Jesus Ortega - Async Notes Part 3

Implementing Promise-Based API

When you implement a promise-based API, you'll be wrapping an asynchronous operation which might use events, or plain callbacks, or a message-passing model. You'll arrange for a **Promise** object to handle the success or failure of that operation properly.

Implementing an alarm() API

In the example below we're implementing a promise-based alarm API - alarm(). The arguments will be the name of a person to wake up and a delay in milliseconds to wait before waking the person up. After the delay, the function will send a 'wake up' message, including the name of the person.

Wrapping setTimeout():

We will use **setTimeout()** to implement our alarm() function. The **setTimeout()** API takes as arguments a callback function and a delay, given in milliseconds. When it is called it starts a timer set to the given delay, and when the time expires, it calls the given function.

Here is the code below of us calling setTimeout() with a callback function and a delay of 1000 milliseconds:

```
const output = document.querySelector('#output');
const button = document.querySelector('#set-alarm');

function setAlarm() {
  setTimeout(() => {
    output.textContent = 'Wake up!';
  }, 1000);
}

button.addEventListener('click', setAlarm);
```

The Promise() Constructor:

The alarm() function will return a **Promise** that is fulfilled when the timer expires. It will pass a "Wake up!" message into the **then()** handler, and will reject the promise if the caller supplies a negative delay value.

This executor function itself takes two arguments, both functions, and are conventionally called **resolve** and **reject**. In your executor implementation, call the underlying asynchronous function. If the asynchronous function succeeds, you call resolve, and if it fails, you call reject. If the executor function throws an error, reject is called automatically. You can pass a single parameter of any type into resolve and reject.

Here is some example code:

```
//with reject and resolve:
function alarm(person, delay) {
    return new Promise((resolve, reject) => {
      if (delay < 0) {
        throw new Error('Alarm delay must not be negative');
      }
      setTimeout(() => {
        resolve(`Wake up, ${person}!`);
      }, delay);
    });
  }
```

This will create and return a new Promise. In the executor for the promise:

1. Check that delay is not negative and throw an error if it is.
2. Call **setTimeout()**, passing a callback and delay. The callback is called when the timer expires, and in the callback we can resolve, passing in our ‘wake up’ message.

Calling **alarm()** and on the returned promise call then() and catch() to set handlers for promise **fulfillment** and **rejection**:

```
const name = document.querySelector('#name');
const delay = document.querySelector('#delay');
const button = document.querySelector('#set-alarm');
const output = document.querySelector('#output');

function alarm(person, delay) {
    return new Promise((resolve, reject) => {
      if (delay < 0) {
        throw new Error('Alarm delay must not be negative');
      }
      setTimeout(() => {
        resolve(`Wake up, ${person}!`);
      }, delay);
    });
}

button.addEventListener('click', () => {
    alarm(name.value, delay.value)
      .then((message) => output.textContent = message)
      .catch((error) => output.textContent = `Couldn't set alarm:
${error}`);
});
```

Now we can set different values for the name and delay.

Using async and await with alarm() API

Since alarm() returns a *Promise*, we can do everything with it that we could do with any other promise: promise chaining, Promise.all(), and **async** / **await**. The code for this below:

```
const name = document.querySelector('#name');
const delay = document.querySelector('#delay');
const button = document.querySelector('#set-alarm');
const output = document.querySelector('#output');

function alarm(person, delay) {
  return new Promise((resolve, reject) => {
    if (delay < 0) {
      throw new Error('Alarm delay must not be negative');
    }
    setTimeout(() => {
      resolve(`Wake up, ${person}!`);
    }, delay);
  });
}

button.addEventListener('click', async () => {
  try {
    const message = await alarm(name.value, delay.value);
    output.textContent = message;
  }
  catch (error) {
    output.textContent = `Couldn't set alarm: ${error}`;
  }
});
```