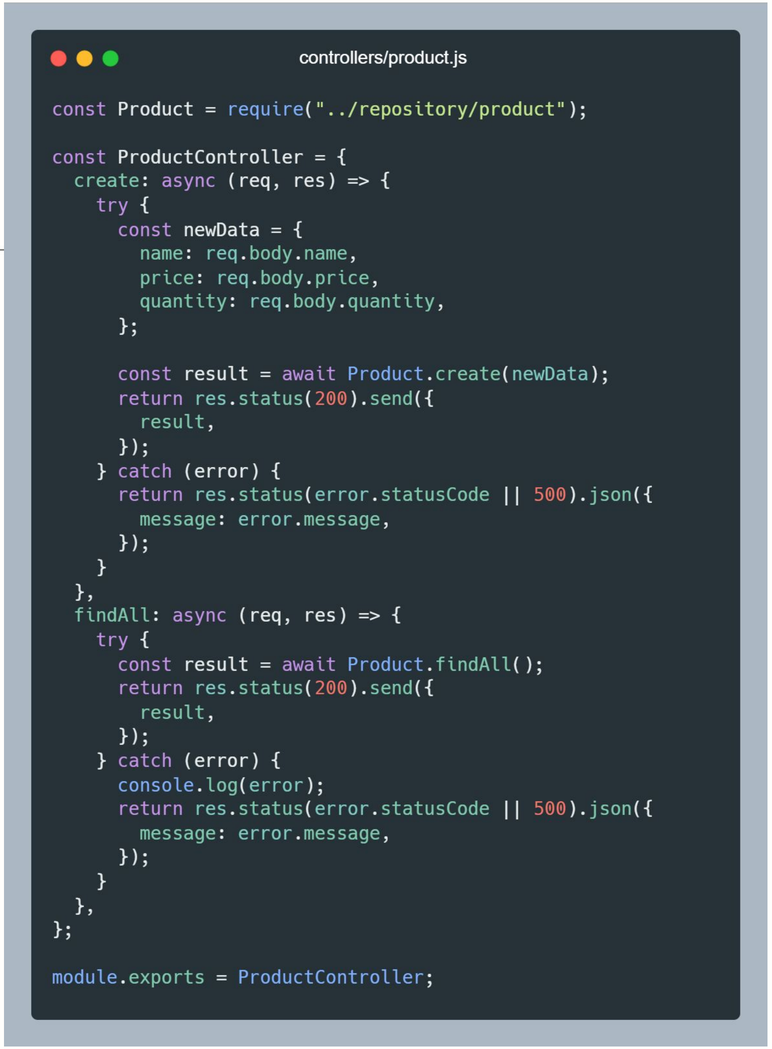

repository/product.js

```
const { Product } = require("../lib/sequelize");

const ProductRepository = {
  create: async (newData) => {
    try {
      const result = await Product.create(newData);
      return result;
    } catch (error) {
      throw error;
    }
  },
  findAll: async (req, res) => {
    try {
      const result = await Product.findAll();
      return result;
    } catch (error) {
      throw error;
    }
  },
};

module.exports = ProductRepository;
```

Mocking in Jest

Create controllers/product.js file

Write down this code!

