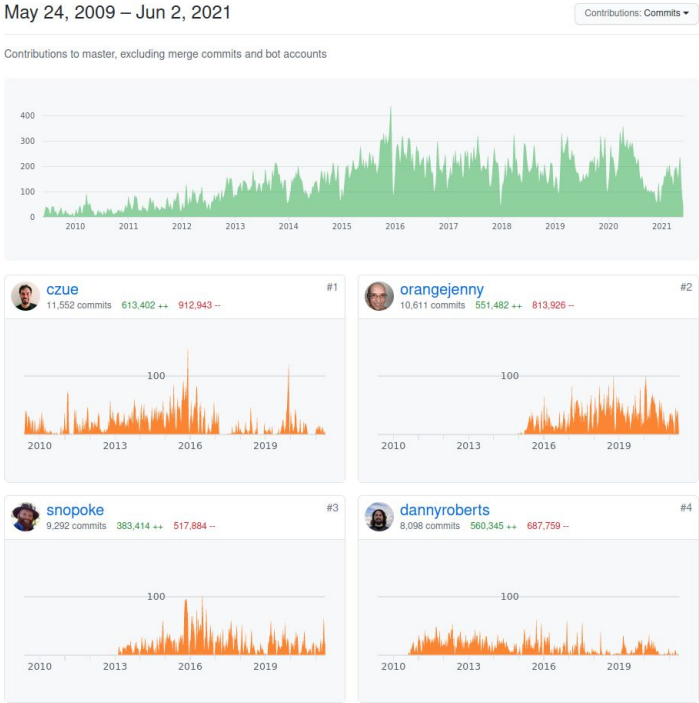
Modern JavaScript
for
Django Developers

Cory Zue | DjangoCon EU 2021
@czue

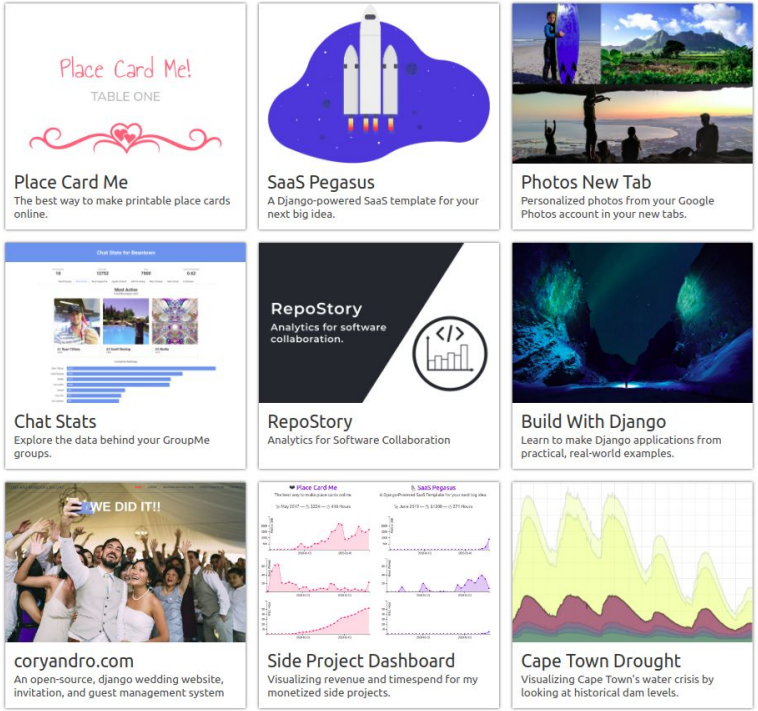
Who am I?

@czue

CTO, Dimagi 2006 - 2017



Side Project Entrepreneur 2017 - today



My Goals

1. Convince you that modern JavaScript is important, useful and not too scary.

2. Give you a roadmap to start using it in your Django projects.

This talk is loosely based on a series of articles I wrote last year available at https://www.saaspegasus.com/guides/

0. Why this talk?

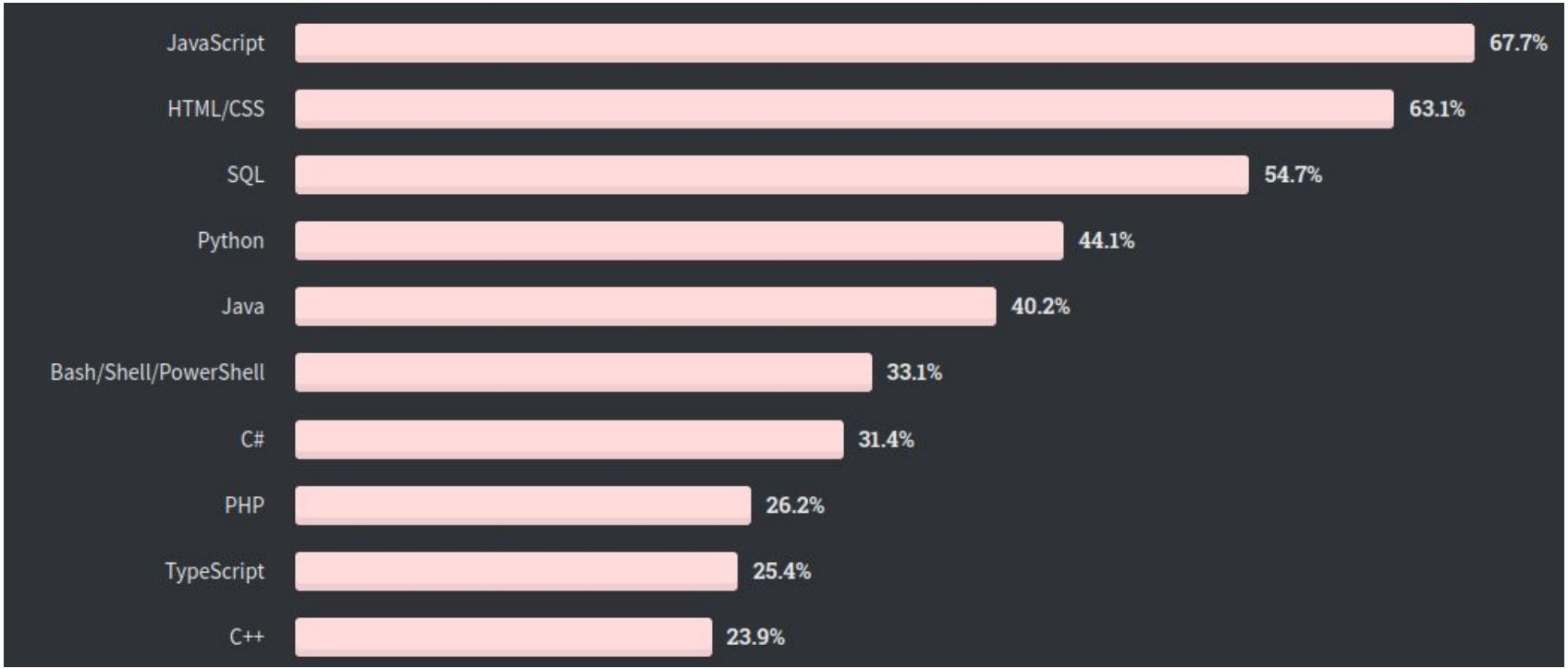
1. JavaScript is useful

Modern JavaScript for Django Developers (DjangoCon EU 2021)

