**Exercise03\_03\_01 – Step 1**

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



1. Create a folder named ***Exercise03\_03\_01***. Copy in all of the contents from ***Exercise03\_02\_01***. Open your IDE to the project folder.
2. Go to the ***/src/lib*** folder and open up the file ***quoteManager.js***, which exports a function that uses the ***callback*** pattern. It accepts a callback function as an argument, and calls it with the data when complete. We will update this to make use of ***promises***. First, we remove the ***callback*** argument:  
    ***updateQuotes () {***
3. Directly below that we will add a statement to ***return*** a newly constructed ***Promise*** object. It will take two arguments, ***resolve*** and ***reject***:  
    updateQuotes() {  
    ***return new Promise((resolve, reject) => {  
     
    });***
4. Move the rest of the asynchronous code into the new constructor:  
   module.exports = {  
    updateQuotes() {  
    return new Promise((resolve, reject) => {  
    ***api.getAllPizzas(function (err, pizzas) {  
    var newData = [], pizza;  
     
    for (var key in pizzas) {  
    pizza = pizzas[key];  
    newData.push({  
    ticker: pizza.ticker,  
    nextQuote: pizza.getNext()  
    });  
    }  
     
    console.log(`${JSON.stringify(newData)}  
    updating quote`);  
    callback(null, newData);  
    })***;  
    });  
    }  
   };
5. Now we need to return the data correctly. We can replace the callback, which never actually handles an ***error***, with a promise ***resolve()*** call:  
    console.log(`${JSON.stringify(newData)}updating quotes`);  
    ***resolve(newData);***
6. Now we can properly handle the possibility of an error, with a promise ***reject()*** call:  
    var newData = [],  
    pizza;  
    ***if (err) {  
    reject(err);  
    }***
7. Let’s properly structure the rest of the function code by putting it into an else block:  
    if (err) {  
    reject(err);  
    }  
    ***else {  
    for (var key in pizzas) {  
    pizza = pizzas[key];  
    newData.push({  
    ticker: pizza.ticker,  
    nextQuote: pizza.getNext()  
    });  
    }  
     
    console.log(`${JSON.stringify(newData)}  
    updating quote`);  
    resolve(newData);  
    }***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. We have no syntax errors, our console.log() debug looks good, but we are no longer getting correct screen behavior. We will have to fix all of the places that use the function we modified.

**Exercise03\_03\_01 – Step 2**



1. Go to the ***/src/lib*** folder and open up the file ***market.js***. There is one call to the ***updateQuotes()*** function that we modified. It is passing a ***callback*** function, which will no longer work. First, we need to modify the call to take no ***parameters*** by modifying the parentheses to be empty. Then we move the function declaration to the next line:  
    ***quoteManager.updateQuotes()  
    function (err, newData) {*** socket.emit('new\_data', JSON.stringify(newData));  
    });
2. We can now convert the anonymous function stub to a chain to a promise ***.then*** call, removing the previously unused ***err*** parameter. We turn the remaining parameter into an ***arrow*** ***function*** to execute on the ***.then*** call as follows:  
    quoteManager.updateQuotes()  
    ***.then((newData) => {*** socket.emit('new\_data', JSON.stringify(newData));  
    });
3. We can now extend the code structure and chain in a ***.catch*** to handle the ***error*** as follows:  
    .then((newData) => {  
    socket.emit('new\_data', JSON.stringify(newData));  
    })  
    ***.catch((err) => {  
    console.error(err);   
    });***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. Everything should be working properly again.

**Exercise03\_03\_01 – Step 3**



1. Let’s practice some more conversions to promises. Go to the ***/src/lib*** folder and open up the file ***api.js***. Let’s convert the ***getPizza()*** function. First we will remove the ***callback*** argument to the function. We open up a new line and build our return with a scaffolded Promise constructor:  
   ***function getPizza(ticker) {  
    return new Promise((resolve, reject) => {  
      
