
INTRASOFT
INTERNATIONAL
Enterprise Solutions
Architectures in VueJS Projects
05/03/2020

Who am I?

INTRASOFT INTERNATIONAL

- Lead Software Engineer @ Intrasoft International
- Research Fellow in University of Piraeus
- Core contributor in various open source projects (Vaadin Flow, Drupal)
- Ambassador of OracleJET and Prestashop

Email: panagiotis.adamopoulos@intrasoft-intl.com

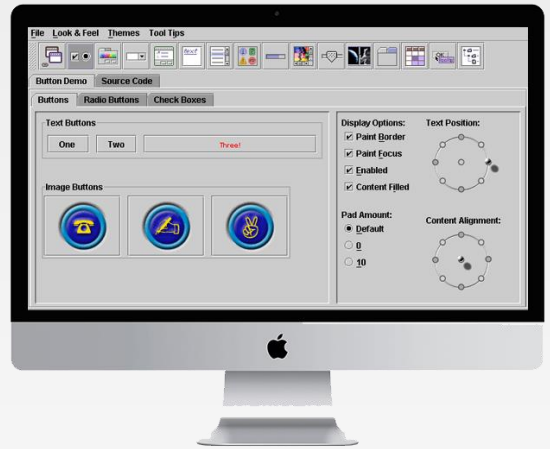
Github: https://github.com/panosadamop

Linkedin: https://www.linkedin.com/in/padamopoulos/

The web evolution

The web evolution

- HTML specification was made public in late 1991 by Tim Berners-Lee.
- CSS was proposed by Håkon Wium Lie in October 1994



1997 - 200x

- Static Sites, Javascript, VBScript
- Flash Websites
- Dialup and slow ADSL connections

Around 2007 - 2008

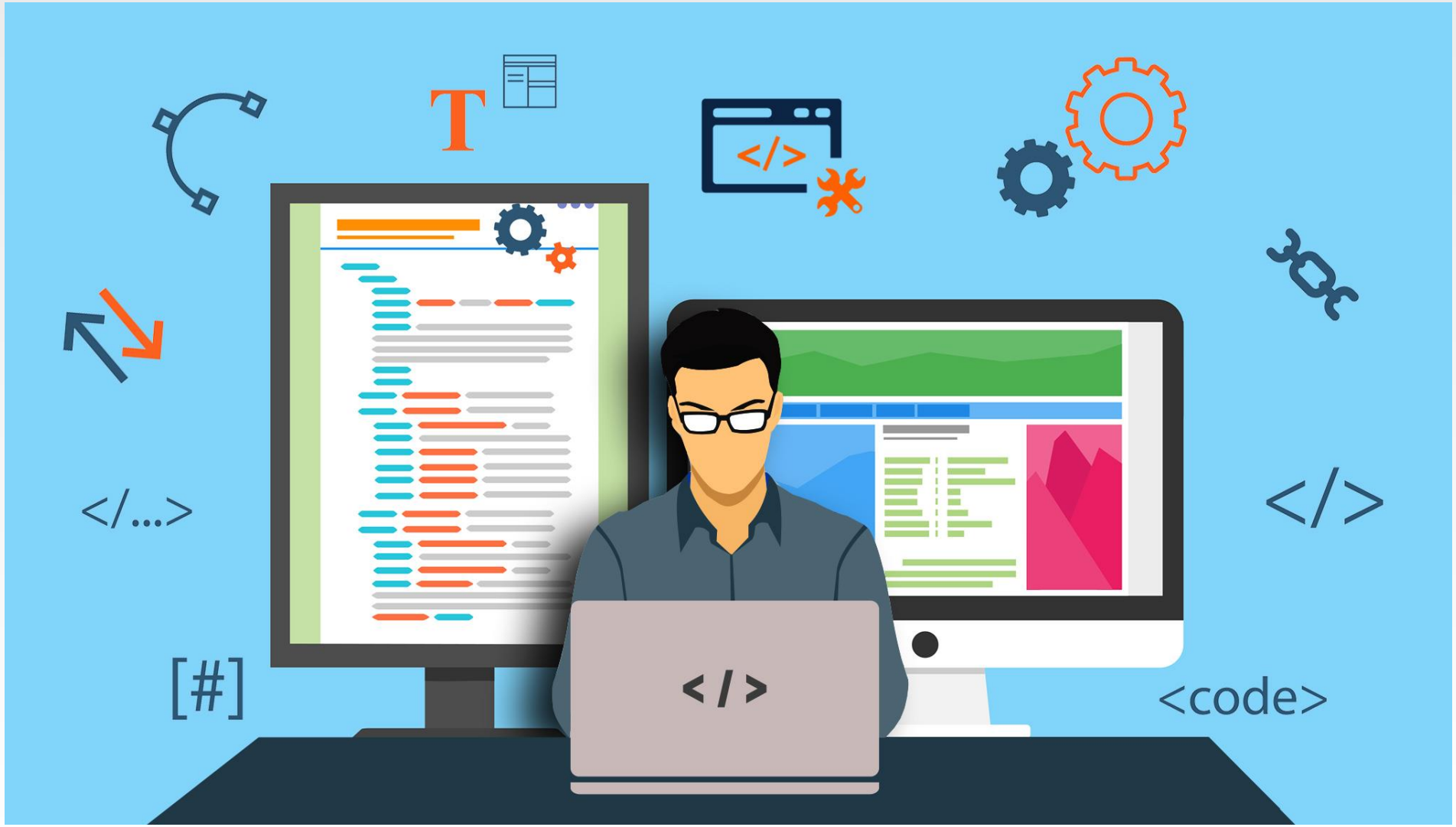
- Web application made a trend
- .NET, php, jsp, database usage, CMS
- Fast internet connections

Today

- Various Frameworks
- MicroServices & MicroApps
- Enterprise Web Applications

Web development... in late 90's

INTRASOFT INTERNATIONAL



Web development… in late 90's

INTRASOFT INTERNATIONAL

- Expectations were low for layout
- The idea of pixel-precision layout was nonsense - people cared about the content more than how it looked.
- Bandwidth was a huge deal, so imagery was small and modest.
- More complex stuff you could use CGI scripts written in PERL or C/C++.
- A CMS was a luxury and if used we built it ourselves!
- The output was just plain HTML the performance was great.
- The entirety of web technology could fit comfortably in our heads

Web development… in the Millennium

INTRASOFT INTERNATIONAL

- Project became more complex and web teams started to appear
- QA was part of the development process
- Agile & Scrum started being used in web and software development

Web development… Let’s start Divide and Rule

- Feature scaling and parallel implementation of features is close to impossible.
- Hard to maintain.
- Scalability, Testing, troubleshooting, Impact Analysis, Deployments etc.., became major issues.