1. JavaScript is useful

Click to add text

2. JavaScript is not going away



Most popular languages (2020)
Source: https://insights.stackoverflow.com/survey/2020

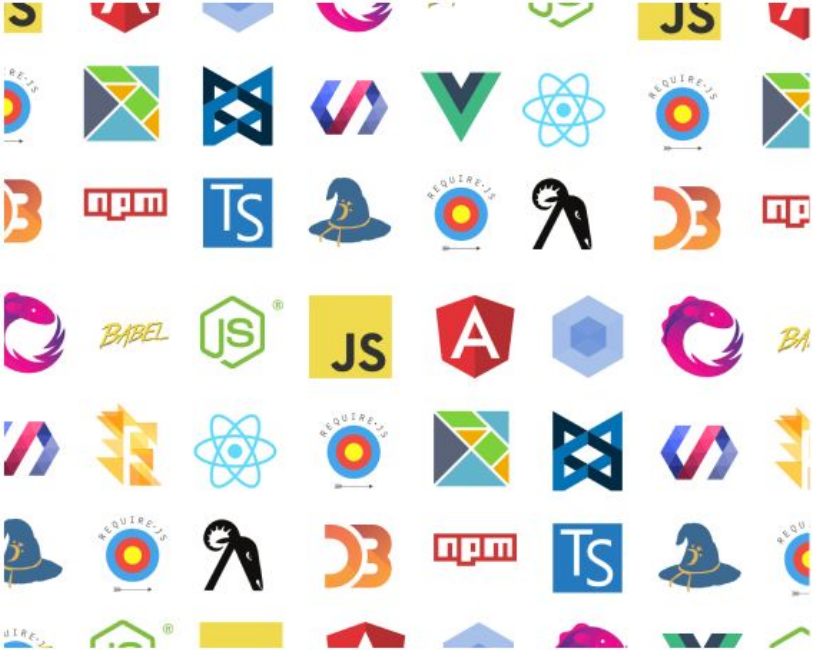
3. JavaScript is better than many think

Most loved frameworks (2020)
Source: https://insights.stackoverflow.com/survey/2020

4. JavaScript is hard

2008

2021



5. JavaScript isn’t standardized in the Django ecosystem

11:30 AM GMT+2 - 12:15 PM GMT+2

Telepath - adding the missing link between Django and rich client apps

Matt Westcott

3:50 PM GMT+2 - 4:35 PM GMT+2

You might not need a frontend framework

Afonso Cerejeira

9:45 AM GMT+2 - 10:35 AM GMT+2

Getting started with React, GraphQL, and Django

Aaron Bassett

JavaScript | Django documentation | Django

While most of **Django** core is Python, the admin and gis contrib apps contain **JavaScript** code.

Please follow these coding standards when writing **JavaScript** ...

Code style · JavaScript tests · Writing tests

3:05 PM GMT+2 - 3:50 PM GMT+2

HTMX: Frontend Revolution

Thomas Güttler

Now Playing

3:05 PM GMT+2 - 3:50 PM GMT+2

Full-stack Django Rest Framework

Israel da Silva Teixeira, Thiago Garcia da ...

11:30 AM GMT+2 - 12:15 PM GMT+2

Build, deploy and scale: Django, GraphQL and SPA

dhilipsiva

Can Django replace JavaScript?

How do I connect JavaScript to Django?

How use Django data in JavaScript?

Feedback

10:00 AM GMT+2 - 10:45 AM GMT+2

Anvil: Full Stack Web with Nothing but Python

Meredydd Luff

5:20 PM GMT+2 - 6:05 PM GMT+2

How to create a full-stack, reactive website in Django with absolutely no JavaScript

Adam Hill

In this talk

1. Organizing your front-end code
2. Integrating a React application into a Django project
3. Benefits and features of modern JavaScript

1. Organizing your front-end code

The “default” architecture

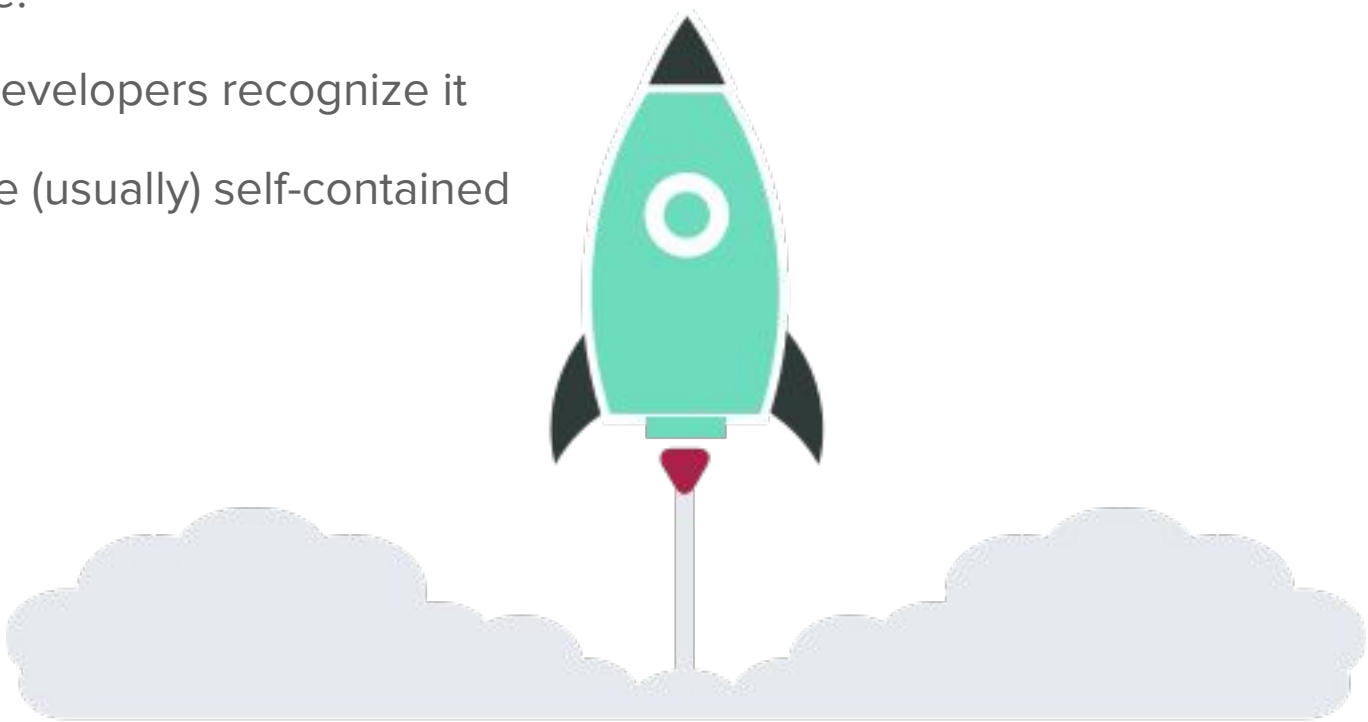
1. Build out site using normal Django tools (views, templates, forms, etc.)
2. Realize you need dynamic functionality on a page
3. Add some inline JS and maybe a library to your template
4. Go to 1

Why this is great

It’s simple!