    });***
2. We should take the rest of the asynchronous code in the function, and move it inside our new constructor:  
    return new Promise((resolve, reject) => {  
    ***request('http://localhost:' + localPort + '/pizza/' + ticker,   
    function (error, res, body) {  
    if (callback) {  
    callback(error, Pizza.hydrate(JSON.parse(body)));  
    }  
    });*** });
3. Now we can repurpose the ***if (callback)*** line to ***reject()*** our ***error*** condition. And we can repurpose the ***callback()*** call to ***resolve*** our ***success*** condition. Make the following changes:  
    request('http://localhost:' + localPort + '/pizza/' + ticker, function   
    (error, res, body) {  
    ***if (error) {  
    reject(error);  
    }  
    else {  
    resolve(Pizza.hydrate(JSON.parse(body)));  
    }*** });
4. Now we have to make changes to whatever calls the modified function. In the ***/src/lib*** folder and open up the file ***popGen.js***. The ***getNewestSlice()*** function uses a call to ***getPizza()***. Close the parameter list directly after the ***’HAWA’*** string and move the function to the next line. Repurpose the function call to a ***chain*** to a ***.then*** call as follows:  
   ***function getNewestSlice() {***  
    ***api.getPizza('HAWA')  
    .then((pizza) => {  
    if (callback) {  
    callback(null, { ticker: 'HAWA', quote: pizza.getLast() });  
    }  
    });***}
5. Now we can handle our error situation by chaining in a ***.catch*** clause as follows:  
    .then((pizza) => {  
    if (callback) {  
    callback(null, { ticker: 'HAWA', quote: pizza.getLast() });  
    }  
    })  
    ***.catch((err) => {  
    callback(err);  
    });***  
   Notice that we are still using a ***callback*** pattern for the outer function. We don’t have to change everything. Promises and callbacks can ***co-exist*** peacefully. We use promises where they make sense, or where there is really gnarly callback 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. Everything should be working properly.

**Exercise03\_03\_01 – Step 4**



1. Go to the ***/src/handlers*** folder and open up the file ***main.js***. As we can see, we have a serious case of Callback Hell, a bunch of ***nested*** function calls with callbacks. We can use a ***Promise.all()*** construct to take care of this. I am going to ***beautify*** this file so it formats to the settings of my IDE. First we need to get all of our ***Promise*** objects into an ***array***. We can call all of the functions, expecting they will each return a Promise, as follows:  
   module.exports = function (request, reply) {  
    var context = {};  
      
    ***const promises = [  
    popGen.getPopularSlices(),  
    popGen.getMostPopular(),  
    popGen.getNewestSlice(),  
    popGen.getMostImproved(),  
    dataStore.getPizzas()  
    ];***
2. Directly below our ***promises*** array, let’s scaffold out our ***Promise.all()*** structure, passing it our array. And let’s scaffold in our ***.then*** structure which gets the ***results*** array of all the promises:   
    ***Promise.all(promises)  
    .then((results) => {  
      
    });***
3. Notice that in our nested ***callback*** section, that the last of the function calls does a ***return*** which is basically rendering the page. Let’s copy that and place it in the body of the arrow function that is the parameter our new ***.then*** call. It is using the ***context*** object, which is declared at function scope. Let’s cut that out and put it where it is used, just above our return statement. We will also change it to a ***const***:  
    Promise.all(promises)  
    .then((results) => {  
    ***const context = {};  
    return reply.view('index', context);*** });
4. We can now build our ***context*** object with the contents of the ***results*** array. We can refer to the function calls below and see which properties of the object are set, and the results array will hold the values in the order of our original promises array:  
    .then((results) => {  
    const context = {  
    ***popSlices: results[0],  
    mostPopular: results[1],  
    newestSlice: results[2],  
    mostImproved: results[3],  
    pizzas: results[4]*** };
5. Now let’s handle the error scenario. We can do that by chaining in a ***.catch***:  
    })  
    ***.catch((err) => {  
    console.error(err);  
    })***;  
   Now we can delete or comment out all the old callback code. A much cleaner and readable implementation.
6. Now we have to make changes to whatever calls these functions. In the ***/src/lib*** folder and open up the file ***popGen.js*** and beautify if you want to. Start with the ***getPopularSlices ()*** function has a callback.  
     