Frontend development

Backend development

Web development… Let’s start Divide and Rule

INTRASOFT INTERNATIONAL

Impact and benefits

- Best suits for medium sized projects and many functionalities.
- Encourages API’s implementation.
- Still a consistent area of focus that accelerate Development speed.

Issues

- Feature scaling and parallel implementation of features is at-least possible in a scalable manner.
- Still hard to maintain.
- Scalability, Testing, troubleshooting, Impact Analysis, Deployments etc.., are still major issues.

Web development...

INTRASOFT INTERNATIONAL

Something happened mysteriously

Web development… Microservices

- Backend Team splits based on their focus area
- Frontend team still create “monolithic” UIs

Frontend development

Backend development

Web development… Microservices

INTRASOFT INTERNATIONAL

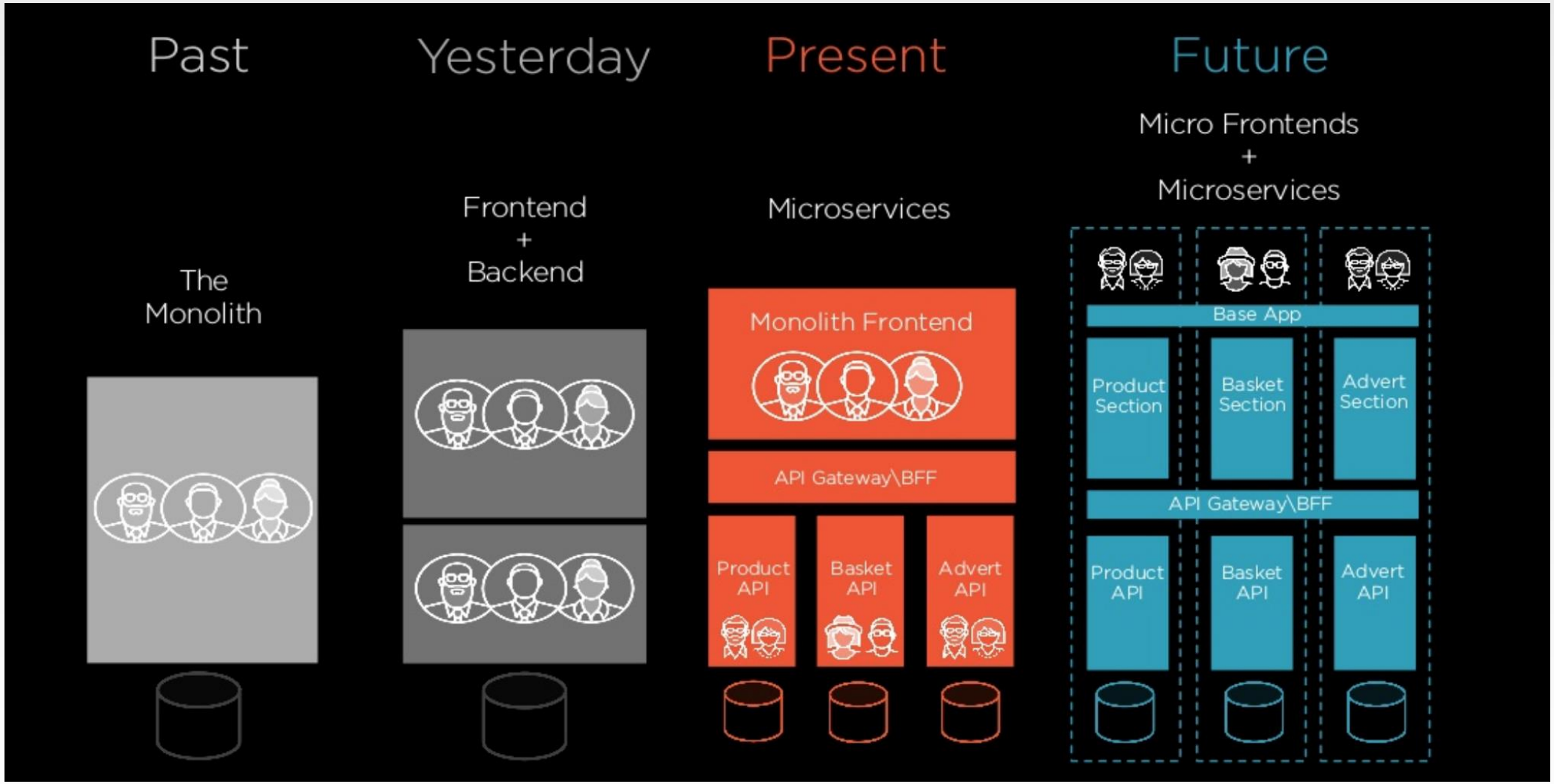
Impact and benefits

- Has all the benefits of previous approach like High speed of development, feature intensive, good documentation, well defined focus area etc..
- Best suits for large sized projects and many functionalities.
- We start understanding Scalability, Testing, troubleshooting, Impact Analysis, Deployments etc.., to much extent.