Django developers recognize it

Pages are (usually) self-contained

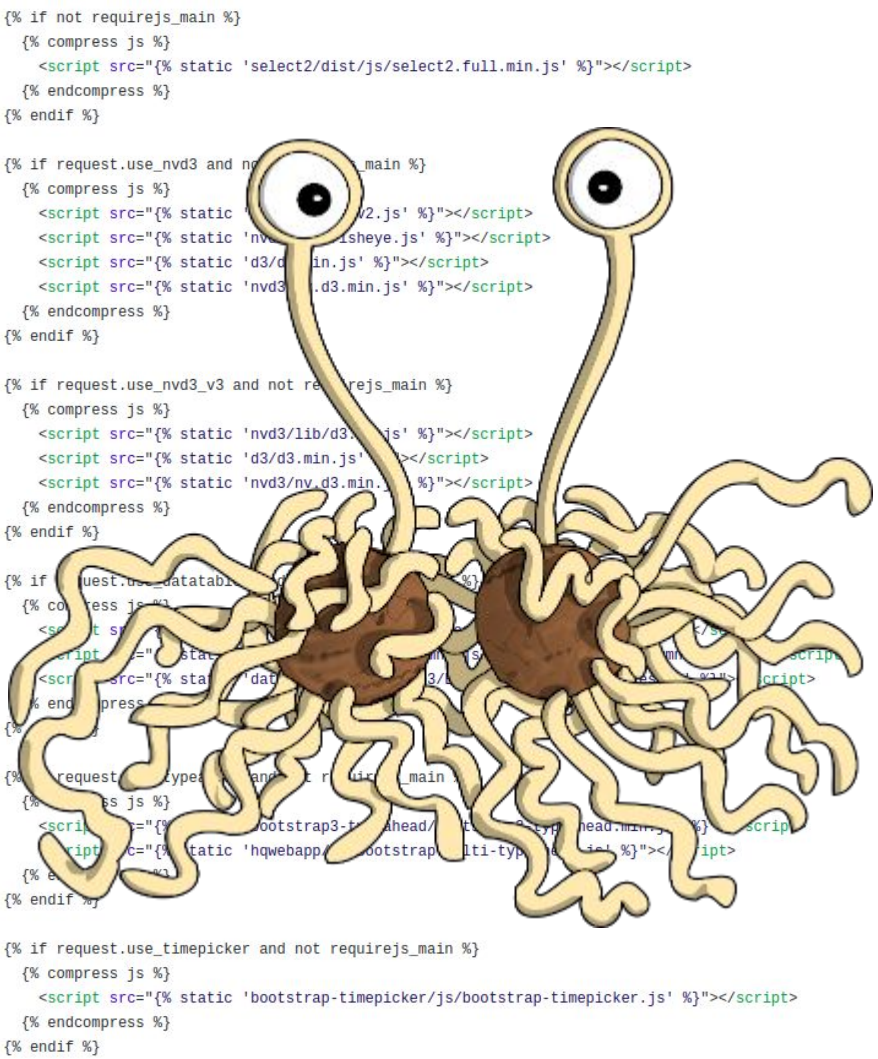


Problems

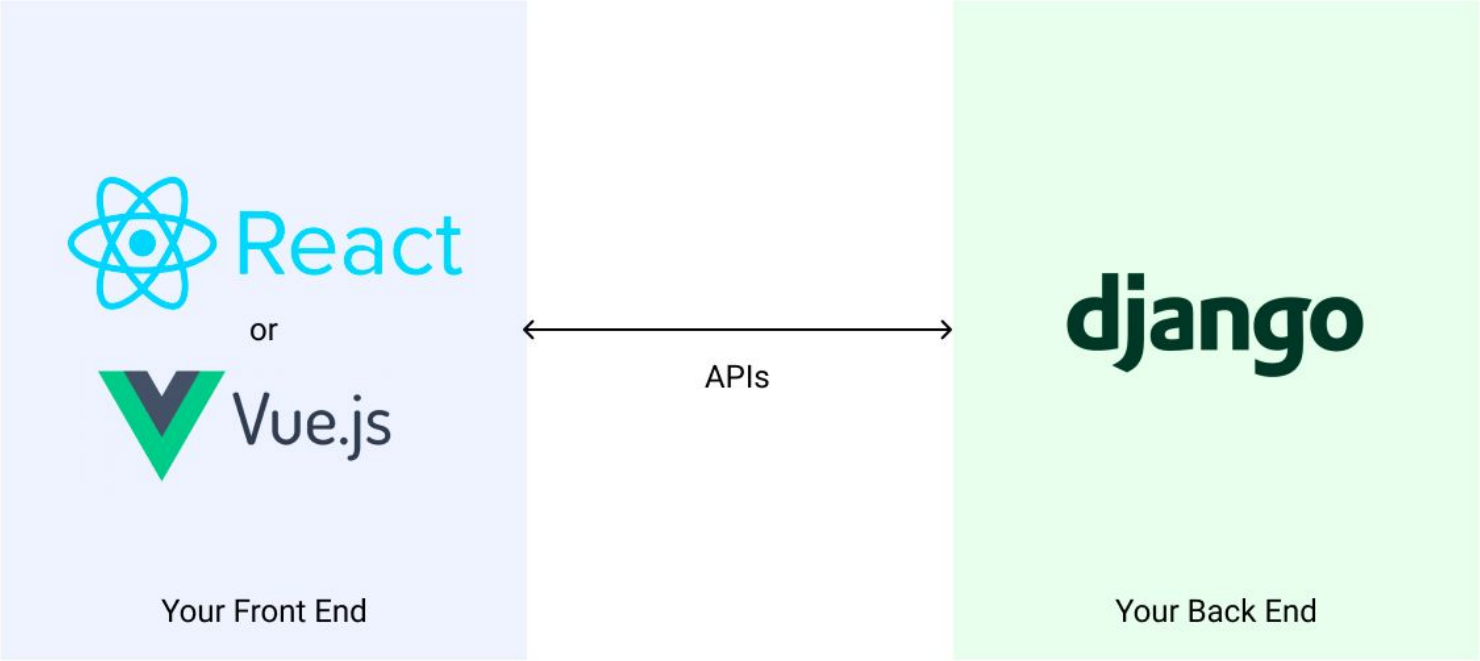
Gets more and more unwieldy over time

- Code sharing
- Template inheritance
- Mixing JS & Django template variables
- Homegrown front-end organization

Not leveraging modern JS features



Is there a better way?

