**Exercise03\_04\_01 – Step 1**

In this Exercise, we will transition to ES6 in Node.js.



1. Create a folder named ***Exercise03\_04\_01***. Copy in all of the contents from ***Exercise03\_03\_02***. Open your IDE to the project folder.
2. Go to the ***/src/lib*** folder and open up the file ***dataStore.js***. The ***initPizzas()*** function uses a ***forEach()*** construct. This construct has always seemed overly complex, compared to other languages. Let’s implement a ***for…of*** to replace it:  
    ***for(const pizza of pizzas) {*** realPizzas[pizza[0]] = new Pizza(startingDate,   
    data.quotes[pizza[0]], ...pizza);  
    ***};***Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.
3. Go to the ***/src/models*** folder and open up the file ***pizza.js***. The ***getDatedQuotes()*** function uses a ***forEach()*** construct. Let’s implement a ***for…of*** to replace it:  
    ***for(const quote of this.quotes) {*** quotesMap[curDate] = quote;  
    curDate.setDate(curDate.getDate() + 1);  
    ***}***Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.

**Exercise03\_04\_01 – Step 2**



1. In the ***/src*** folder create a new folder called ***/generators***. Create a new file in this folder called ***mainContent.js***. The purpose of this file will be to create a ***context*** object to pass to our template. We have done this one time before in ***/src/handlers/main.js*** with a ***Promise.all()***, but now we will use generators. Let’s scaffold out a generator function called ***mainContent()***:  
   ***function\* mainContent() {  
      
   }***
2. Now let’s build our context object dynamically with the generator. We can make each of its properties be set to ***yield***, which will be the value ***passed*** ***into*** ***it*** by a ***.next*** function call. And finally we will ***return*** the context object. We will also need to export the generator function:  
   function\* mainContent() {  
    ***const context = {  
    popSlices: yield,  
    mostPopular: yield,  
    newestSlice: yield,  
    mostImproved: yield,  
    pizzas: yield  
    };  
    return context;***}  
    ***module.exports = mainContent;***
3. Go to the ***/src/handlers*** folder and open up the file ***main.js***. Firstly, we need to ***require()*** our new file to get the generator function:  
   var popGen = require('../lib/popGen'),  
    dataStore = require('../lib/dataStore'), ***mainContent = require('../generators/mainContent');***Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.
4. Now let’s get our ***iterator*** for the generator function. And we can start the generator function with a next(). This starts the first function:  
   module.exports = function (request, reply) {  
    ***const gen = mainContent();  
    gen.next();***
5. Now let’s convert the ***Promise.all()*** into code that uses our generator. Let’s call our first Promise function and pass the result from the promise into the generator with a ***next()*** call. This will populate our first ***yield*** in the generator:   
    gen.next();  
    ***popGen.getPopularSlices()  
    .then((popSlices) => {  
    gen.next(popSlices);  
    })***
6. Now let’s ***chain*** these Promises together. We add the call to our second promise function into the .then as a ***return***, which will allow us to chain to a following ***.then*** function:  
    .then((popSlices) => {  
    gen.next(popSlices);  
    ***return popGen.getMostPopular();*** })
7. Now let’s do the chain to the following ***.then***: which will take care of our second ***yield*** and allow us to chain further:  
    .then((popSlices) => {  
    gen.next(popSlices);  
    return popGen.getMostPopular();  
    })  
    ***.then((mostPopular) => {  
    gen.next(mostPopular);  
    return popGen.getNewestSlice();  
    })***
8. Now let’s complete the next two of the ***promise*** and ***generator*** ***chain***:  
    ***.then((newestSlice) => {  
    gen.next(newestSlice);  
    return popGen.getMostImproved();  
    })  
    .then((mostImproved) => {  
    gen.next(mostImproved);  
    return dataStore.getPizzas();  
    })***
9. The last ***.then*** will be a little different. It will complete our ***context*** object, just as we completed it with the final ***return*** of our ***Promise.all()***. So we can cut out the final ***return*** of our ***Promise.all()*** and paste into a last chained ***.then*** as follows:  
    ***.then((pizzas) => {  
    return reply.view('index', context);  
    })***
10. Now let’s change the ***content*** ***parameter*** to be the result of our last ***.next()*** call. This will provide the value for our last ***yield***, the generator function will complete with no more ***yield*** statements. The ***done*** property will be ***true***, and the ***value*** property will hold our completed ***context*** object:  
     .then((pizzas) => {  
     ***return reply.view('index', gen.next(pizzas).value);*** })
11. Now we can delete all of our ***promises*** array and ***Promise.all()*** code().  
    Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.

**Exercise03\_04\_01 – Step 3**



1. In the ***/generators*** folder, create a new file in this folder called ***itrObject.js***. Let’s scaffold out a generator function called ***itrObject()*** that has afuc parameter that will be an object:  
   ***function\* itrObject(obj) {  
      
   }***
2. Now let’s build our generator out. It will use a ***for…in*** inside itself, which will ***iterate*** over an ***object*** and return ***true*** if a property is part of the object itself and not a property of its parent ***prototype***. Notice the shorthand syntax for the ***key*** property of the yield object:  
   function\* itrObject(obj) {  
    ***for (const key in obj) {  
    if (obj.hasOwnProperty(key)) {  
    yield {  
    key,  
    val: obj[key]  
    };   
    }  
    }***}  
     
   module.exports = itrObject;
3. Go to the ***/src/lib*** folder and open up the file ***popGen.js***. Firstly, we need to ***require()*** our new file to get the generator function:  
   var api = require('./api'),  
    \_ = require('lodash'),  
    ***itrObject = require('../generators/itrObject');***Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.
4. Scroll down to the ***getMostImproved()*** function. There is a ***for…in*** construct which can be cleaned up and modified to use our generator. Let’s change it as follows to use a ***for…of*** which uses our ***itrObject()*** generator:  
    ***for (const prop of itrObject(allQuotes)) {***
5. We can now modify the ***diffQuotes.push()*** call to use the results of the ***yield*** returns from our generator as the loop iterates:   
    for (const prop of itrObject(allQuotes)) {  
    diffQuotes.push({  
    ***ticker: prop.key,  
    diff: prop.val[prop.val.length - 1] - prop.val[0],  
    quote: prop.val[prop.val.length - 1]*** });  
   Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.