Issues

- Communication is challenging
- Product vision start deviating
- Lot’s and Lot’s of documentation is required.
- With the growing backend features, Frontend is still a monolith and becomes bottleneck of Time to Market.

Web development... Timeline



The Future

Web development... The Future

INTRASOFT INTERNATIONAL

PRINCIPLES

- Stand-alone run
- Independent development
- Stand-alone deployment

APPROACHES

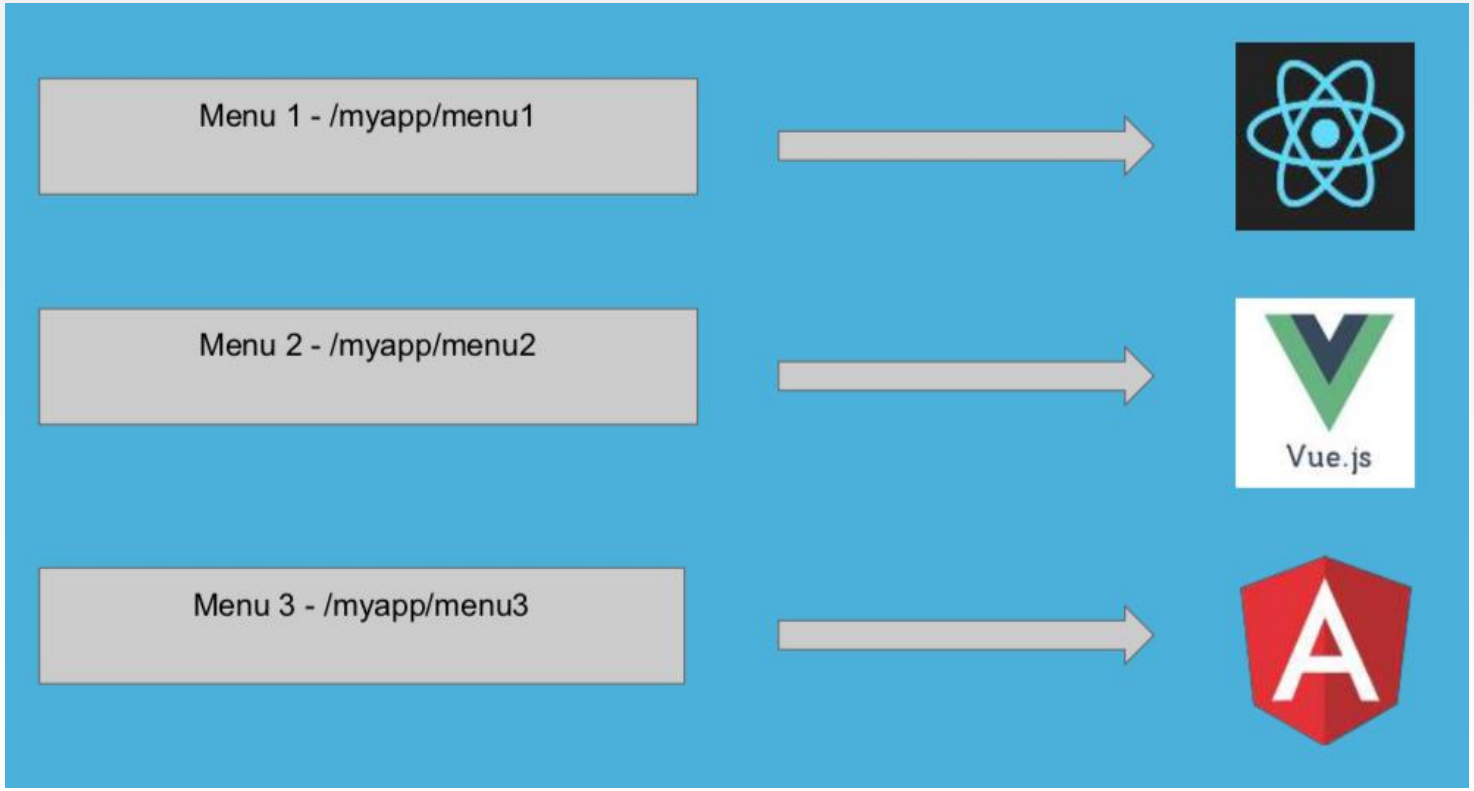
- Routing
- iFrame
- Micro-apps
- Pure components
- Web components

Web development... The Future

INTRASOFT INTERNATIONAL

Routing

- Each route is a different project
- Use HTTP server routing to redirect multiple apps
- Easy implementation and Independent development