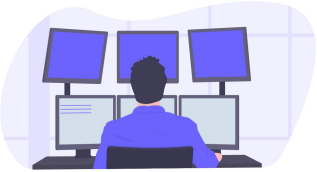


saaspegasus.com

Why this is great

Can go “all in” with a JS framework

Clean separation of back end and front end



React

or

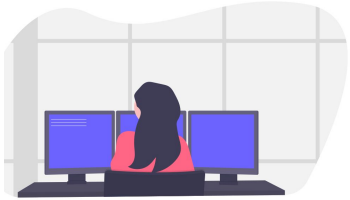
Vue.js

Your Front End

APIs

django

Your Back End

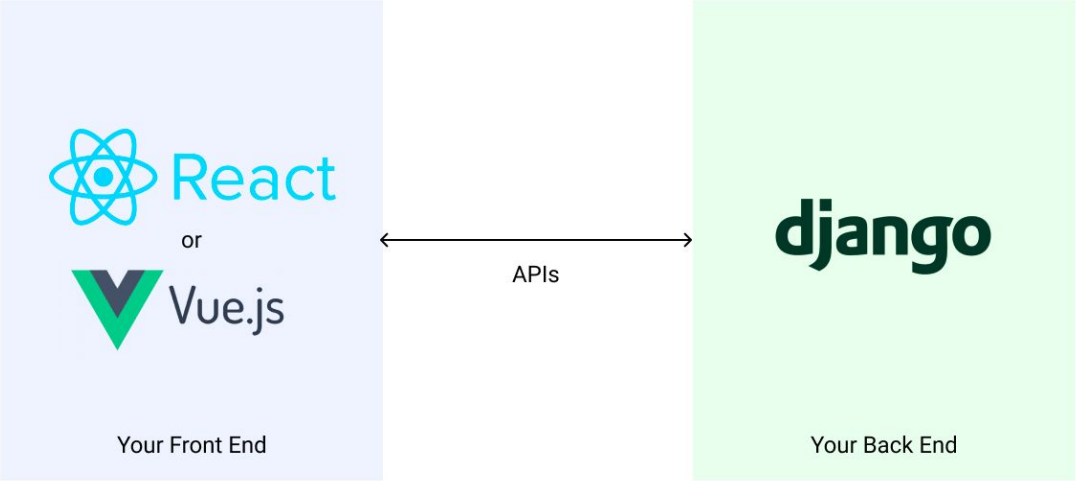


saaspegasus.com

Problems

It’s complicated! Simple tasks are more difficult. Deployment harder

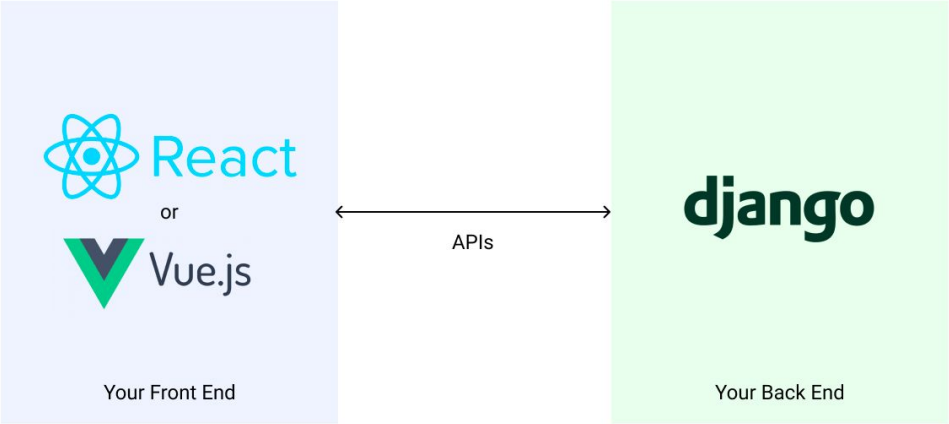
Lose features/familiarity of Django



saaspegasus.com

Intro to Django

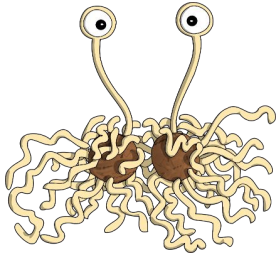
- ✅ Object-r[illegible]mal mapper
- ❌ URLs and
- ❌ Templates
- ❌ Forms
- 🤔 Authentic[illegible]
- ✅ Admin
- 🤔 Internationalization
- 🤔 Security



saaspegasus.com

“Batteries Included” → “Bring your own Batteries”

Our options...

	“Default”	**Client / Server**
Mascot		
Benefits	Simple! Familiar	Decoupled! “Modern” Good for front-end devs
Drawbacks	Gets unwieldy No “modern” JS	Complicated Slower velocity Lose much of Django

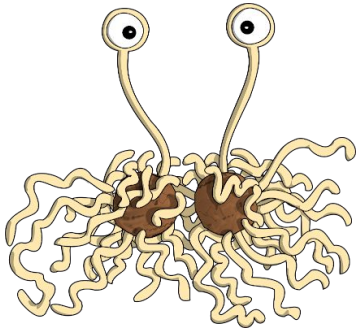
Can we get the best of both worlds?

Default to Django urls, views, templates, etc.

