stackbuilders

Are promises our only choice?

Using functional programming and some alternative asynchronous computations like:
Task, TaskEither, RemoteData, and Futures

I’m Nataly Rocha

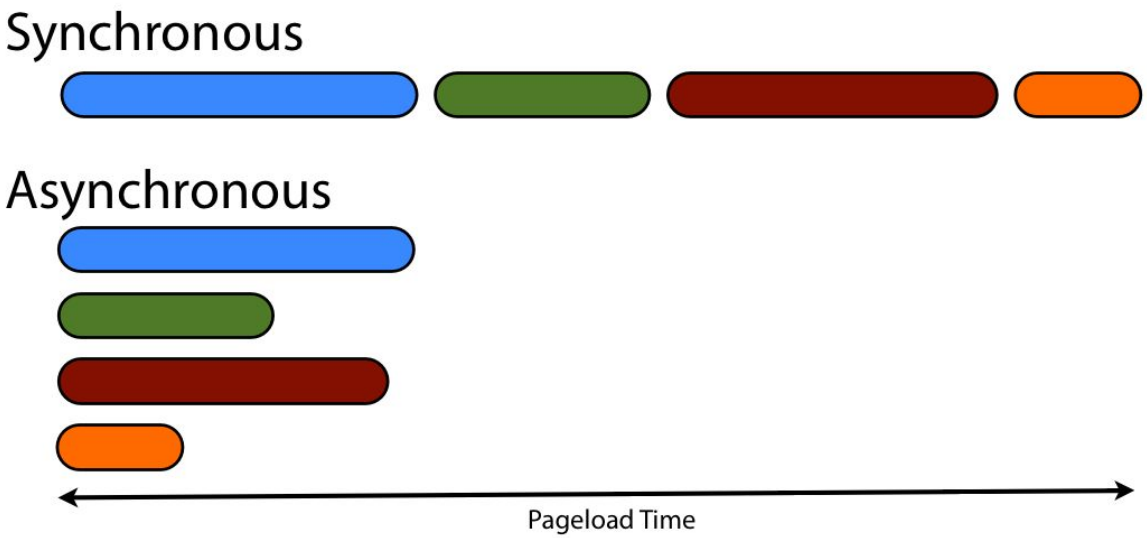
Quito - Ecuador
Software Developer

stackbuilders

Asynchronous operations

Think of asynchronous code as code that can start now, and finish its execution later.

- Callbacks
- Promises
- Async/Await



Promises

- Promises solved the callback hell
- Declarative
- Control Flow
- Railway oriented programming
- Fun to program in

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

Promises

- Promises solved the callback hell
- Declarative
- Control Flow
- Railway oriented programming
- Fun to program in

```
asyncThing1()
  .then(asyncThing2)
  .then(asyncThing3)
  .catch(asyncRecovery1)
  .then(asyncThing4, asyncRecovery2)
  .catch(() => console.log("Don't worry about it"))
  .then(() => console.log("All done!"))
```

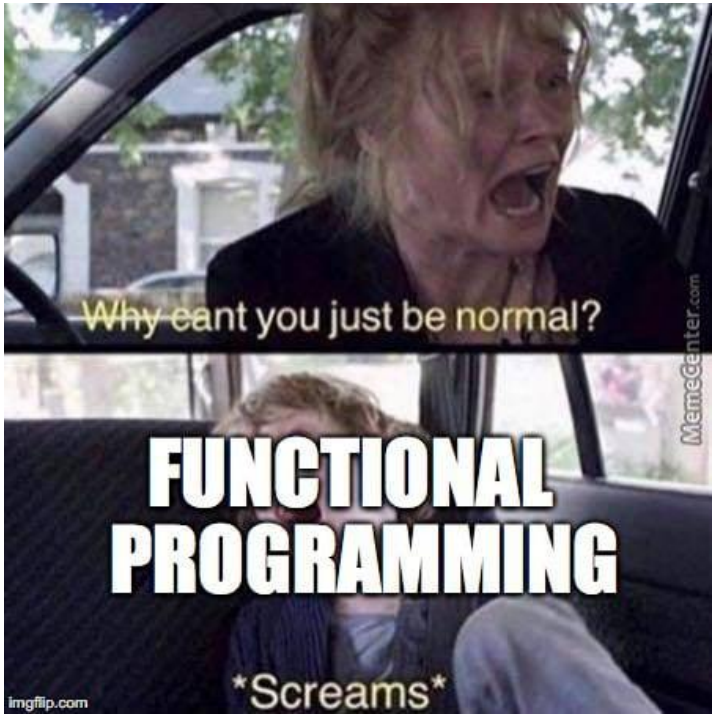
As good as promises are

Downsides:

- Eager evaluation
- Error types
- Unchecked exceptions
- Poor referential transparency
- Nestability can be complicated

stackbuilders

Functional Programming



What is it?

Functional programming (FP) is the process of building software by composing functions while avoiding: shared states, mutable data and side effects.

- FP is declarative rather than imperative in style.
- Some of the principles of FP are:
  - Immutability
  - Pure functions
  - Functions as "First-class citizens
  - Higher-order functions

Functional alternatives to promises

- Tasks
- TaskEither
- Futures
- RemoteData

fp-ts

effect-ts

Demo time!

stackbuilders

Other topics you can check

- Effect-ts
- RX-JS
- ADT: Algebraic Data Types
- Functional programming
- io-ts

fp-ts

effect-ts

Conclusions

- Promises are great but can always be improved
- Functional Programming is a mindset and a paradigm that can help you to have cleaner, more testable, more control and more readable code
- Learning curve can be a bit high but is worth it
- Use these libraries with caution
- Documentation can be hard to understand
- Enjoy programming

stackbuilders

Further Lectures

- https://dev.to/anthonyjoeseph/taskeither-vs-promise-2g5e
- https://dev.to/avaq/fluture-a-functional-alternative-to-promises-21b
- https://dev.to/gcanti/functional-design-algebraic-data-types-36kf
- https://github.com/MostlyAdequate/mostly-adequate-guide

stackbuilders

Thank you!

Nataly Rocha

stackbuilders