Handling Errors

Comment all codes related to geocoding.
The first possible problem is that there may not be a network to connect.
To test it, temporarily disable the wifi and test the script

node app.js

We see the error message, "cannot read property body of undefined"
When there is a problem, error argument contains value and response does not.
Response is currently undefined. We cannot fix the network problem but we can display a better error message.

console.log(error)
Comment the other console.log comment causing the error.

Change as follows

```
if (error) {
        console.log('Unable to connect to weather service!')
} else {
        .. //uncomment the log below and cut it to here.
}
```

Rerun the app.
Node app.js

Reenable the wifi and test the app again.
Node app.js

If there is a value in error, there is no value in response. And vice versa.

Lets test if the user provides bad input.
Copy the url and paste it to chrome.

Now mess up the input to generate an error. Remove the latitude and longitude from the url.
Check the response/error.

The servers responds with an error message. But it comes in the response body.

```
if (error) {
        console.log('Unable to connect to weather service!')
} else if (response.body.error) {
        console.log('Unable to find location')
} else {
        .. //uncomment the log below and cut it to here.
}
```

Remove lat and lon from the url and test the application.
Node app.js

Now comment all the code for the weather service and uncomment the code for the geocoding service.

Setup error handling for the geocoding service.

Copy the url and paste it to chrome. Change from California to some random numbers like 12what and see the response. Features array is now empty.

Turn off wifi and run the app.

```
if (error) {
        console.log('Unable to connect to location services!')
} else {
        .. //uncomment the log below and cut it to here.
}
```

Run the script now.
Reenable network connection.
Rerun the application.

Now mess-up the search script.

```
if (error) {
        console.log('Unable to connect to location services!')
} else if (response.body.features.length === 0) {
        console.log('Unable to find location. Try another search.')
} else {
        .. //uncomment the log below and cut it to here.
}
```

Callback function

Go to playground folder and create a new file for tests callback.js
Add the following code

```
setTimeout( () => {
        console.log('Two seconds time out')
}, 2000)
```

This is a callback function.

```
() => {
        console.log('Two seconds time out')
}
```

Not all callback functions are asynchronous.

```
const names = ['Ali', 'Berk', 'Cengiz']
const shortNames = names.filter( (name) => {
        return name.length <= 4
})
```

We will return names with a length of 4 or less.
This is a synchronous callback function.

Lets assume that I have 4 and 5 different locations that I want to access a location. I can define a function to access the geocoding functionality.

```
const geocode = () => {

}
```

Settimeout takes a function and timeout as arguments. We take address and callback.

```
const geocode = (address, callback) => {
```

```
    const data = {
        latitude: 0,
        longitude: 0
    }
}
```

I could just return the data without a callback

```
const geocode = (address, callback) => {
    const data = {
        latitude: 0,
        longitude: 0
    }
    return data
}

Const data = geocode('Bursa')
console.log(data)
```

Go to the playground and test the code.
node callback.js

We see the results. But there is nothing asynchronous here. Later on we will add a request and it will be asynchronous. Lets simulate the delay for the request.

```
const geocode = (address, callback) => {
    setTimeout( () => {
        const data = {
        latitude: 0,
        longitude: 0
        }
        return data
    }, 2000)
}
```

Rerun the program. We see undefined as the result.
Return is part of the inner set timeout function. Geocode function does not have a return. Thats why we don't get a value back. Lets change our code to callback. Remember that callback runs after main finishes. Here we tried to get the value before the main finishes.

```
const geocode = (address, callback) => {
    setTimeout( () => {
        const data = {
        latitude: 0,
        longitude: 0
        }
        callback(data)
    }, 2000)
}

geocode('Bursa', (data) => {
    console.log(data)
})
```

Now rerun the code
Node app.js

We wait two seconds as expected and get the data.
We could use return if it was synchronous.

Comment all the code in the callback file.
Add the following code.
Define the add function with the correct arguments.
Set timeout to simulate 2 seconds delay.

```
add(1, 4, (sum) => {
        console.log(sum)
})

const add = (a, b, callback) => {
        setTimeout( () => {
                //we cannot write return here. Instead call callback
                callback(a + b)
        }, 2000)
}
```

Test the code. After 2 seconds, it prints 5.

Now we are going to create an http request inside a function
If I need the weather forecast for 5 different locations, instead of copying the code we can create a function for this service. It will also be easier to do something in order. We want to geocode first and then use its output in the weather forecast function. Without using functions, complexity will increase and the reusability decreases.

Comment weather forecast request and also comment the geocoding request.
First, we will create a function. Then call it.

```
const geocode = (address, callback) => {

}

geocode('Bursa', (error, data) => {

})
```

Only one of error and data will have a value. This is optional but highly recommended.

```
const geocode = (address, callback) => {
        const url = 'https://api.mapbox.com/geocoding/v5/mapbox.places/' + encodeURIComponent(address)
+
'.json?access_token=pk.eyJ1IjoiaWZzNSIsImEiOiJja215YnU1NHIwMmU4MnVvMnl0dnl2YndjIn0.ilitdw0fICDP
jInjP0KI7g&limit=1' '

        request({ url: url, json: true}, (error, response) => {
                if (error) {
                        //instead of logging to the console, lets return it so that it can be sent as email or save to a
log file.
                        callback('Unable to connect to location services', undefined)
                } else if (response.body.length === 0) {
                        callback('Unable to find location. Try another search.', undefined)
                }
        })
}
```

Encoding will only be useful when the address contains special characters that have a meaning in urls like ?.