Drop in modern JS when it makes sense

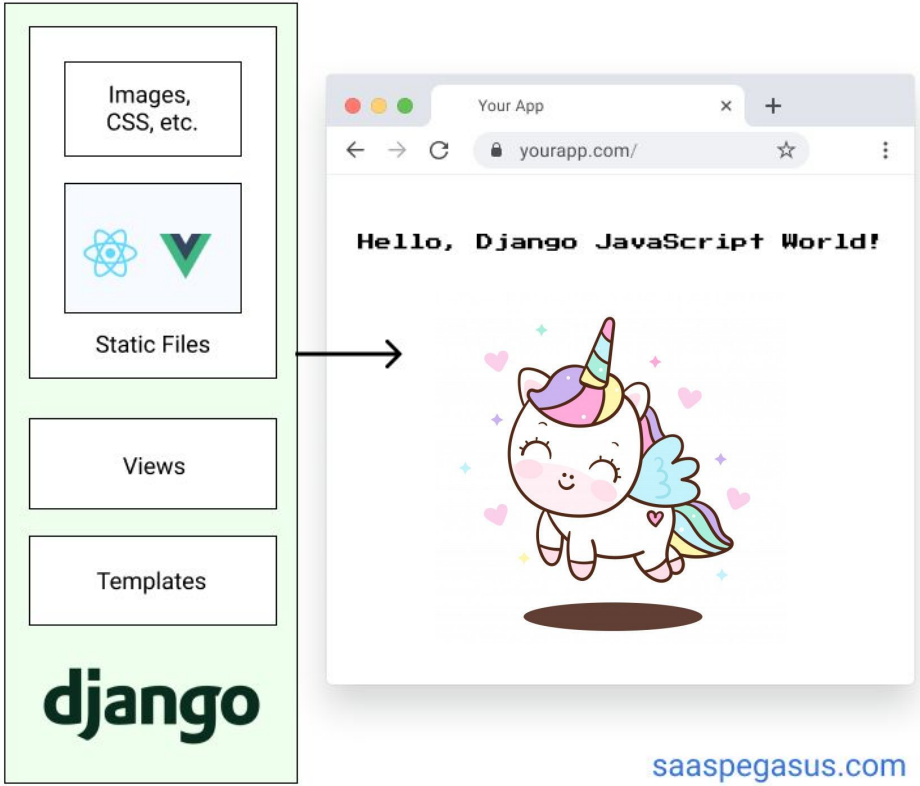
Yes we can!

Summarizing the options

	“Default”	**Client / Server**	**Hybrid**
Mascot			
When to use it	Very small projects When you have no complexity in the front end (but kind of never)	You prefer using a JS framework to Django You have a team with separate front/back-end devs	You like the idea of using Django for most things but also want a modern front end Solo-developers

2. Integrating modern JavaScript (React) into a Django project

What we want to do



saaspegasus.com

My experience adding React to a Django project

```
import React from 'react';
import ReactDOM from "react-dom";

ReactDOM.render(
  <h1>Hello, react!</h1>,
  document.getElementById('root')
);
```

I Need* a Toolchain!

Why JavaScript Toolchains?

In short, to use modern JavaScript on legacy browsers.

```
import React from 'react';
import ReactDOM from "react-dom";

ReactDOM.render(
  <h1>Hello, react!</h1>,
  document.getElementById('root')
);
```

Modules!

New Syntaxes!

What Toolchain should I use?

Recommended Toolchains

The React team primarily recommends these solutions:

- If you're **learning React** or **creating a new single-page app,** use Create React App.
- If you're building a **server-rendered website with Node.js,** try Next.js.
- If you're building a **static content-oriented website,** try Gatsby.
- If you're building a **component library** or **integrating with an existing codebase**, try More Flexible Toolchains.

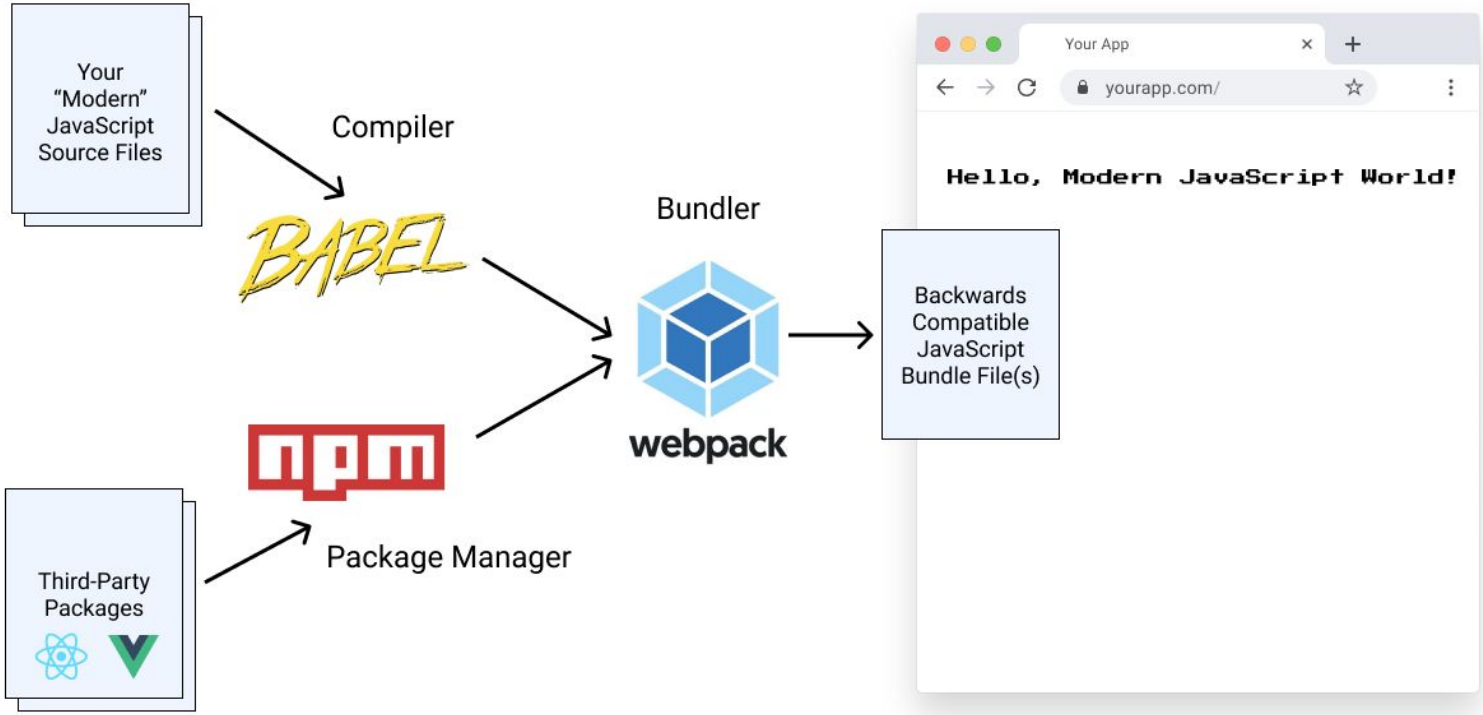
What Toolchain should I use?

More Flexible Toolchains

The following toolchains offer more flexibility and choice. We
experienced users:

- **Neutrino** combines the power of webpack with the simplicity of presets, and includes a preset for React apps and React components.
- **Nx** is a toolkit for full-stack monorepo development, with built-in support for React, Next.js, Express, and more.
- **Parcel** is a fast, zero configuration web application bundler that works with React.
- **Razzle** is a server-rendering framework that doesn't require any configuration, but offers more flexibility than Next.js.