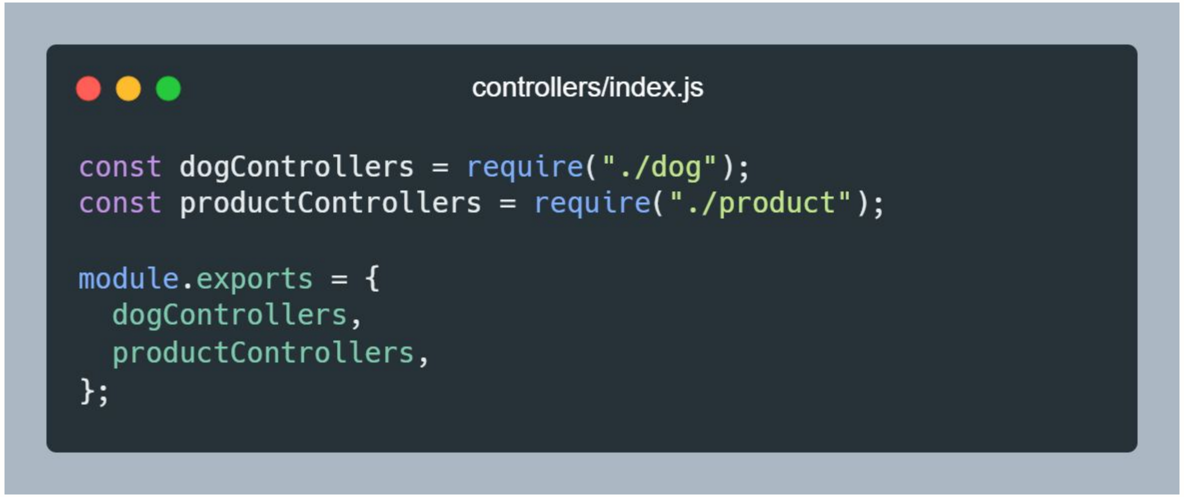

controllers/product.js

```
const Product = require("../repository/product");

const ProductController = {
  create: async (req, res) => {
    try {
      const newData = {
        name: req.body.name,
        price: req.body.price,
        quantity: req.body.quantity,
      };

      const result = await Product.create(newData);
      return res.status(200).send({
        result,
      });
    } catch (error) {
      return res.status(error.statusCode || 500).json({
        message: error.message,
      });
    }
  },
  findAll: async (req, res) => {
    try {
      const result = await Product.findAll();
      return res.status(200).send({
        result,
      });
    } catch (error) {
      console.log(error);
      return res.status(error.statusCode || 500).json({
        message: error.message,
      });
    }
  },
};

module.exports = ProductController;
```

Mocking in Jest

Modify controllers/index.js file

Write down this code!

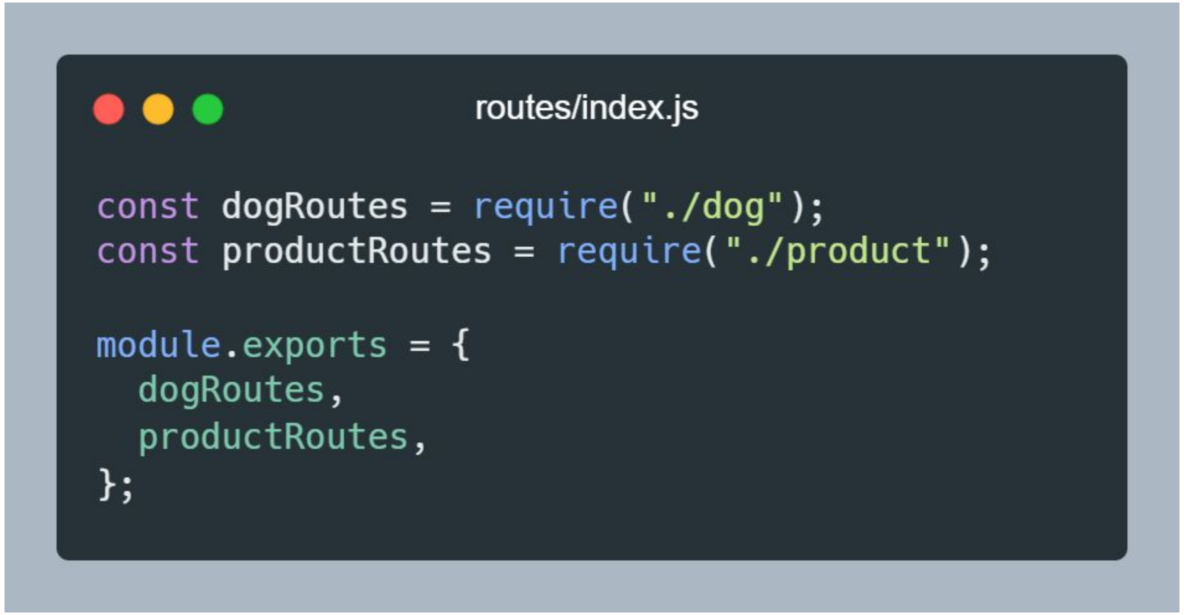

controllers/index.js

```
const dogControllers = require("./dog");
const productControllers = require("./product");

module.exports = {
  dogControllers,
  productControllers,
};
```

Mocking in Jest

Modify routes/index.js file

Write down this code!

