SPA-Maintainability

Maintainability

What is maintainable code?

it is easy to understand and troubleshoot

it is easy to test

it is easy to refactor

Maintainability

What is hard-to-maintain code?

- it has many dependencies, making it hard to understand and hard to test independently of the whole
- it accesses data from and writes data to the global scope, which makes it hard to consistently set up the same state for testing
- it has side-effects, which means that it cannot be instantiated easily/repeatably in a test
- it exposes a large external surface and doesn't hide its implementation details, which makes it hard to refactor without breaking many other components that depend on that public interface

Maintainability

What is modular code?

Modular code is code which is separated into independent modules.

Modularity is not just about code organization.

NameSpaces

The problem with namespaces

- Choices about privacy have to be made on a global basis
- Modules become dependent on global state.

Do not -

Do not leak global variables

Do not expose implementation details

Do not mix definition and instantiation/initialization

Do not mix definition and instantiation/initialization

Do not modify objects you don't own

how can we implement modules and packages for our single page application?

We want to solve three problems:

- Privacy

- Avoid putting things in the global namespace

- We should able to create packages

Building modules and packages using CommonJS

CommonJS modules.

CommonJS is the module format that Node.js uses natively. A CommonJS module is simply a piece of JS code that does two things:

it uses require() statements to include dependencies

it assigns to the exports variable to export a single public interface

common JS

What are the benefits?

It does not accidentally modify global state, and it only exports one thing

Dependencies are easy to locate, without being modifiable or accessible in the global scope

The module does not give itself a name

It comes with a distribution system

There are thousands of compatible modules

CommonJS modules can be nested to create packages.

Big Principles

1. Use a build system and a module convention that supports granular privacy.

2. Independent packages/modules

3. Do not intermingle state and definition

4. Small external surface

Guidelines for new projects

1. Start with the package.json file.

2. Add a single bootstrap function. Loading modules should not have side-effects.

3. Write tests before functionality.

4. Hide implementation details.

5. Minimize your exports. Small surface area.

6. Localize dependencies.

Tooling: npm

NPM - Node Package Manager