Elements of a Toolchain



saaspegasus.com

The Package Manager

Is pip for JavaScript

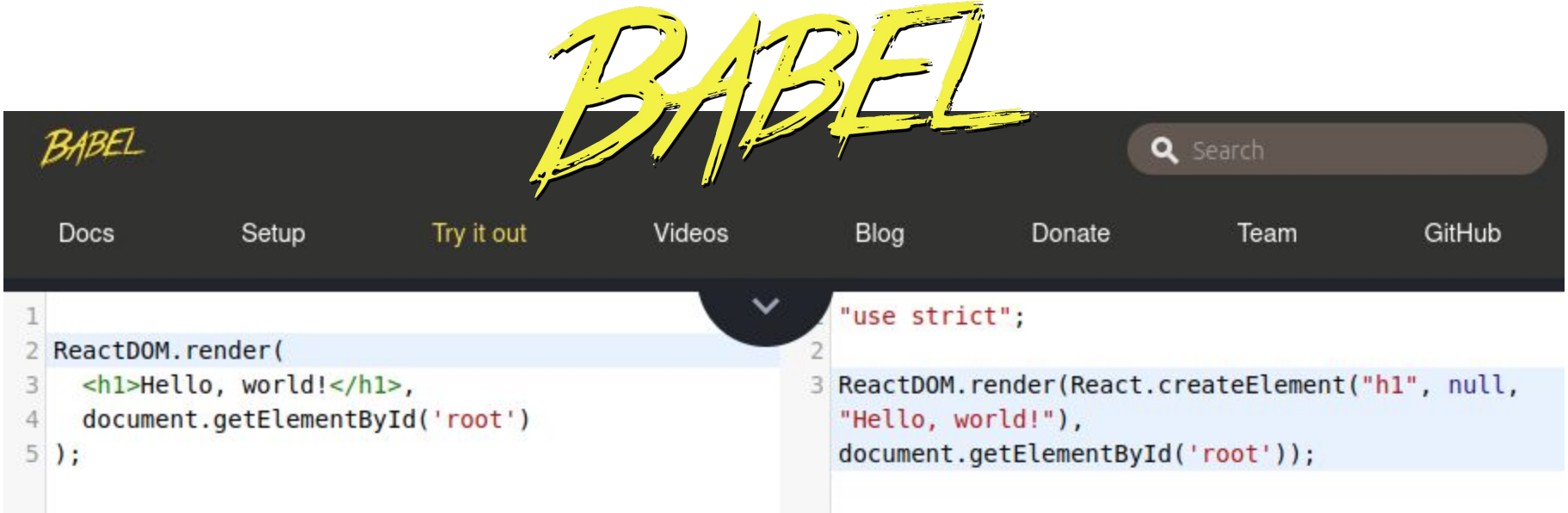
Main options are npm and Yarn

Doesn’t *really* matter what you pick, **if in doubt go with npm**

The Compiler

Takes new features and syntaxes and converts them to browser-friendly JavaScript

Babel is the most popular one out there today, and will do everything you need.



The Bundler

Makes your code more performant by bundling it into single, small files

Manages dependencies for you (so you don't have individually include libraries)

Often have ways of adding optional pre/post-processing (e.g. integrating a compiler)

Lots of options out there, **when in doubt, use webpack.**

```
{% if not requirejs_main %}
  {% compress js %}
    <script src="{% static 'select2/dist/js/select2.full.min.js' %}"></script>
  {% endcompress %}
{% endif %}

{% if request.use_nvd3 and not requirejs_main %}
  {% compress js %}
    <script src="{% static 'nvd3/lib/d3.v2.js' %}"></script>
    <script src="{% static 'nvd3/lib/fisheye.js' %}"></script>
    <script src="{% static 'd3/d3.min.js' %}"></script>
    <script src="{% static 'nvd3/nv.d3.min.js' %}"></script>
  {% endcompress %}
{% endif %}

{% if request.use_nvd3_v3 and not requirejs_main %}
  {% compress js %}
    <script src="{% static 'nvd3/lib/d
    <script src="{% static 'd3/d3.min.js
    <script src="{% static 'nvd3/nv.d3.mi
  {% endcompress %}
{% endif %}

{% if request.use_datatables and not
  {% compress js %}
    <script src="{% static 'datatables/media/js/jquery.dataTables.min.js' %}"></script>
    <script src="{% static 'datatables-fixedcolumns/js/dataTables.fixedColumns.js' %}"></script>
    <script src="{% static 'datatables-bootstrap3/BS3/assets/js/datatables.js' %}"></script>
  {% endcompress %}
{% endif %}

{% if request.use_typeahead and not requirejs_main %}
  {% compress js %}
    <script src="{% static 'bootstrap3-typeahead/bootstrap3-typeahead.min.js' %}"></script>
    <script src="{% static 'hqwebapp/js/bootstrap-multi-typeahead.js' %}"></script>
  {% endcompres
{% endif %

{% if requ                    and
  {% compr
    <scripu                    bootstrap-timepicker/js/bootstrap-timepicker.js' %}"></script>
  {% endcompress
{% endif %}
```

