7) A refresher on promises

Promises represent number 1 way in which we will interact with mongo db. so here we will do quick refresher on promises. We will make a game using promises.

8)let’s Play a Game

We will make a game using vanilla ES5, so with no promises nothing like that whatsoever.

Then in next lecture we will refactor our code and use promises.

So game is this-

Whenever user refreshes the page, game stars.if user clicks the button atleast 6 times in 2 seconds, game stops.so user got 2 seconds to click 6 times. If we does it then user wins otherwise he is lost. We can refresh the browser to start game again. Here is our code-

<head></head>

<body>

<button>Click!</button>

<script>

let counter = 0;

document.querySelector('button').addEventListener('click', () => {

++counter;

});

setTimeout(() => {

if (conter > 5) {

alert('you won');

}

else {

alert('You Lost');

}

}, 2000);

</script>

</body>

This works fine. But we have to write a lot of code in setTimeOut function. I will be really nice if we can wrap up our entire game as a standalone library os sorts. So ,may be I can publish this as a reusable package or something like that so that other developers can use it., in really nice fashion. All they have to do is to call a function called startGame and may be once the call startGame, they a nice interface to write some logic to handle whenever a user wins or loses. O our code is good but we can do better.

9)Winning a Game

In last lecture we discussed what we want to do, we want to publish our code as libray. Other developer can run game and then we need to allow them to specify what to do when user wins or loses. Now we write our code using promises like this-

<head></head>

<body>

<button>Click!</button>

<script>

function startGame() {

let counter = 0;

document.querySelector('button').addEventListener('click', () => {

++counter;

});

return new Promise((resolve, reject) => {

setTimeout( ()=> {

if (counter> 5) {

resolve();

}

else {

reject();

}

}, 2000);

});

}

startGame()

.then(() => alert('You Won'))

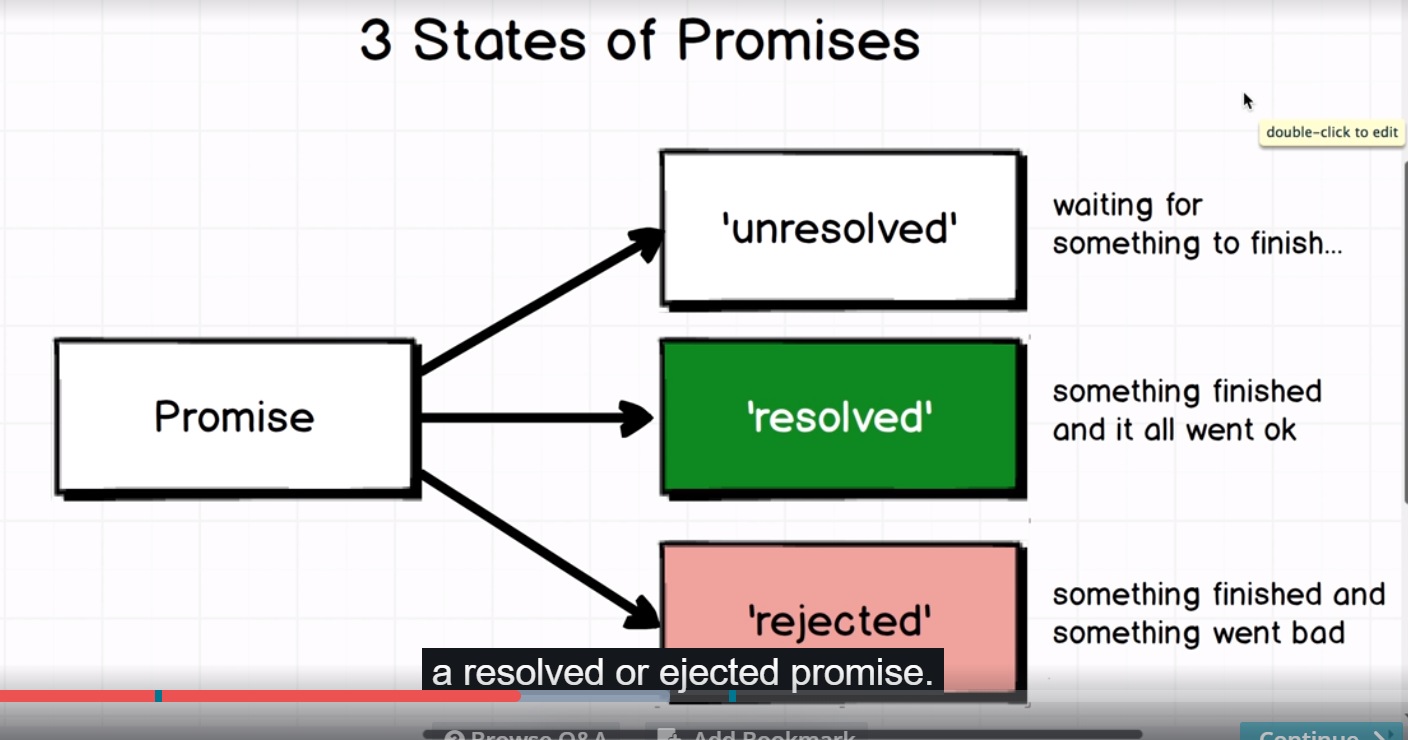
.catch(() =>alert('You Lost'));

</script>

</body>

10)Behind the scenes with promises

Promise is a tool for handling any code that is going to execute at some point in future. State of promise. Promise can be in 3 states-



It is upto us to decide what constitutes resolved and rejected promise. In our case case we changed resolved the promise if user clicked on button for more than 6 times and it is rejected if he fails to do so. We can change the state of promise by calling reolsve and reject state. When we resolve or reject the promise the any callbacks that we add to the promise are then called. We chained .then and .catch to our promise. We are returning promise from our function so these are attached to our promise. If promise is resolved then callback passed to then is called, if it is rejected then callback passes to catch is executed. As I know my promise will be either resolved or rejected, I know that either of these 2 callbacks will be executed.

Once we change the state of promise to resolve or reject, it enters that state. It will never ever change again.

Here we have created a promise then we used it. Most of js libarries like mongoose , will require you only to use promise. You dnt need to create one from scratch.