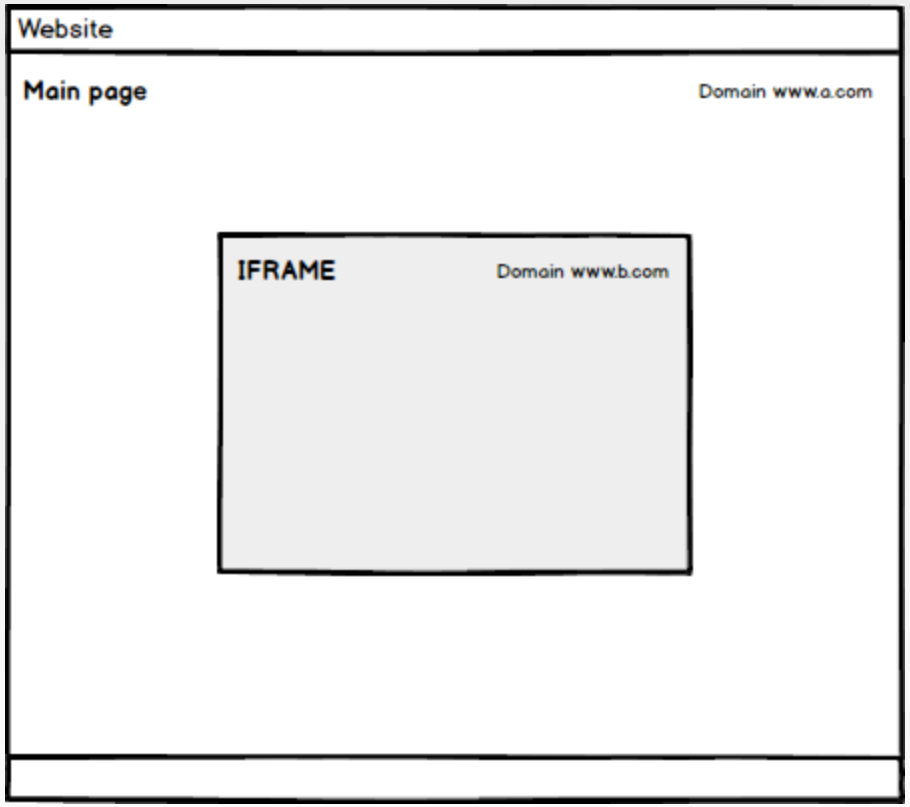


Web development... The Future

INTRASOFT INTERNATIONAL

iFrame

- Embedded modules and projects
- Easy to implement
- Complex to maintain
- Very useful in legacy projects



Web development... The Future

INTRASOFT INTERNATIONAL

Micro-Apps

- Fast distributed development
- Challenging integration
- Few reusable components
- Independent development

Web development… The Future

INTRASOFT INTERNATIONAL

Pure components

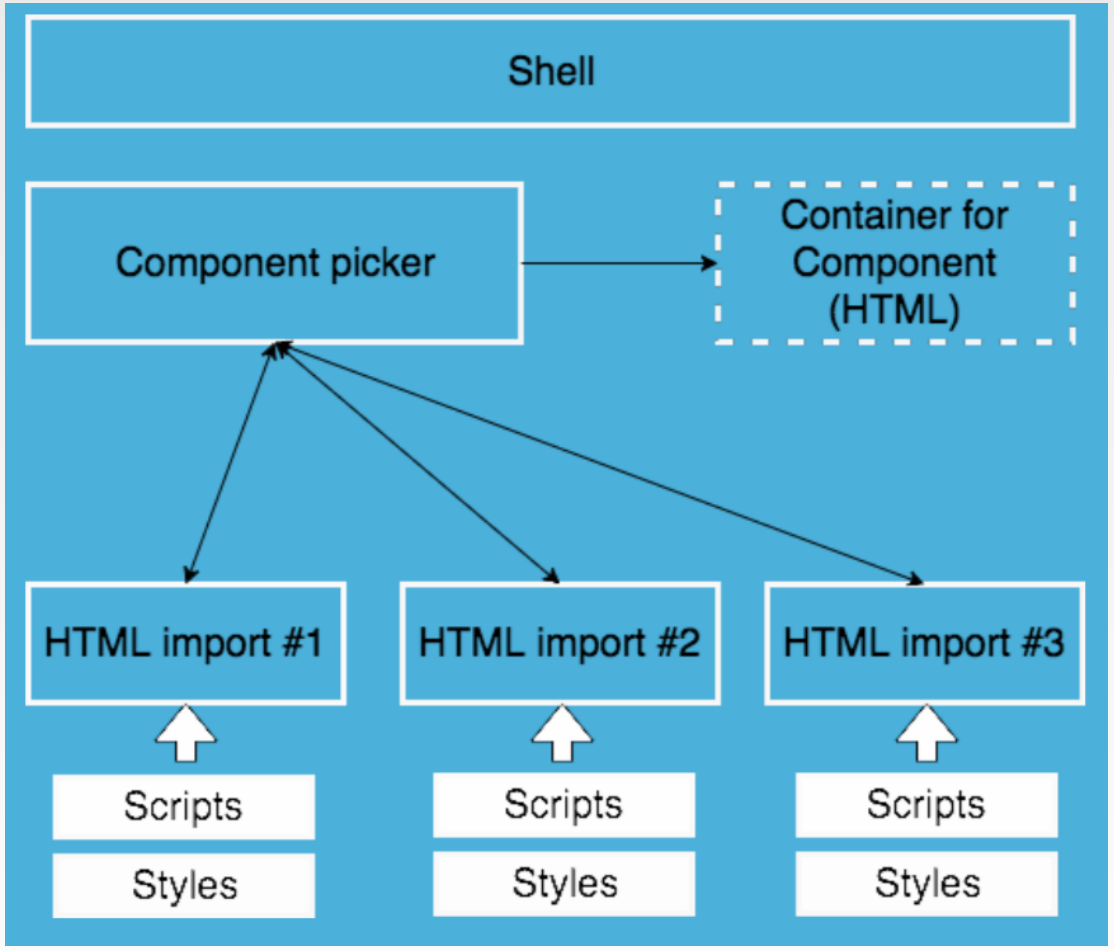
- Javascript functions
- Internal libraries
- Npm, unpkg, etc.
- Advanced JS knowledge is required

Web development… The Future

INTRASOFT INTERNATIONAL

Web components

- High complexity
- Teams must be coordinated
- Standard UI
- Communication between components is a big deal
- Advanced JS knowledge is required



