Async JS Unit Testing

Team: El Pimpi

Mihail Gaberov

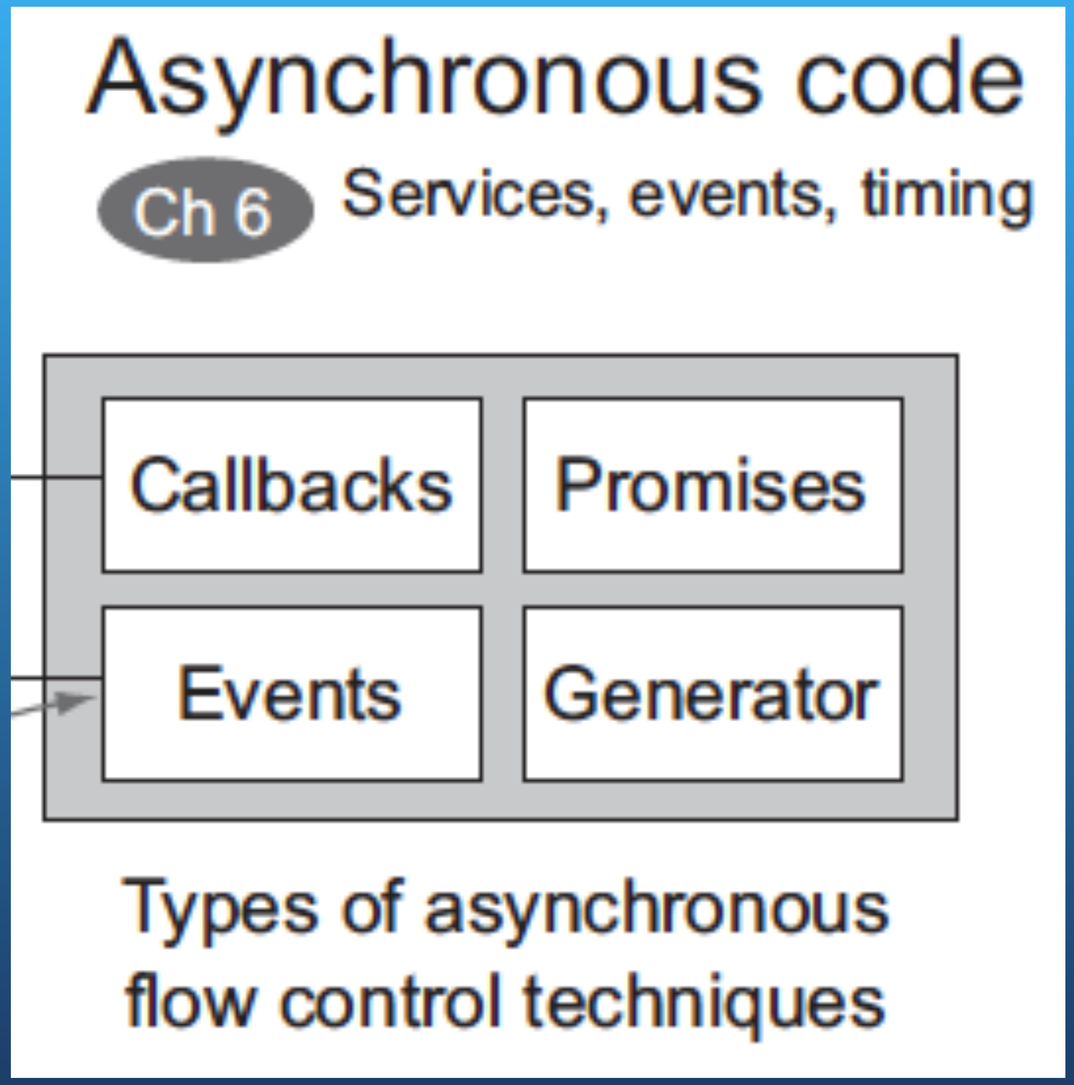
Lottoland

dreamIT

Motivation

- To have all team/community members acknowledged in the field and be able to write unit test for our Javascript based projects.
- This could be continued as series of internal workshops in order to keep improving ourselves in all front-end skills that we need or might need.

