VulcanJS

The fullstack React+GraphQL framework

7 nov. 2018

Apollinaire Lecocq

THE PROBLEM

How it feels to learn JavaScript in 2016

No JavaScript frameworks were created during the writing of this article.

The following is inspired by the article "It's the future" from Circle CI. You can read the original here. This piece is just an opinion, and like any JavaScript framework, it shouldn't be taken too seriously.

Hey, I got this new web project, but to be honest I haven't coded much web in a few years and I've heard the landscape changed a bit. You are the most up-to date web dev around here right?

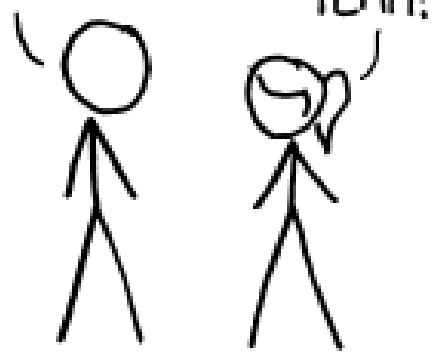
-The actual term is Front End engineer, but yeah, I'm the right guy, I do

THE SOLUTION

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.

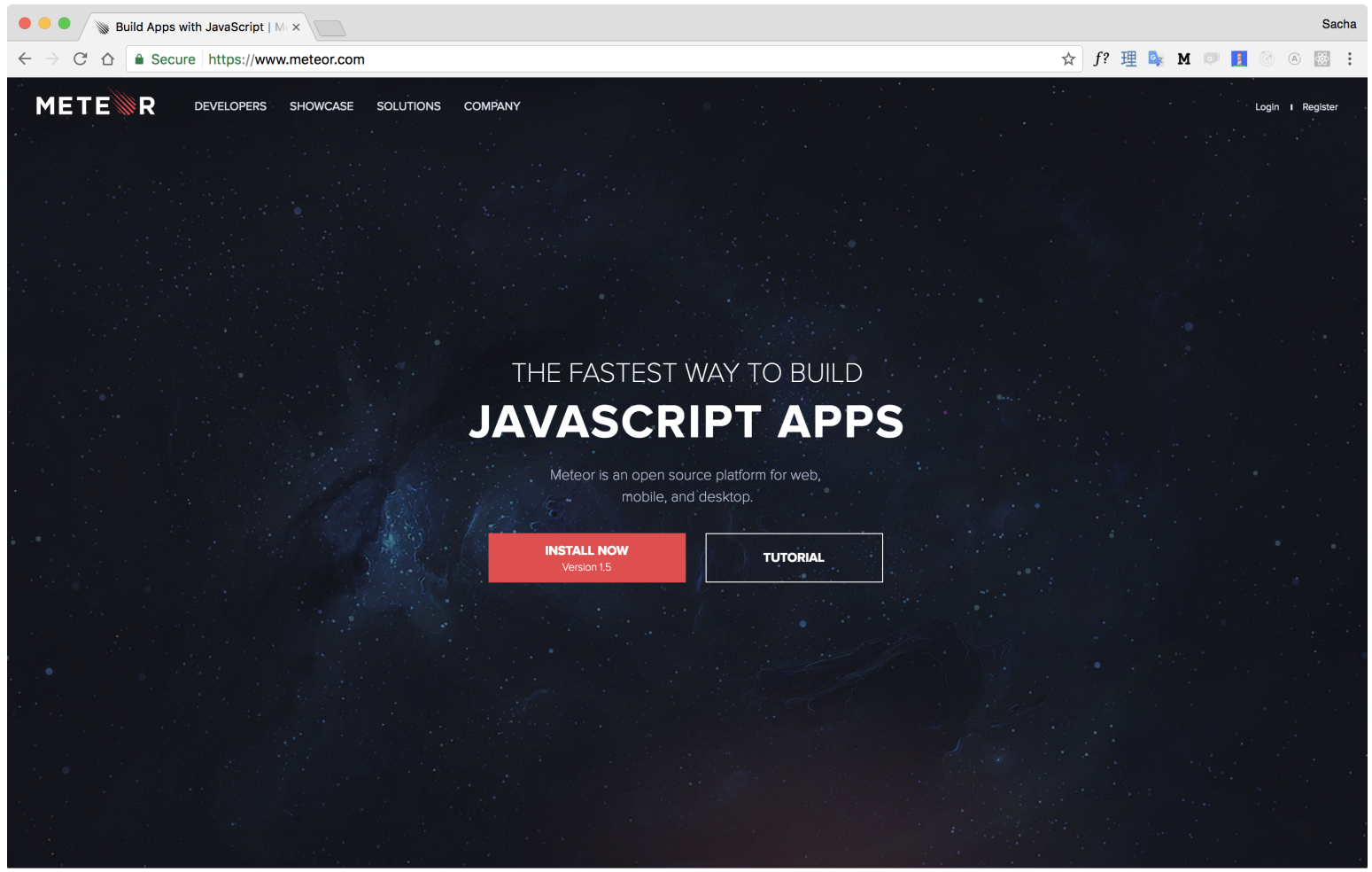


SOON:

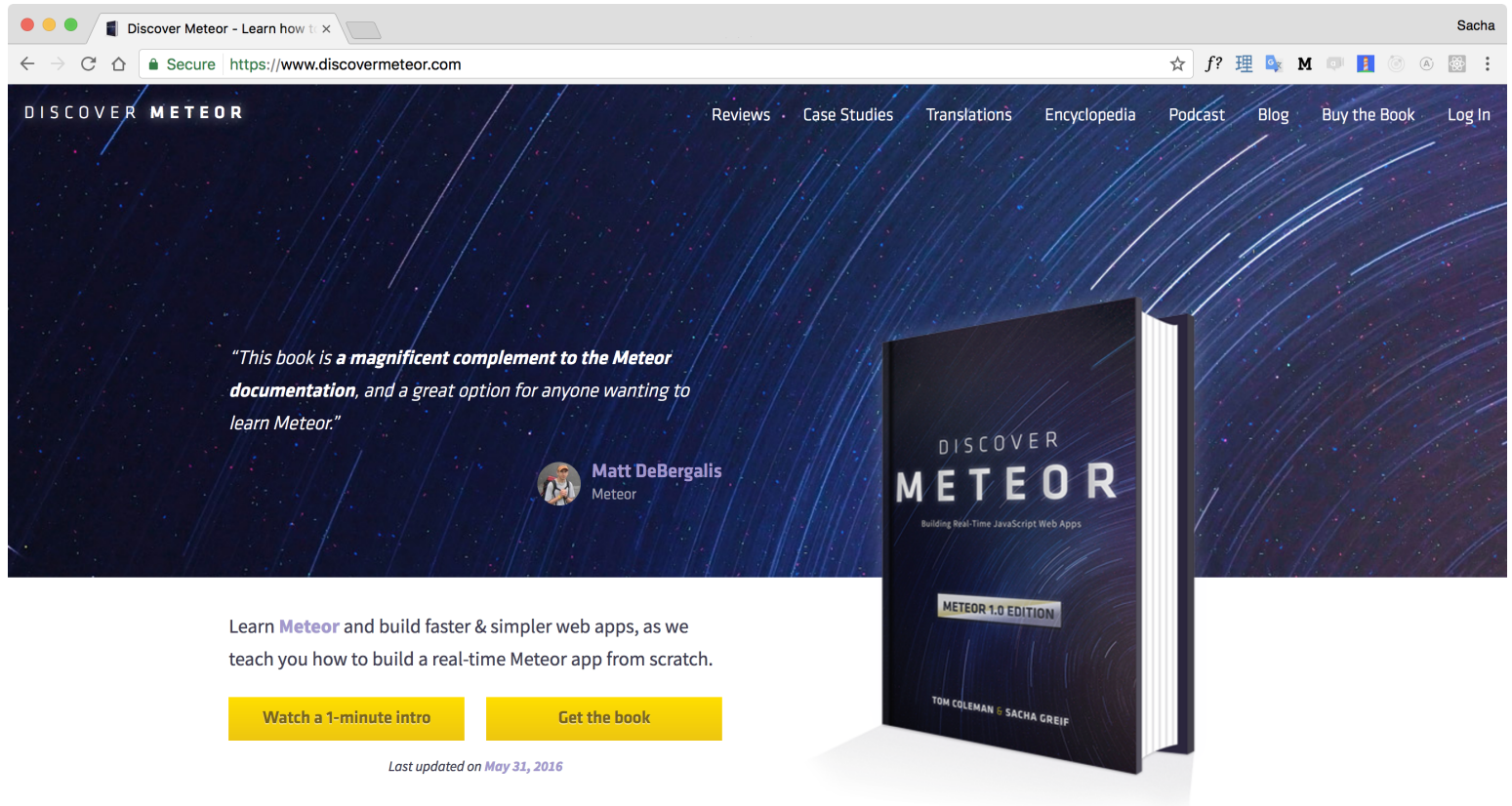
SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

ORIGIN STORY

2012: Meteor


Build Apps with JavaScript | M
Sacha
Secure
https://www.meteor.com
METEOR
DEVELOPERS
SHOWCASE
SOLUTIONS
COMPANY
Login
Register
THE FASTEST WAY TO BUILD
JAVASCRIPT APPS
Meteor is an open source platform for web, mobile, and desktop.
INSTALL NOW
Version 1.5
TUTORIAL

2013: Discover Meteor



Learn Meteor and build faster & simpler web apps, as we teach you how to build a real-time Meteor app from scratch.