Web development... The Result

INTRASOFT INTERNATIONAL

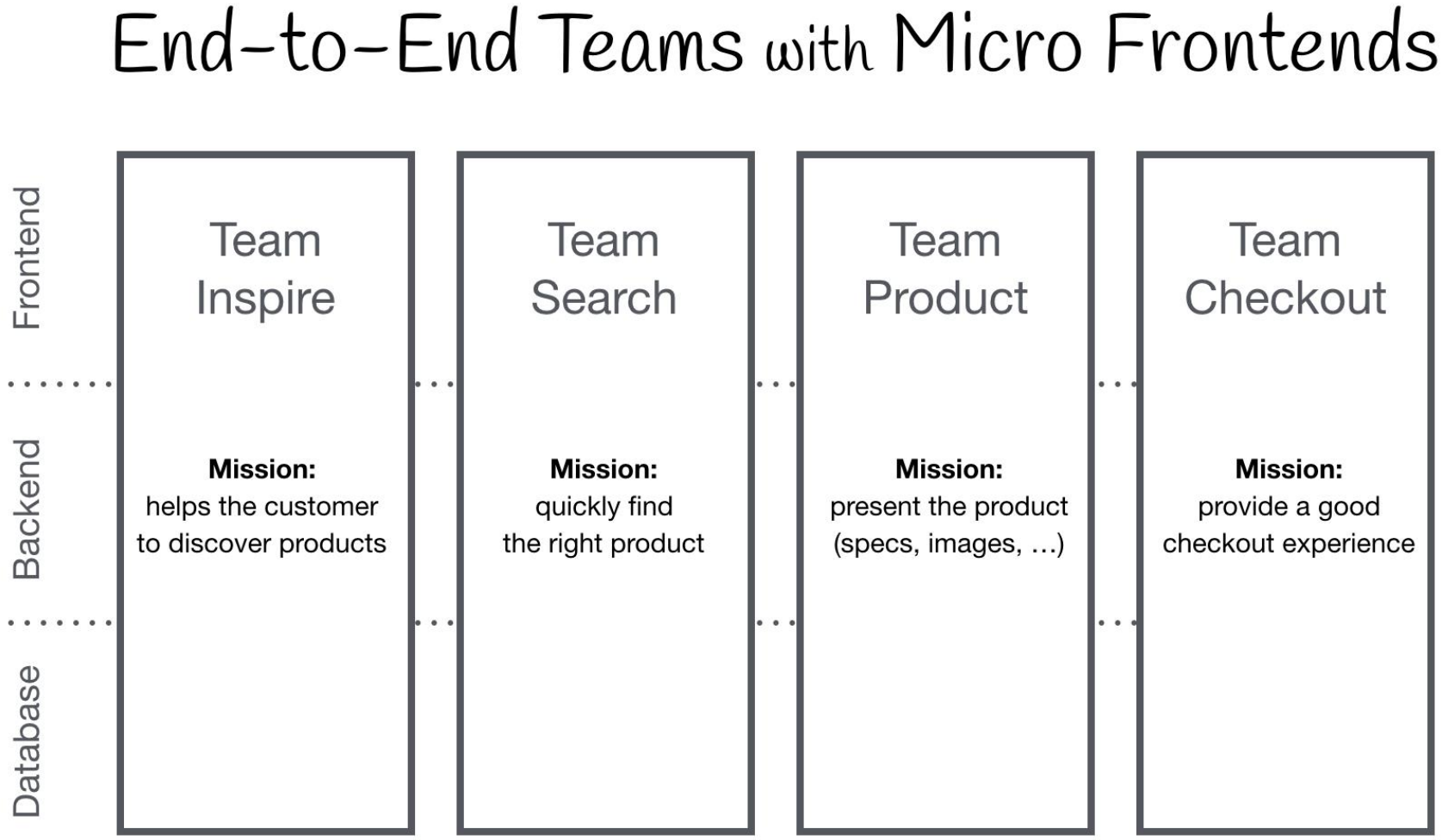
CHALLENGES

- The UI have to look and feel consistent
- Complex integration
- Initial down time can increase.
- Communication between services

BENEFITS

- Scale large application
- Use new technologies in a current production application. Independent development
- Independent deployment
- Reduce risk of bugs
- Easier testing
- Easy to integrate new members

Web development... The Result



Developer Life

Developer Life - Our clients

Use case

INTRASOFT INTERNATIONAL

- Let’s start a new project
- We want to use microservices
- We need an enterprise architecture
- We need high performance, fail-tolerance etc.
- We need a flexible and elegant UI/UX
- We need a mobile app as well

Use case

INTRASOFT INTERNATIONAL

- You know we have a few applications from another vendors that we need to integrate.
- You know some of these apps are legacy.
- You know we need to have single code base.
- You know we need the above for web, desktop, mobile etc.
- You know iFrames are anti-pattern.

Use case

Use case

Use case

INTRASOFT INTERNATIONAL

- Decide how to implement.
- Create High Level Designs.
- Select frameworks for implementation.
- Discuss our thoughts and decisions with teammates and managers.

Architect a large scale Vue.js Application

Architect a large-scale Application

INTRASOFT INTERNATIONAL

CHALLENGES

- how to structure the application as modular, flexible and scalable,
- how to handle http calls,
- how to do state management,
- how to add exception handling,
- how to add logging

Architect a large-scale Application

INTRASOFT INTERNATIONAL

MILESTONES

- Initial setup with zero configuration
- Do as much as possible with framework tools, like CLI (Vue CLI 3 in our case)
- Define application layers

It is useful to design frontend applications in 3 layers

UI layer — Application UI components
Business layer — Contains the application business logics, often called as services
Application state — Application state management