Lets try now.

```
geocode('Bursa', (error, data) => {
	console.log('Error', error)
	console.log('Data', data)
})
```

Disable internet connection and run the app.
node app.js

See the error. Enable internet connection. Search for a bogus address and see the result by running the app.

```
const geocode = (address, callback) => {
	const url = 'https://api.mapbox.com/geocoding/v5/mapbox.places/' + encodeURIComponent(address)
+
'.json?access_token=pk.eyJ1IjoiaWZzNSIsImEiOiJja215YnU1NHIwMmU4MnVvMnl0dnl2YndjIn0.ilitdw0fICDP
jlnjP0Kl7g&limit=1' '

	request({ url: url, json: true}, (error, response) => {
		if (error) {
			//instead of logging to the console, lets return it so that it can be sent as email or save to a
log file.
			callback('Unable to connect to location services', undefined)
		} else if (response.body.length === 0) {
			callback('Unable to find location. Try another search.', undefined)
		} else {
			callback(undefined, {
				latitude: response.body.features[0].center[0]
				longitude: response.body.features[0].center[1]
				location: response.body.features[0].place_name
			})
		}
	})
}
```

Try again.

```
geocode('Bursa', (error, data) => {
	console.log('Error', error)
	console.log('Data', data)
})
```

Run the app.

Create a new folder in weather-app named utils. In this folder, create a new file named geocode.js
Cut the geocode function from the app.js and paste it to the geocode.js

Add require to request at the top of the file

```
const request = require('request')
```

Also add exports at the end of the file.

module.exports = geocode

Go to app.js
Add the require at the top.
const geocode = require('./utils/geocode')

Change the location name and test the app again.

Lets do the same things for the weather forecast.
First remove the commented geocode code.
Add the call for forecast function. Comment the geocode function call.

```
const request = require('request')
const geocode = require('./utils/geocode')

//geocode('Bursa', (error, data) => {
//	console.log('Error', error)
//	console.log('Data', data)
//})

forecast(-75.7088, 44.1545, (error, data) => {
	console.log('Error', error)
	console.log('Data', data)
})
```

In utils older create a new file named forecast.js
In the new file define the forecast function. Export it.

```
const forecast = () => {
}

module.exports = forecast
```

Add require at the top in the app.js
const forecast = require('utils/forecast')

Add the arguments and add the require.

```
const request = require('request')
const forecast = (latitude, longitude, callback) => {
	conts url =
'http://api.weatherstack.com/current?access_key=81c6957beda8a6484399ecabcfdd7eae&query=' + latitude
+ ',' + longitude + '&units=f'
	request( {url: url, json:true}, (error, response) => {
		if (error) {
			callback('Unable to connect to weather service', undefined)
		}
		else if (response.body.error) {
			callback('Unable to find location', undefined)
		} else {
			callback(undefined, 'Hava şu anda: ' + response.body.current.weather_descriptions[0] + '
Hava sıcaklığı şu anda: ' + response.body.current.temperature + ' derece ve hissedilen sıcaklık: ' +
response.body.current.feelslike + ' derece')
		}
```

```
        })
}

module.exports = forecast
```

Test the code. First disable internet connection and rerun the script.
Provide a bad coordinate next. Finally test the correct case.

Uncomment geocode function call in app.js and remove the commented function definitions.
Remove request require at the top.

Callback chaining

```
const geocode = require('./utils/geocode')
const forecast = require('utils/forecast')

geocode('Bursa', (error, data) => {
        console.log('Error', error)
        console.log('Data', data)

        forecast(-75.7088, 44.1545, (error, data) => {
                console.log('Error', error)
                console.log('Data', data)
        })
})
```

Remove static parts.

```
geocode('Bursa', (error, data) => {
        console.log('Error', error)
        console.log('Data', data)
        forecast(data.latitude, data.longitude, (error, data) => {
                console.log('Error', error)
                console.log('Data', data)
        })
})
```

Rerun the file.
Longitude should be first in the order! Use index 0 for it.

If an error in the geocode, do not call the forecast function.

```
geocode('Bursa', (error, data) => {
        if (error) {
                return console.log(error) // if an error, it will not continue to run.
        }

        forecast(data.latitude, data.longitude, (error, data) => {
                if (error) {
                        return console.log(error)
                }
                console.log('Data', data)
        })
})
```

Since both results share the same name data, inner function shadows the outer value. We can change the names instead.

```
geocode('Bursa', (error, data) => {
    if (error) {
        return console.log(error) // if an error, it will not continue to run.
    }

    forecast(data.latitude, data.longitude, (error, forecastData) => {
        if (error) {
            return console.log(error)
        }
        console.log(data.location)
        console.log(forecastData)
    })
})
```

Rerun the app.

Provide the location from the command line argument without using yargs.
Add the following at the beginning.

```
console.log(process.argv)
```

Run the code. The third argument is the argument we need.

Now define a variable for the address.

```
const address = process.argv[2]

geocode(address, (error, data) => {
    if (error) {
        return console.log(error) // if an error, it will not continue to run.
    }

    forecast(data.latitude, data.longitude, (error, data) => {
        if (error) {
            return console.log(error)
        }
        console.log('Data', data)
    })
})
```

Run node app.js Boston
Or node app.js "New York"

Check if address is provided.

```
const address = process.argv[2]

if (!address) {
    console.log('Please provide an address')
} else {
    //cut and paste the geocode function call here. Can also be used return here.
}
```

Rerun the code for some locations.