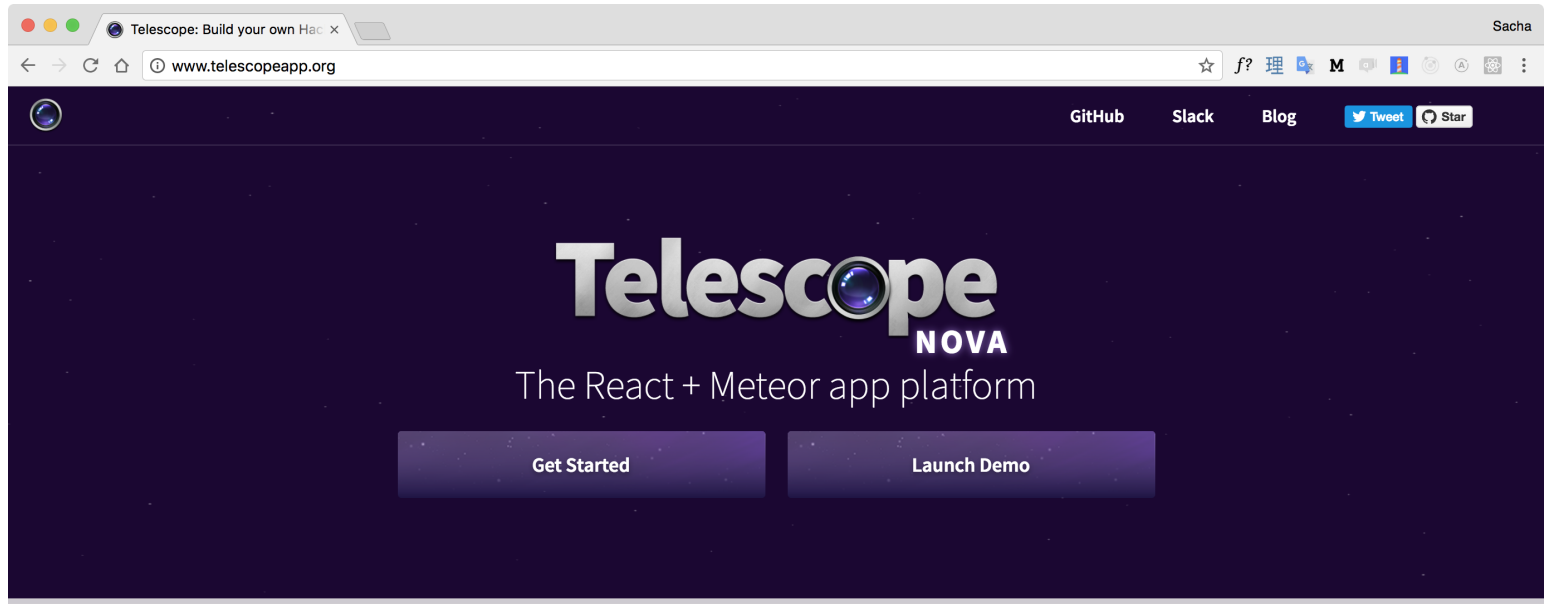
Watch a 1-minute intro | Get the book

Last updated on May 31, 2016

What People Are Saying

Contact us

2013-2017: Telescope/Nova



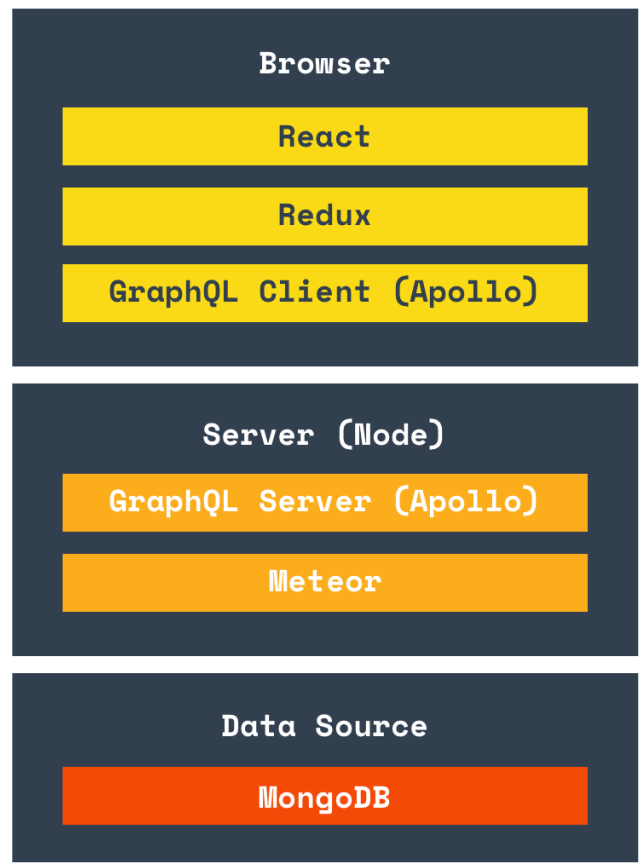
Telescope Nova is a **free**, **open-source** app platform built with **React** and powered by **Meteor** on the back-end.