   Notice the strange references to ***\_.orderBy()*** and ***\_.take()***,references to methods of the ***lodash*** utility library, a ***dependency*** in package.json and ***required*** into popGen.js. You can Google this library. The ***\_.orderBy()*** creates an array of elements with a specified sort. In this case by the quote field descending. The ***\_.take()*** creates a slice of array with ***n*** elements taken from the beginning, in this case 4 elements. Modify as follows:  
   ***function getPopularSlices() {  
    return new Promise((resolve, reject) => {  
    \_getFinalQuotes()  
    .then((finalQuotes) => {  
    const orderedQuotes = \_.orderBy(finalQuotes,  
    ['quote'], ['desc']);  
    resolve(\_.take(orderedQuotes, 4));  
    })  
    .catch(reject);  
    });  
   }***Run popGen.js with a Brackets ***alt-n*** to test for syntax errors.
7. We will continue in ***popGen.js*** with the ***getMostPopular()*** function. Modify as follows:  
   ***function getMostPopular () {  
    return new Promise((resolve, reject) => {  
    \_getFinalQuotes()  
    .then((finalQuotes) => {  
    const mostPopular = finalQuotes .reduce(function (best, curr) {  
    if (curr.quote > best.quote) {  
    return curr;  
    }  
    return best;  
    }, { quote: 0 });  
    resolve(mostPopular);  
    })  
    .catch(reject);  
    });  
   }***Run popGen.js with a Brackets ***alt-n*** to test for syntax errors.
8. We will continue in ***popGen.js*** with the ***getNewestSlice()*** function, which we have already modified once. Finish the job as follows:  
   ***function getNewestSlice() {  
    return new Promise((resolve, reject) => {  
    api.getPizza('HAWA')  
    .then((pizza) => {  
    resolve({  
    ticker: 'HAWA',  
    quote: pizza.getLast()  
    });  
    })  
    .catch((err) => {  
    reject(err);  
    });  
    });  
   }***Run popGen.js with a Brackets ***alt-n*** to test for syntax errors.
9. We will continue in ***popGen.js*** with the ***getMostImproved()*** function. Modify as follows:  
   ***function getMostImproved () {  
    return new Promise((resolve, reject) => {  
    api.getAllQuotes()  
    .then((allQuotes) => {  
    const diffQuotes = [];  
    for (const key in allQuotes) {  
    diffQuotes.push({  
    ticker: key,  
    diff: allQuotes[key][allQuotes[key].length - 1] -   
    allQuotes[key][0],  
    quote: allQuotes[key][allQuotes[key].length - 1]  
    });  
    }  
     
    const mostImproved = diffQuotes.reduce(function (best, curr) {  
    if (curr.diff > best.diff) {  
    return curr;  
    }  
    return best;  
    }, { diff: 0});  
    resolve(mostImproved);  
    })  
    .catch(reject);  
    });  
   }***Run popGen.js with a Brackets ***alt-n*** to test for syntax errors.
10. The previous work modified the call to ***getAllQuotes()***. We now must go into ***api.js*** and modify it from callback to promise:  
    ***function getAllQuotes() {  
     return new Promise((resolve, reject) => {  
     request('http://localhost:' + localPort + '/quotes',   
     function (error, res, body) {  
     if (error) {  
     reject(error);  
     } else {  
     resolve(JSON.parse(body));  
     }  
     });  
     });  
    }***  
    Run apis.js with a Brackets ***alt-n*** to test for syntax errors.
11. While we are in ***api.js*** modify ***getAllPizzas()*** from callback to promise:  
    ***function getAllPizzas() {  
     return new Promise((resolve, reject) => {  
     request('http://localhost:' + localPort + '/pizzas', function   
     (error, res, body) {  
     if (error) {  
     reject(error);  
     } else {  
     const staticPizzas = JSON.parse(body),  
     pizzas = [];  
      