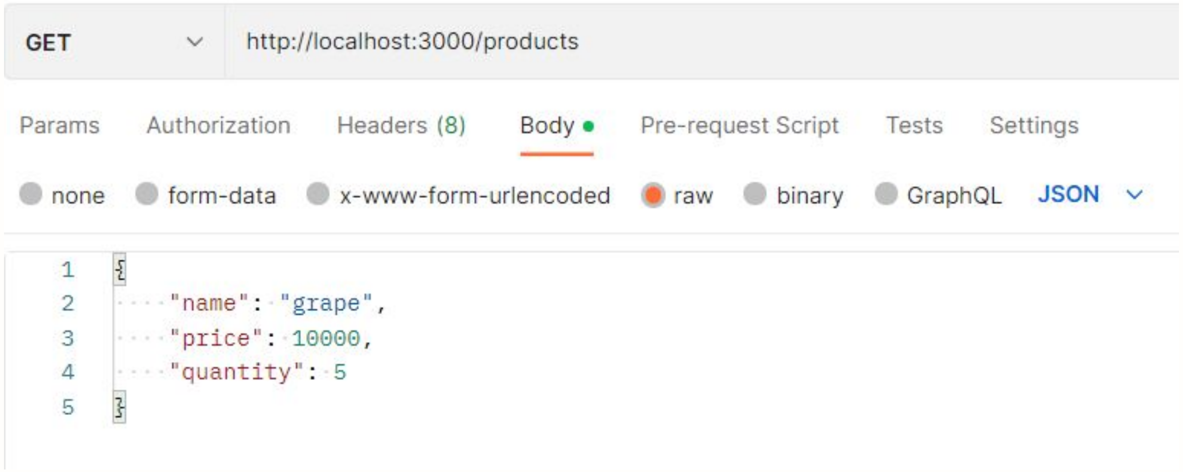


routes/index.js

```
const dogRoutes = require("./dog");
const productRoutes = require("./product");

module.exports = {
  dogRoutes,
  productRoutes,
};
```

Mocking in Jest

Lets create our unit test based on product action we just made. Before we create the unit test, please add several data into table product before we go to the next step.



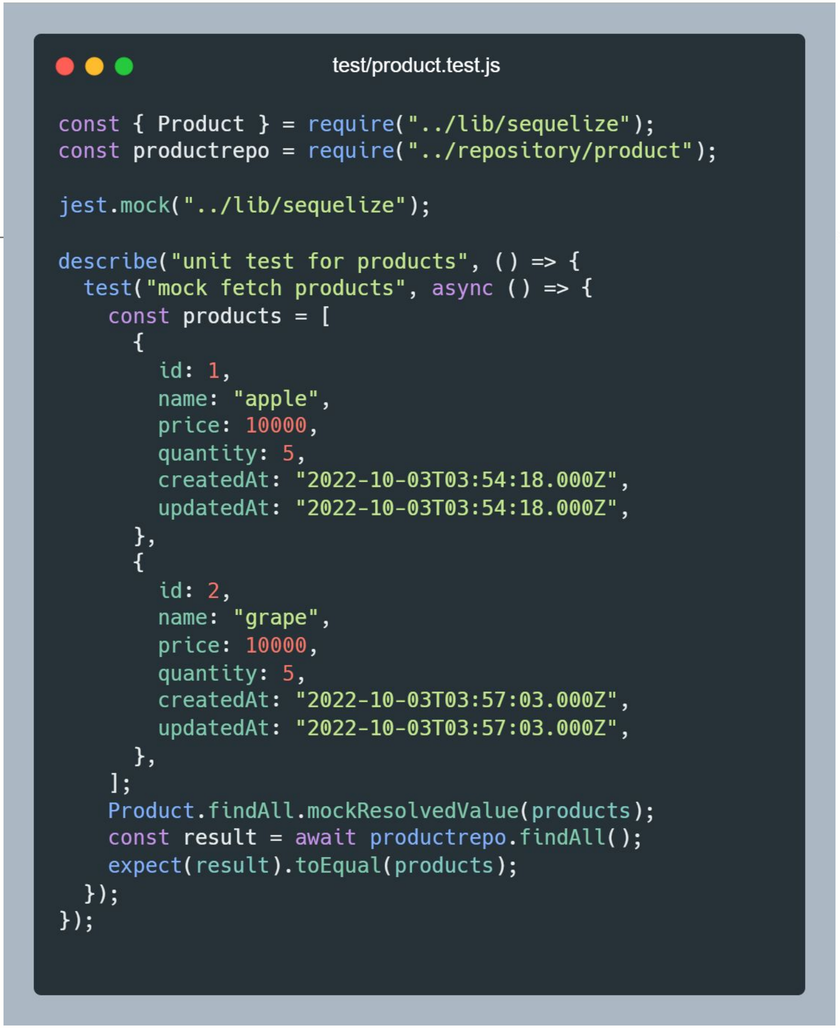
http://localhost:3000/products

```
{
    "name": "grape",
    "price": 10000,
    "quantity": 5
}
```

Mocking in Jest

Lets make a unit test!