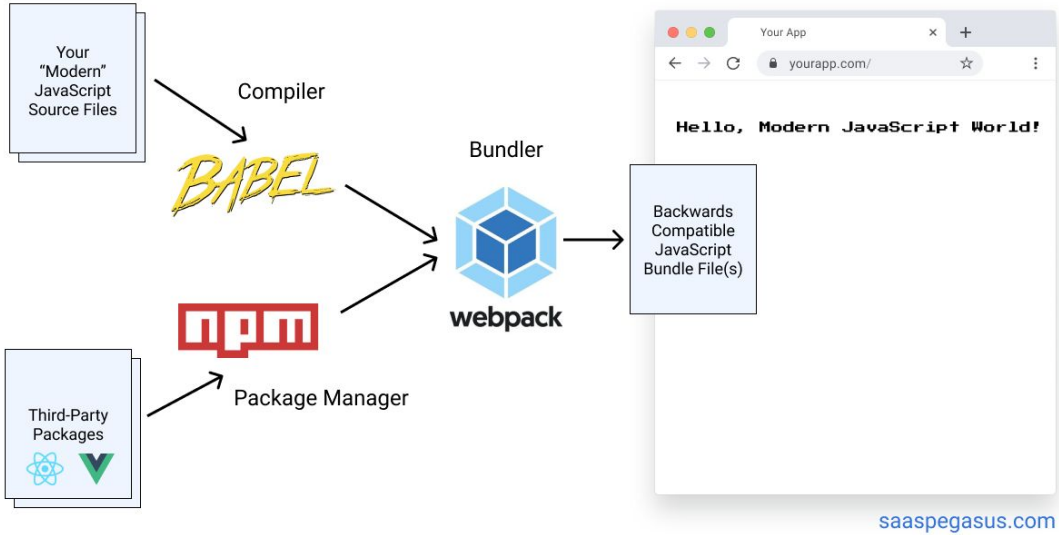
webpack

Putting it all together

Npm manages library imports

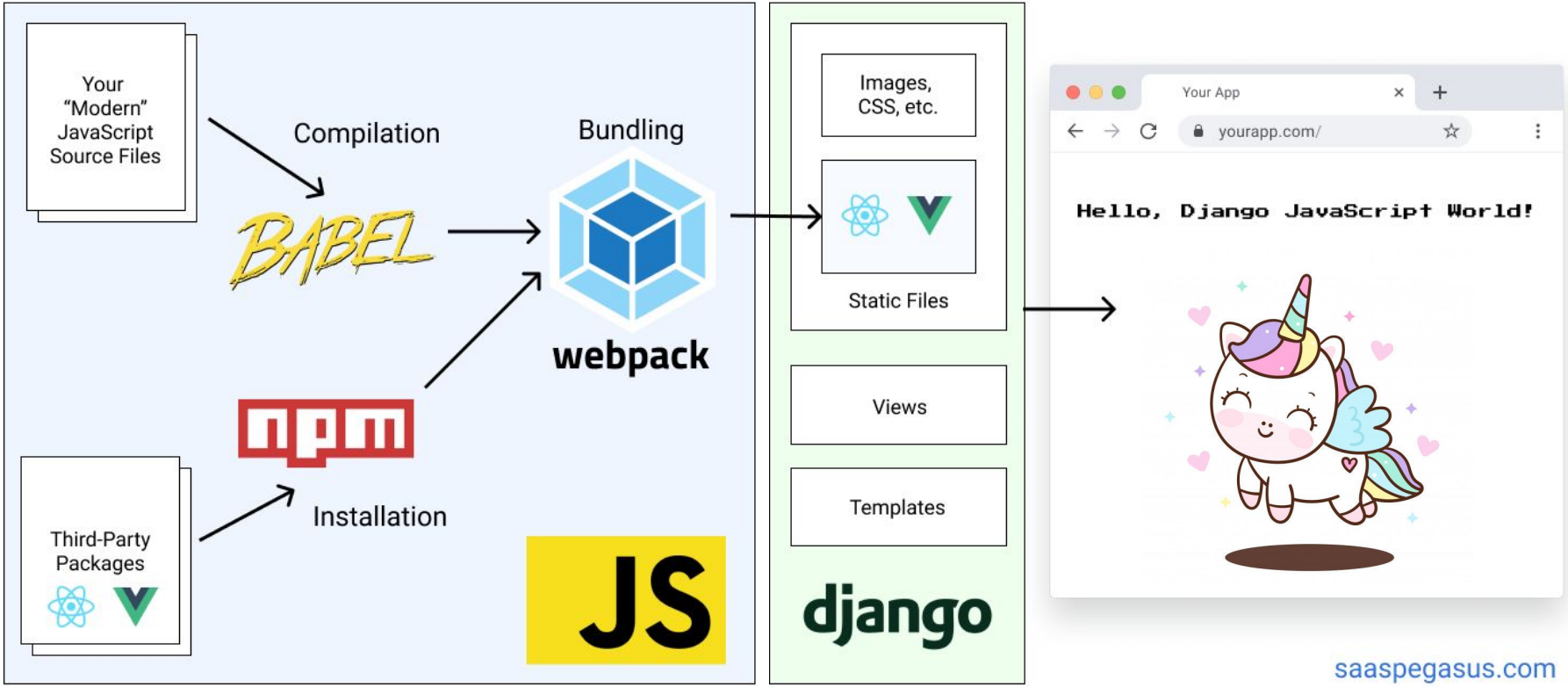
Babel compiles our code so we can use newer features and syntaxes and make them work in older browsers

Webpack bundles everything into a small number of JavaScript files to add to our pages



saaspegasus.com

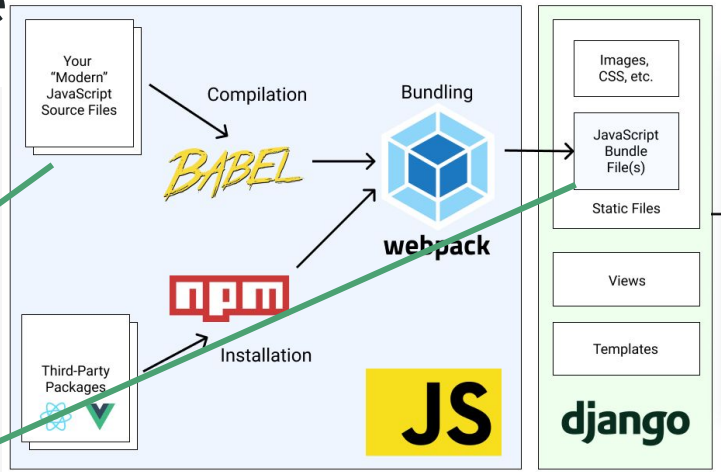
Using a Toolchain in a Django Hybrid Architecture

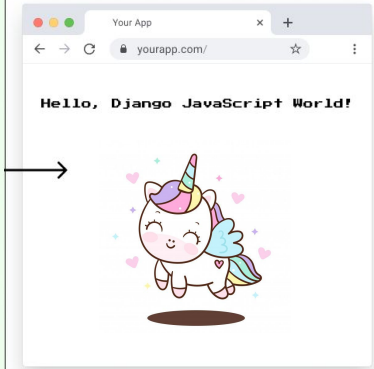


saaspegasus.com

Sample Project Structure

```
├── manage.py
├── mysite
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── myapp       <----- a normal Django app
│   ├── __init__.py
│   ├── models.py
│   ├── urls.py
│   └── views.py
├── assets      <----- our front-end project source
│   ├── javascript
│   └── styles
├── static      <----- our front-end project outputs (and other Django static files)
│   ├── css
│   ├── images
│   └── js
└── templates   <----- our Django template files
    └── myapp
```

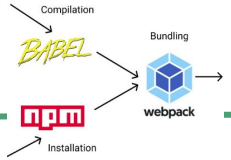




A “Single-Page” React App

React JavaScript file

```
import React from 'react';
import ReactDOM from "react-dom";

ReactDOM.render(
  <h1>Hello, react!</h1>,
  document.getElementById('root')
);
```



Django template file

```
{% load static %}
<!doctype html>
<html>
  <head>
    <title>Getting Started with Django and React</title>
  </head>
  <body>
    <div id="root" />
    <script src="{% static 'index-bundle.js' %}"></script>
  </body>
</html>
```

A much more comprehensive example of integrating a real single-page React app can be found in part 4 of the guide here: https://www.saaspegasus.com/guides/modern-javascript-for-django-developers/integrating-django-react/

3. Modern JavaScript: What’s the payoff?

What’s the payoff?