     for (var ix in staticPizzas) {  
     pizzas.push(Pizza.hydrate(staticPizzas[ix]));  
     }  
     resolve(pizzas);  
     }  
     });  
     });  
    }***  
    Run apis.js with a Brackets ***alt-n*** to test for syntax errors.
12. The previous work modified the call to ***getAllQuotes()***. We now must go into ***datastore.js*** and modify it from callback to promise:  
    ***function getAllQuotes() {  
     return new Promise((resolve) => {  
     resolve(data.quotes);  
     });  
    }***Run datastore.js with a Brackets ***alt-n*** to test for syntax errors.
13. In ***datastore.js*** we must now modify ***getPizzas()***, the last call in our ***promises*** array:  
    ***function getPizzas() {  
     return new Promise((resolve) => {  
     resolve(data.pizzas);  
     });  
    }***Run datastore.js with a Brackets ***alt-n*** to test for syntax errors.
14. The previous work modified the call to ***getAllPizzas()***. We now must go into ***quoteManager.js*** and modify it from callback to promise:  
    ***module.exports = {  
     updateQuotes() {  
     return new Promise((resolve, reject) => {  
     api.getAllPizzas()  
     .then((pizzas) => {  
     const newData = [];  
     let pizza;  
     for (const key in pizzas) {  
     pizza = pizzas[key];  
     newData.push({  
     ticker: pizza.ticker,  
     nextQuote: pizza.getNext()  
     });  
     }  
     console.log(`${JSON.stringify(newData)} updating   
     quotes`);  
     resolve(newData);  
     })  
     .catch(reject);  
     });  
     }  
    };***Run apis.js with a Brackets ***alt-n*** to test for syntax errors.
15. The previous work modified the call to \_***getFinalQuotes()*** in ***popGen.js*** go back to it and modify it from callback to promise:  
    ***function \_getFinalQuotes () {  
     return new Promise((resolve, reject) => {  
     api.getAllQuotes()  
     .then((allQuotes) => {  
     const finalQuotes = [];  
     for (const key in allQuotes) {  
     finalQuotes.push({  
     ticker: key,  
     quote: allQuotes[key][allQuotes[key].length - 1],  
     diffLast: \_percentOf(allQuotes[key][allQuotes[key]  
     .length - 2], allQuotes[key][allQuotes[key].length - 1])  
     });  
     }  
     resolve(finalQuotes);  
     })  
     .catch(reject);  
     });  
    }***Run apis.js with a Brackets ***alt-n*** to test for syntax errors.  
      
    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. Everything should be working properly.
16. The previous work modified ***dataStore.getPizza()***. Go to ***/src/handlers/pizza.js*** it and modify it from callback to promise:  
     ***dataStore.getPizza(ticker)  
     .then(reply)  
     .catch(reply)  
    };***Run it with a Brackets ***alt-n*** to test for syntax errors. 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. Everything should be working properly.

**Exercise03\_03\_01 – Step 5**



1. Go to the ***/src/lib*** folder and open up the file ***popGen.js***. Find the ***getFinalQuotes()*** function call. It currently uses an array called ***finalQuotes***, which we can do a better implementation with as a ***Set***. Replace the array declaration with a Set object ***constructor***:  
   module.exports = function (request, reply) {  
    ***const finalQuotes = new Set();***
2. Inside the for loop, we merely need to change out the ***array.push()*** call for a ***Set.add()*** call:   
    for (const key in allQuotes) {  
    ***finalQuotes.add({***
3. The ***getMostPopular()*** function uses the ***finalQuotes*** array. It is using an ***array.reduce()*** call, which won’t work on a Set. The simplest solution is to use ***spread*** syntax to turn the set into an array:  
    ***const mostPopular = [...finalQuotes].reduce(function (best,   
    curr) {***
4. Another place the finalQuotes array is used is in the ***getPopularSlices()*** function. We can use the same spread syntax technique to fix it:  
    ***const orderedQuotes = \_.orderBy([...finalQuotes], ['quote'],   
    ['desc']);***  
   The advantage of using the set is that it enforces unique value rules, which are actually needed in this situation. 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. Everything should be working properly again.

**Exercise03\_03\_01 – Step 6**



1. Go to the ***/src/lib*** folder and open up the file ***dataStore.js***. It currently uses an object called ***data***, which it actually uses like a ***Map***. Replace the array declaration with a Map object ***constructor***:  
   ***const data = new Map();***
2. Now let’s convert the ***data.quotes*** property assignment to Map syntax:  
    ***data.set('quotes', require('../mock/quotes'));***
3. Now let’s convert the ***data.pizzas*** property assignment to Map syntax:  
    ***data.set('pizzas', pizzas);***
4. Now let’s modify the code that ***retrieves*** values from our Map. Let’s modify the ***return*** statement in ***getQuotes()***:  
    ***return data.get('quotes')[ticker];***
5. Next, let’s modify the ***data.quotes*** statement in ***getAllQuotes()***:  
    ***callback(null, data.get('quotes'));***
6. Next, let’s modify the ***data.pizzas*** statement in ***getPizzas()***:  
    ***callback(null, data.get('pizzas'));***
7. Next, let’s modify the ***data.pizzas*** statement in ***getPizza()***:  
    ***callback(null, data.get('pizzas')[ticker]);***
8. Lastly, let’s modify the ***data.quotes*** statement in ***initPizzas()***:  
    ***realPizzas[pizza[0]] = new Pizza(startingDate,   
    data.get('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. Everything should be working properly again.