This time we try to mock findAll method in product models.

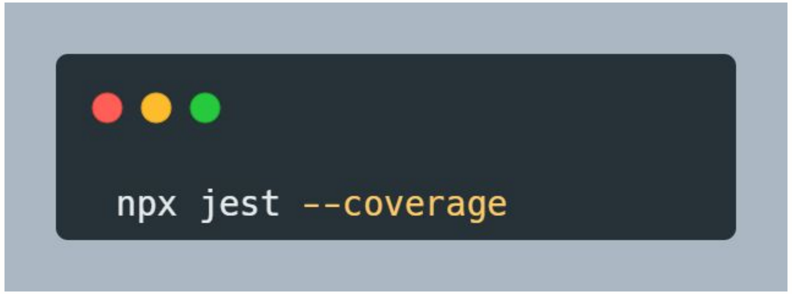


test/product.test.js

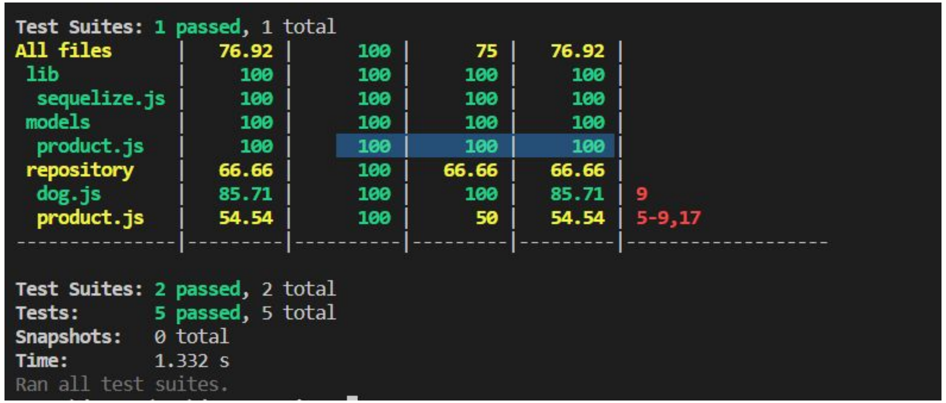
```javascript
const { Product } = require("../lib/sequelize");
const productrepo = require("../repository/product");

jest.mock("../lib/sequelize");

describe("unit test for products", () => {
  test("mock fetch products", async () => {
    const products = [
      {
        id: 1,
        name: "apple",
        price: 10000,
        quantity: 5,
        createdAt: "2022-10-03T03:54:18.000Z",
        updatedAt: "2022-10-03T03:54:18.000Z",
      },
      {
        id: 2,
        name: "grape",
        price: 10000,
        quantity: 5,
        createdAt: "2022-10-03T03:57:03.000Z",
        updatedAt: "2022-10-03T03:57:03.000Z",
      },
    ];
    Product.findAll.mockResolvedValue(products);
    const result = await productrepo.findAll();
    expect(result).toEqual(products);
  });
});
```

Mocking in Jest

Try to execute this command! See the result. Create another unit test based on not coverage code in your projects!



```
npx jest --coverage
```



```
Test Suites: 1 passed, 1 total
All files      |   76.92 |      100 |      75 |   76.92 |
 lib           |     100 |      100 |     100 |     100 |
  sequelize.js |     100 |      100 |     100 |     100 |
 models        |     100 |      100 |     100 |     100 |
  product.js   |     100 |      100 |     100 |     100 |
 repository    |   66.66 |      100 |   66.66 |   66.66 |
  dog.js       |   85.71 |      100 |     100 |   85.71 | 9
  product.js   |   54.54 |      100 |      50 |   54.54 | 5-9,17
---------------|---------|----------|---------|---------|-------------------

Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        1.332 s
Ran all test suites.
```

Thank You!

Purwadhika
Digital Technology School