Architect a large-scale Application

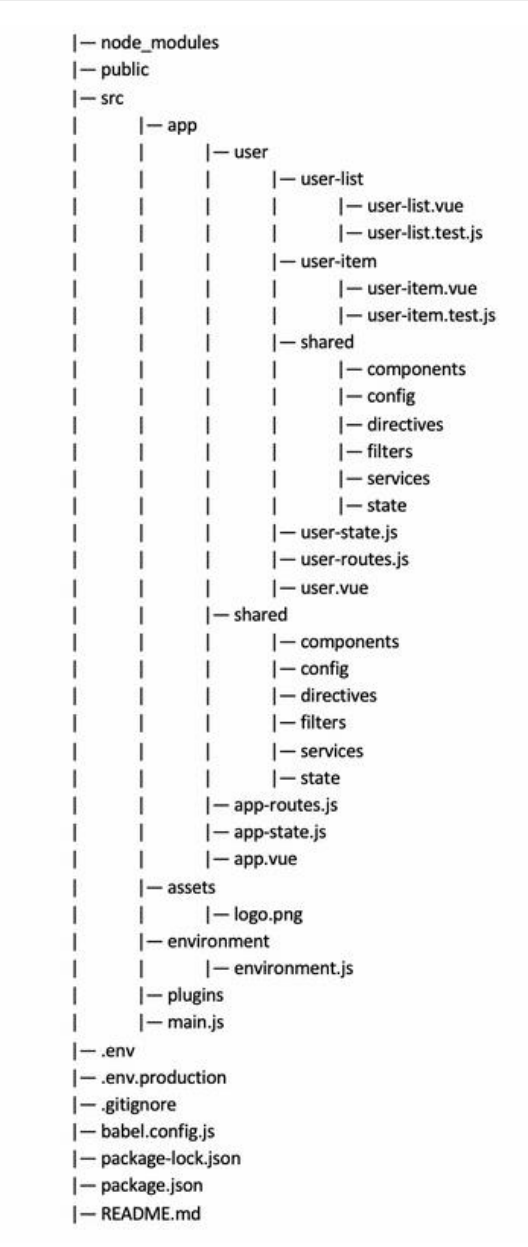
INTRASOFT INTERNATIONAL

There are many structure approaches, such as folder-by-type and folder-by-feature.

I prefer folder-by-feature approach.

On the root level, there will be folders/files created with Vue CLI.

Environment files (.env and .env.production) are recommended by Vue CLI for managing the different environment configurations in the application.



Architect a large-scale Application

INTRASOFT INTERNATIONAL

src folder:

In the src folder, there is an app folder which contains the main source code for the application and in addition, there will be assets, environment and main.js

assets:
contains the static assets/images.

environment:
It contains environment.js file. We can access environment variables using global process.env but this is error-prone. There should be common interface through which we can access/limit the access of environment variables.

plugins:
This will contains the Vue plugins.

main.js:
This file is responsible for bootstrapping the Vue application.

```
|— node_modules
|— public
|— src
|       |— app
|       |       |— user
|       |       |       |— user-list
|       |       |       |       |— user-list.vue
|       |       |       |       |— user-list.test.js
|       |       |       |— user-item
|       |       |       |       |— user-item.vue
|       |       |       |       |— user-item.test.js
|       |       |       |— shared
|       |       |       |       |— components
|       |       |       |       |— config
|       |       |       |       |— directives
|       |       |       |       |— filters
|       |       |       |       |— services
|       |       |       |       |— state
|       |       |       |— user-state.js
|       |       |       |— user-routes.js
|       |       |       |— user.vue
|       |       |— shared
|       |       |       |— components
|       |       |       |— config
|       |       |       |— directives
|       |       |       |— filters
|       |       |       |— services
|       |       |       |— state
|       |       |— app-routes.js
|       |       |— app-state.js
|       |       |— app.vue
|       |— assets
|       |       |— logo.png
|       |— environment
|       |       |— environment.js
|       |— plugins
|       |— main.js
|— .env
|— .env.production
|— .gitignore
|— babel.config.js
|— package-lock.json
|— package.json
|— README.md
```

Architect a large-scale Application

INTRASOFT INTERNATIONAL

app folder:

Here is the folder by feature. It will also contain application level shared components, config, directives, filters, services etc. Apart from these, app folder will contain 3 more files:

app-routes.js:
This file will be responsible for combining all the route configuration from feature modules. It will also have some app level routes such as wild-card routes.

app-state.js:
All the state management related configuration will be done in this file. For eg. Vuex related configuration will be done here. All the states from the different feature modules will be combined here and will be configured with the state management system.

app.vue:
This is the root component which will contain the main view of the application.

```
|— node_modules
|— public
|— src
|       |— app
|       |       |— user
|       |       |       |— user-list
|       |       |       |       |— user-list.vue
|       |       |       |       |— user-list.test.js
|       |       |       |— user-item
|       |       |       |       |— user-item.vue
|       |       |       |       |— user-item.test.js
|       |       |       |— shared
|       |       |       |       |— components
|       |       |       |       |— config
|       |       |       |       |— directives
|       |       |       |       |— filters
|       |       |       |       |— services
|       |       |       |       |— state
|       |       |       |— user-state.js
|       |       |       |— user-routes.js
|       |       |       |— user.vue
|       |       |— shared
|       |       |       |— components
|       |       |       |— config
|       |       |       |— directives
|       |       |       |— filters
|       |       |       |— services
|       |       |       |— state
|       |       |— app-routes.js
|       |       |— app-state.js
|       |       |— app.vue
|       |— assets
|       |       |— logo.png
|       |— environment
|       |       |— environment.js
|       |— plugins
|       |— main.js
|— .env
|— .env.production
|— .gitignore
|— babel.config.js
|— package-lock.json
|— package.json
|— README.md
```

Feature Module

INTRASOFT INTERNATIONAL

- Application folder structure is divided based on the features so that feature is called as feature module.
- Root of the feature module will be having the components along with their test and a shared folder.
- Each feature module will also be having its routes, state, and main component file.

e.g. In user feature module,

- user-list and user-item are components
- user-routes.js contains the user feature module routes
- user-state.js will combine the all user states
- user.vue is the main component of the user feature module.

Shared Folder

INTRASOFT INTERNATIONAL

- One shared folder will be on app level

- Each feature module will also have their own shared folder.

The key difference between them is that, app level shared folder will contain the application level shared things and module level shared folder will contain the module level shared things.

The shared folder contains the following subfolders:

- components
- config
- directives
- filters
- services
- state

Shared Folder

INTRASOFT INTERNATIONAL

components:
This folder will contain the shared components.

config:
This folder will contain the configuration such as api related config, configuration constants etc.

directives:
This folder will contain the shared directives.

filters:
This folder will contain the shared filters.

services:
Services will contain the business logic or utilities or http calls that is independent of UI logic. Most people call them utils or helpers or services.

state:
App level shared folder will contain the state that will be sharable between different feature modules.