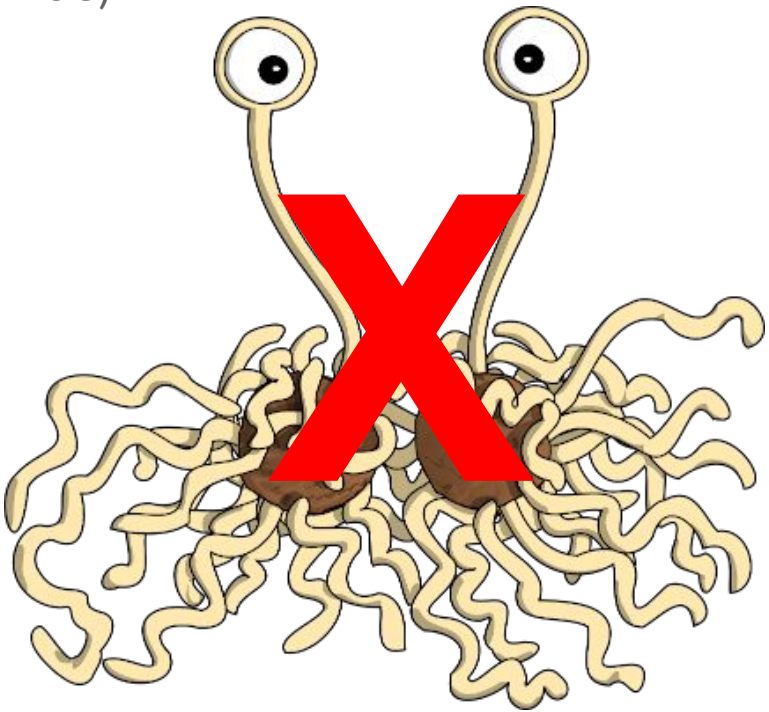
Use latest JavaScript frameworks (e.g. React / Vue)

Better dependency management

New features and convenient syntaxes

New language extensions (e.g. TypeScript)

Other front-end tooling (e.g. Sass)



ES6 (a.k.a. making JavaScript more like Python)

Modules

Then `module.js`:

```
export function hello() {
  return "Hello";
}
```

Then `main.js`:

```
import { hello } from './module.js';
let val = hello();  // val is "Hello";
```

Classes

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
}
mycar = new Car("Ford");
```

Arrow (lambda) functions

```
// ES5
var x = function(x, y) {
   return x * y;
}

// ES6
const x = (x, y) => x * y;
```

Template (F) Strings

```
// Simple string substitution
var name = "Brendan";
console.log(`Yo, ${name}!`);

// => "Yo, Brendan!"
```

Default Argument Values

```
function myFunction(x, y = 10) {
  // y is 10 if not passed or undefined
  return x + y;
}
myFunction(5); // will return 15
```

JSX

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);
```

Vue’s take

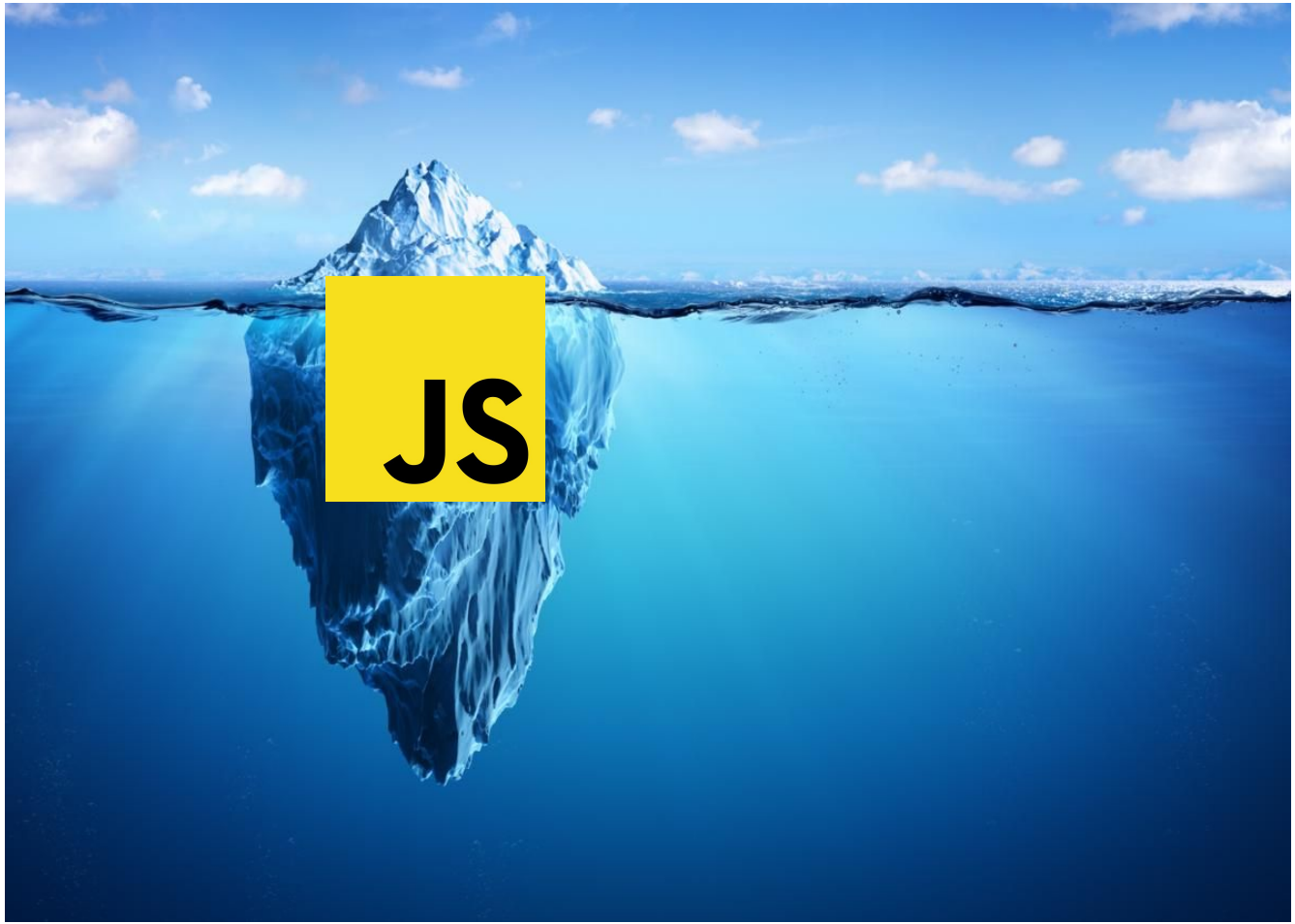
Combining templates, logic and styles into a single file

```
<template>
  <p>{{ greeting }} Vue!</p>
</template>

<script>
module.exports = {
  data: function() {
    return {
      greeting: "Hello"
    };
  }
};
</script>

<style scoped>
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

TypeScript

```
interface User {
  id: number
  firstName: string
  lastName: string
  role: string
}

function updateUser(id: number, update: Partial<User>) {
  const user = getUser(id)
  const newUser = {...user, ...update}
  saveUser(id, newUser)
}
```

And more!



Thank you!

For code samples and a complete write up on all of this go to **saaspegasus.com** and click “guides”

@czue | coryzue.com