The Way of the Manul

Meteor lands in the land of choices

2014: Meteor was an all-in-one solution with its own ecosystem

2016: Meteor opened to React, Angular, NPM and announced new data-layer

Ye olde castle

Angular

The new kids in the block

Functional programming

React

GraphQL

Redux

JS-World is evolving fast

- Evolution means change
- Not every aspect of an app evolves the same:

  Data-layer, UI, Business-logic

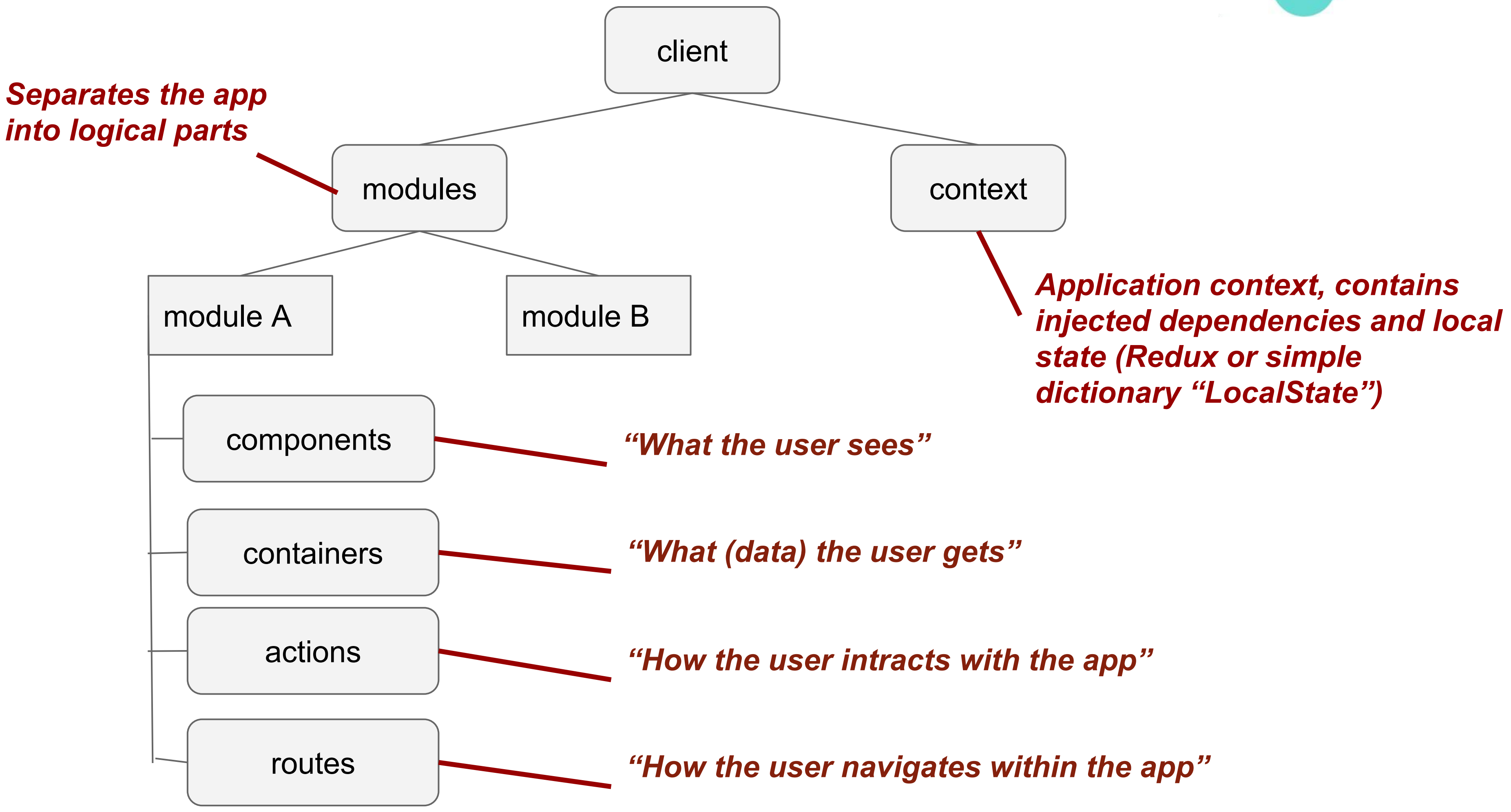
→ We need to embrace change, but keep maintainability

Enter Mantra

Architecture spec for component-driven frontend-development

- contains best practices
- disciplined, but straight-forward
- designed for meteor but works without
- makes testing easier
- future-proof
- mantra-cli helps creating entities
- small core helper library (100 loc)
- targets currently client only

Mantra Architecture


client
modules
context
Separates the app into logical parts
module A
module B
Application context, contains injected dependencies and local state (Redux or simple dictionary "LocalState")
components
"What the user sees"
containers
"What (data) the user gets"
actions
"How the user intracts with the app"
routes
"How the user navigates within the app"

Components

We prefer stateless, *functional* (react) components:

```
const BlogPost = ({title, text, imageUrl, edit}) => (
    <article>
        <h1>{title}</h1>
        <img src={imageUrl} style={{width: “100%”}} />
        <p>{text}</p>
        <button onClick={edit} >Edit this post</button>
    </article>
);
```

aka.
“Pure” components,
“dumb” components,
UI-components

Components

Components define the UI.