Microfrontends with Vue.js

What are Micro-frontends?

INTRASOFT INTERNATIONAL

Think about a app as a composition of features owned by independent teams.

Each team has a specific area of business or mission it cares about and specializes in.

A team is cross-functional and develops features end-to-end, from the database to the user interface

Design Principles

INTRASOFT INTERNATIONAL

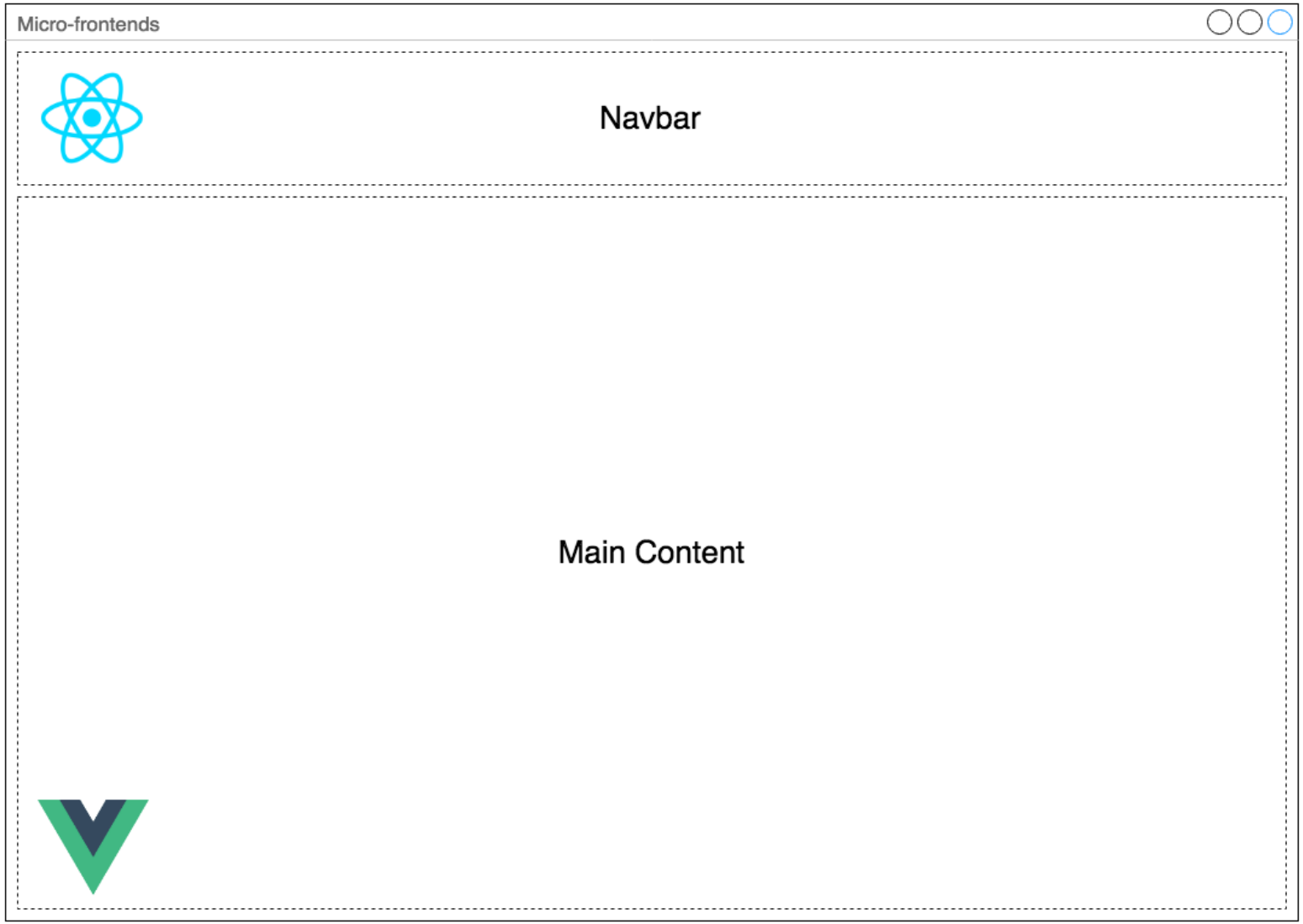
Team ownership
An entire team is responsible to develop a set of features that belong to a specific business domain, including developing, testing and deploy process.

Tech agnostic
The team can choose any framework like Vue.js or React.js regardless of what's using the other ones.

Favor native browser features over custom APIs
Use Browser Events for communication instead of building a global PubSub system. If you really have to build a cross-team API, try keeping it as simple as possible.

Resilient site
Your feature should be useful, even if JavaScript failed or hasn't executed yet. Use Universal Rendering and Progressive Enhancement to improve perceived performance.

Micro-frontends using Vue.js, React.js, and Hypernova



Both are Isomorphic Web Applications so the views are server-side rendered and client-side hydrated in order to make them client-side dynamic.

Reference
https://medium.com/@ElyseKoGo/an-introduction-to-isomorphic-web-application-architecture-a8c81c42f59

Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL

WHAT IS HYPERNOVA?

- A service for server-side rendering JavaScript views.
- It is originally developed by AirBNB.
- Has many clients like hypernova-react

TERMINOLOGY

- **hypernova/server** - Service that accepts data via HTTP request and responds with HTML.
- **hypernova** - The universal component that takes care of turning your view into the HTML structure it needs to server-render. On the browser it bootstraps the server-rendered markup and runs it.
- **hypernova-${client}** - This can be something like hypernova-ruby or hypernova-node. It is the client which gives your application the superpower of querying Hypernova and understanding how to fallback to client-rendering in case there is a failure.

Source: https://github.com/airbnb/hypernova

Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL

The project contains three parts:

- **hypernova-server-vue:** contains the Product List component built using Vue.js.
- **hypernova-server-react:** contains the Navbar component built using React.js
- **hypernova-aggregator:** requests views to the hypernova servers providing data and composes all the views inside an HTML Document.