What is asynchronous code?



What is an async test?

- Unit tests for asynchronous code, code that returns promise of something that will happen in the future.

```
it("Using a Promise with async/await that resolves successfully with wrong expectation!", async function() {
    var testPromise = new Promise(function(resolve, reject) {
        setTimeout(function() {
            resolve("Hello World!”);
        }, 200);
    });

    var result = await testPromise;

    expect(result).to.equal("Hello!");
});
```

How do we write JS unit tests?

- Callbacks - callbacks hell:

```
function callbackHell () {
  const api = new Api()
  let user, friends
  api.getUser().then(function (returnedUser) {
    user = returnedUser
    api.getFriends(user.id).then(function
(returnedFriends) {
      friends = returnedFriends
      api.getPhoto(user.id).then(function (photo) {
        console.log('callbackHell', { user, friends,
photo })
      })
    })
  })
}
```

"In a real codebase, each callback function might be quite long, which can result in huge and deeply indented functions. Dealing with this type of code, working with callbacks within callbacks within callbacks, is what is commonly referred to as "callback hell"."

How do we write JS unit tests?

- Promises - they can be chained by returning another promise inside each callback, this way we can keep all of the callbacks on the same indentation level. We're also using arrow functions to abbreviate the callback function declarations.

```
function promiseChain () {
  const api = new Api()
  let user, friends
  api.getUser()
    .then((returnedUser) => {
      user = returnedUser
      return api.getFriends(user.id)
    })
    .then((returnedFriends) => {
      friends = returnedFriends
      return api.getPhoto(user.id)
    })
    .then((photo) => {
      console.log('promiseChain', { user, friends, photo })
    })
}
```

"Promises are a new, built-in type of object that help you work with asynchronous code. A promise is a placeholder for a value that we don't have yet but will at some later point. They're especially good for working with multiple asynchronous steps."

How do we write JS unit tests?

- Generators - Generators are a special type of function. Whereas a standard function produces at most a single value while running its code from start to finish, generators produce multiple values, on a per request basis, while suspending their execution between these requests.

```
function* WeaponGenerator(){
yield "Katana";
yield "Wakizashi";
yield "Kusarigama";
}

for(let weapon of WeaponGenerator()) {
assert(weapon !== undefined, weapon);
}

or

const weaponsIterator = WeaponGenerator();
const result1 = weaponsIterator.next();
```

"A generator function is defined by putting an asterisk right after the function keyword. We can use the new *yield* keyword in generator functions."

How do we write JS unit tests?

- Promises + Generators = Async await

```
async(function*() {
  try {
    const ninjas = yield getJSON("data/ninjas.json");
    const missions = yield getJSON(ninjas[0].missionsUrl);
    //All information received
  }
  catch(e) {
    //An error has occurred
  }
});
```

```
(async function (){
  try {
    const ninjas = await getJSON("data/ninjas.json");
    const missions = await getJSON(missions[0].missionsUrl);
    console.log(missions);
  }
  catch(e) {
    console.log("Error: ", e);
  }
})()
```

"We put the code that uses asynchronous tasks in a generator, and we execute that generator function. When we reach a point in the generator execution that calls an asynchronous task, we create a promise that represents the value of that asynchronous task. Because we have no idea when that promise will be resolved (or even if it will be resolved), at this point of generator execution, we yield from the generator, so that we don't cause blocking."

How do we write JS unit tests?

- Sinonjs - test spies, stubs and mocks for JS

```
function myFunction(condition, callback){
  if(condition){
    callback();
  }
}

describe('myFunction', function() {
  it('should call the callback function', function() {
    var callback = sinon.spy();

    myFunction(true, callback);

    assert(callback.calledOnce);
  });
});
```

Examples

https://github.com/mihailgaberov/javascript-testing

References:

Simplifying Asynchronous Coding with Async Functions

Error handling Promises in JavaScript

Async/Await in JavaScript

Testing Asynchronous Code with MochaJS and ES7 async/await
Testing Asynchronous JavaScript

Async/Await will make your code simpler

Secrets of the JavaScript Ninja