(Stateless) UI-components...

- are expressive
- easy to reason about
- easy to test / verify
- framework-agnostic (they should never contain meteor code)
- have no access to context or actions
- can be designed outside the application with *storybook* → *Demo*
- *functional* is King.

Components tests

Components can be unit-tested with Airbnb's Enzyme:

```
import {expect} from 'chai';
import {mount} from 'enzyme';
import Button from '../Button';
import sinon from 'sinon';

describe('Button', () => {
  it('calls onClick when clicked', () => {
    const onClick = sinon.spy();
    const el = mount(
      <Button onClick={onClick} />
    );
    el.find('button').simulate('click');
    expect(onClick).to.have.been.called();
  })
});
```

Containers

Container knows how to fetch data and resolve depencencies.

Containers...

- know how to fetch data:
  - from Meteor.Collection and subscribe to data
  - from local state (Redux or LocalState)
  - can easily changed to appollo (graphQL), Redux, Ajax-requests, or whatever you like
- can use depencencies in context
- can map actions to properties for the nested component

Containers

In mantra we use react-komposer to “kompose” containers.

Example:
https://git.panter.ch/pvl/biketowork/blob/develop/app/client/modules/core/containers/company_search.js#L0

Containers tests

Containers export the composed component as default, but export also the compose-function and the depsMapper for testing:

```
import {composer} from '../company_search';
describe('core.containers.company_search', () => {
  describe('composer', () => {
    const Meteor = {
      subscribe: spy()
    };
    it('should subscribe to company.search', () => {
      composer({Meteor}, spy());
      expect(Meteor.subscribe).to.have.been.calledWith('company.search', 'test query')
    })
  });
});
```

Actions

Actions define logic that a user can trigger.

Actions...

- can change the local state (Redux / LocalState)
- can call meteor-methods to manipulate data
- have access to the context

Example:
https://git.panter.ch/pvl/biketowork/blob/develop/app/client/modules/account/actions/account.js

Actions - tests

They are unit-tested similar to containers:

```
import actions from '../account';
describe('actions.account.login', () => {
  it('should set LOGIN_ERROR on wrong logins', () => {
    const LocalState = {set: spy()};
    const Meteor = {loginWithPassword: (username, email, callback) =>
callback({message: "Not found"})};
    actions.login({Meteor, LocalState}, {email: "test@example.com", password:
"test1234"});
    const args = LocalState.set.args[0];
    expect(args[0]).to.be.equal('LOGIN_ERROR');
    expect(args[1]).to.match(/Not found/);
  });
});
```