Nova provides simple building blocks such as posts, comments, forms, and modals, and makes it easy to quickly customize them to build modern, social web apps.

2017: ENTER VULCAN.JS

So what's Vulcan.js ?

The Vulcan.js Stack



FEATURES

User accounts

GraphQL schema generation

```
const schema = {
  _id: {
    type: String,
    optional: true,
    canRead: ['guests'],
  },
  createdAt: {
    type: Date,
    optional: true,
    canRead: ['guests'],
  },
  userId: {
    type: String,
    optional: true,
    canRead: ['guests'],
    resolveAs: {
      fieldName: 'user',
      type: 'User',
      resolver: (movie, args, context) => {
        return context.Users.findOne({ _id: movie.use
      },
      addOriginalField: true,
    },
  },
  name: {
    label: 'Name',
    canRead: ['guests'],
    type: String,
    optional: false,
  },
  year: {
    label: 'Year',
    canRead: ['guests'],
    type: Number,
    optional: true,
  },
  review: {
    label: 'Review',
    canRead: ['guests'],
    type: String,
    optional: true,
  },
};

export default schema;
```

Query
movie(input:SingleMovieInput): SingleMovieOutput
movies(input:MultiMovieInput): MultiMovieOutput

SingleMovieOutput
result: Movie

MultiMovieOutput
results: [Movie]
totalCount: Int

Movie
_id: String
createdAt: Date
user: User
userId: String
name: String
year: Float
review: String

CRUD Operations

```
const Movies = createCollection({
  collectionName: 'Movies',
  typeName: 'Movie',
  schema: schema,
  resolvers: getDefaultResolvers({typeName:'Movie'}),
  mutations: getDefaultMutations({typeName: 'Movie'}),
});
```

Simple data loading and updating

```
const listOptions = {
  collection: Movies,
  fragment: fragment
};

export default withMulti(listOptions)(MoviesList);
```

Automated forms

```
<Components.SmartForm collection={Movies} />
```

```
<Components.SmartForm
  collection={Movies}
  documentId={this.props.documentId} />
```

Never code a form again!

Insert New Document

Name

Year

Review

Submit

Edit Movie

Name: Rush Hour

Year: 1993

Review: Jackie Chan at his best.

Submit

Delete

Permission system

```
const schema = {
  name: {
    type: String,
    label: 'Name',
    optional: false,
    canRead: ['guests'],
    canCreate: ['members'],
    canUpdate: ['admins', Users.owns]
  }
}
```

Internationalization

And more!

SSR

Components

Debug tools

Integrations

Community plugins

Theming

Email, newsletter and templates

Getting started tutorial

Welcome to Vulcan!

This interactive tutorial will teach you the basics of using Vulcan. In fact, the tutorial itself is the Vulcan app that we'll be working on! So meta!

On your left, you'll find this tutorial's **steps**. Right now they're greyed out, but as you unlock them by completing tasks you'll be able to progress forward.

On your right is your **work area**. As we build up our app starting on Step 10, its components will appear there.

During this tutorial, you'll have to edit a number of files. These are all located inside the **Getting Started** example package, which you can find at `/your-vulcan-directory/packages/getting-started`. From now on, file paths will be referenced using that directory as root.

While we're on this topic, let's take a quick look at that package's file architecture:

```
lib
  |- client           # client-only code
  |
  |- components       # React components
  |  |- movies
  |  |- other
  |  +- steps
  |
  |- hocs             # higher-order components
  |
  |- modules          # other JavaScript modules
  |
  |- server           # server-only code
  |
  |- stylesheets      # stylesheets
  |
  +- package.js       # package manifest
```

Everything starts with the `package.js` file, which defines the contents of the package. From there, two entry points in `client/main.js` and `server/main.js` tell Meteor which files to load in each environment.

By convention, every file *outside* of `client` and `server` will be loaded in *both* environments. This includes React components, higher-order components (special component wrapper functions), and all the other files in `modules`.

Throughout this tutorial, we'll be modifying files in both `components` and `modules`. But of course, feel free to take a look at the other parts of the package as well!

QUESTIONS ?

vulcanjs.org

slack.vulcanjs.org

github.com/VulcanJS/Vulcan