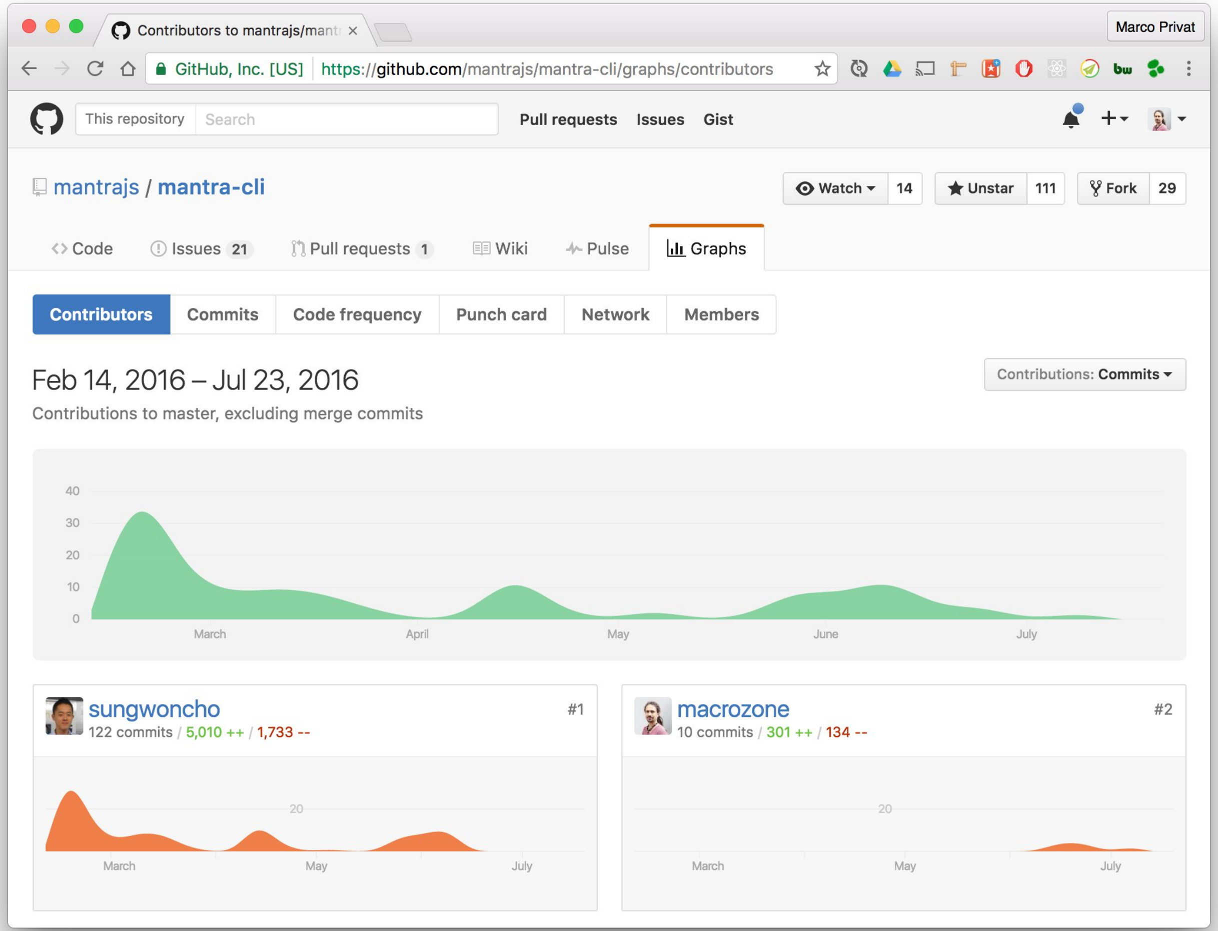
Routes

Routes define (surprise) local routes:

```
import { mount } from 'react-mounter';
import BlogPost from './containers/blog_post’;

export default function (injectDeps, {FlowRouter) {
  const MainLayoutCtx = injectDeps(MainLayout);
  FlowRouter.route('/:postId', {
    name: 'blogPost',
    action({postId) {
      mount(MainLayoutCtx, {
         content: () => (<BlogPost postId={postId} />)
      });
    }
  });
```

mantra-cli



Contributors to mantrajs/mant ×

Marco Privat

GitHub, Inc. [US] | https://github.com/mantrajs/mantra-cli/graphs/contributors

mantrajs / mantra-cli

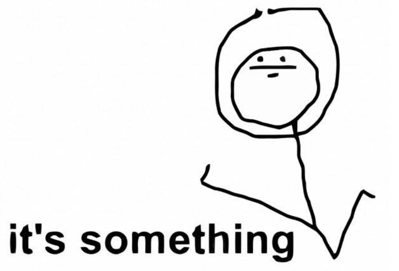
Feb 14, 2016 – Jul 23, 2016

Contributions to master, excluding merge commits

Contributions: Commits

sungwoncho #1
122 commits / 5,010 ++ / 1,733 --

macrozone #2
10 commits / 301 ++ / 134 --



Styling

- **C**SS is hard
- Isolating component styles with CSS is hard
- BEM solves scoping problems by adressing dom-elements implicitly, but BEM is ~~hard~~ ugly
- Managing CSS dependencies, preprocessors, etc. is hard
- Creating a UI-Component and styling it is usually one task, but CSS separates it → hard

→ CSS is often the wrong tool!

Enter inline-styles

- Highly expressive
- No more dead code
- No more cascading hell
- Manage style like any other code: as JS!
- Calculate styles depending on properties
- React native does it too…
- easy, nothing new to learn

Caveats:

- No direct support for pseudo-selectors (:hover, :active, :before, …)
- → can be solved with Radium
- CSS still a good solution for general styling (reset, fonts, etc.)
- Overriding inline-styles depend on implementation of component

  → UI-libraries need good patterns

What’s next

Apollo:

- New datalayer for meteor
- based on GraphQL from Facebook
- Opens meteor for new databases and -interfaces
- In development, client and server already available,
- but no sophisticated meteor integration yet

React native:

- Create native IOS and Android Apps with react
- mantra&meteor work nicely with react native