CSS files are distributed by the aggregator because it’s usually responsible for the branding and look and feel.

Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL

Hypernova Vue

- We have inside hypernova-server-vue an entry point for our hypernova server on src/index.js
- We can run it using **yarn dev** or **npm run dev**

Index.js

```
import hypernova from 'hypernova/server'
import { renderVue, Vue } from 'hypernova-vue'
import express from 'express'
import path from 'path'

import ProductList from './components/ProductList.vue'

hypernova({
  devMode: true,
  getComponent (name, context) {
    if (name === 'ProductList') {
      return renderVue(name, Vue.extend(ProductList))
    }
  },
  port: 3030,

  createApplication () {
    const app = express()

    app.use(express.static(path.join(process.cwd(), 'dist')))

    return app
  }
})
```

Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL

The getComponent provides the right component decorating it with the renderVue function of hypernova-vue.

It's a simple vue component that receives props like title, items but as you can see it has a method called select that creates a CustomEvent and dispatches it using the document reference.

The mechanism to communicate micro-frontends is pretty similar to the micro-services do except micro-frontends use the DOM API as pub/sub*.

*Source: https://aws.amazon.com/pub-sub-messaging/

ProductList.vue

```vue
<template>
  <div class="k-product-list">
    <h2 class="k-product-list__header">{{title}}</h2>
    <ul>
      <li v-for="(item, idx) in items" :key="idx" class="k-product-item" @click="select(item)">
        <img :src="item.imageUrl" class="k-product-item__image">
        <h4 class="k-product-item__title">{{ item.title }}</h4>
      </li>
    </ul>
    <button @click="addItem">Add Item</button>
  </div>
</template>

<script>
export default {
  props: {
    title: {
      type: String,
      required: true
    },
    items: {
      type: Array,
      default: () => []
    }
  },
  methods: {
    addItem() {
      this.items.push({ title: `Product ${this.items.length + 1}`, imageUrl: 'https://via.placeh
    },
    select(item) {
      const event = new CustomEvent('itemSelected', { detail: item });
      document.dispatchEvent(event)
    }
  }
}
</script>
```

Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL

The client.js script is our entry point for client-side hydration.

client.js

```
1  import { renderVue, Vue } from 'hypernova-vue'
2  import ProductList from './components/ProductList.vue'
3
4  renderVue('ProductList', Vue.extend(ProductList))()
```

Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL

Hypernova React

- We have inside hypernova-server-react a similar set-up as we have on hypernova-server-vue
- We can run it using **yarn dev** or **npm run dev**

Index.js

```
1  import express from 'express';
2  import path from 'path';
3  import hypernova from 'hypernova/server';
4  import { renderReact } from 'hypernova-react';
5
6  import Header from './components/Header';
7
8  hypernova({
9    devMode: true,
10   getComponent(name) {
11     if (name === 'Header') {
12       return renderReact(name, Header);
13     }
14
15     return null;
16   },
17   port: 3031,
18
19   createApplication() {
20     const app = express();
21
22     app.use(express.static(path.join(process.cwd(), 'dist')));
23
24     return app;
25   },
26 });
```

Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL

As you can see the entry point is pretty similar to the one but it uses renderReact function of hypernova-react instead.

The Header.jsx component subscribes to the itemSelected event in the componentDidMount method

Every time a itemSelected event is dispatched it will increment the itemsSelected variable on the state and store the last item selected.

Header.jsx

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      itemsSelected: 0,
      item: null,
    };
    this.itemSelected = this.itemSelected.bind(this);
  }

  componentDidMount() {
    document.addEventListener('itemSelected', this.itemSelected);
  }

  componentWillUnmount() {
    document.removeEventListener('itemSelected', this.itemSelected);
  }

  itemSelected({ detail: item }) {
    const { itemsSelected } = this.state;
    this.setState({ itemsSelected: itemsSelected + 1, item });
  }

  render() {
    const { title, links } = this.props;
    const { itemsSelected, item } = this.state;
    return (
      <header className="k-header">
        <div className="k-header__brand">{title}</div>
        { item && <span className="k-header__last-item">{`Last Item: ${item.title}`}</span> }
        <span className="k-header__space" />
        <span>{`Items Clicked: ${itemsSelected}`}</span>
        <NavBar links={links} />
      </header>
    );
  }
}

Header.propTypes = {
  title: PropTypes.string.isRequired,
  links: NavBar.propTypes.links,
};

Header.defaultProps = {
  links: [],
};

export default Header;
```

Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL

The client.js script is our entry point for client-side hydration.

client.js

```
1  import { renderReact } from 'hypernova-react';
2
3  import Header from './components/Header';
4
5  renderReact('Header', Header)();
```

Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL

Hypernova Aggregator

- The Aggregator is a basic express application that uses axios to request the views to the hypernova servers and putting the result HTML in a template using interpolation, as well as, adding the necessary client-side scripts from each micro-frontend.
- We can run it using **yarn dev** or **npm run dev**
- http://localhost:8080/

Index.js

```
const express = require('express')
const path = require('path')
const { getContent } = require('./services/content')
const { getLayout } = require('./services/layout')

const app = express()

app.use(express.static(path.join(__dirname, 'public')))

app.get('/', async (req, res) => {
  const promises = [getContent(), getLayout()]
  const [content, { header }] = await Promise.all(promises)
  const html = `
    <!DOCTYPE html>
    <html lang="en">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <meta http-equiv="X-UA-Compatible" content="ie=edge">
      <title>Document</title>
      <link rel="stylesheet" href="/style.css">
    </head>
    <body>
      ${header.html}
      ${content.html}
      <script src="http://localhost:3031/client.js"></script>
      <script src="http://localhost:3030/client.js"></script>
    </body>
    </html>
  `

  return res.send(html)
})

app.listen(8080, () => console.log('Aggregator Running'))
```

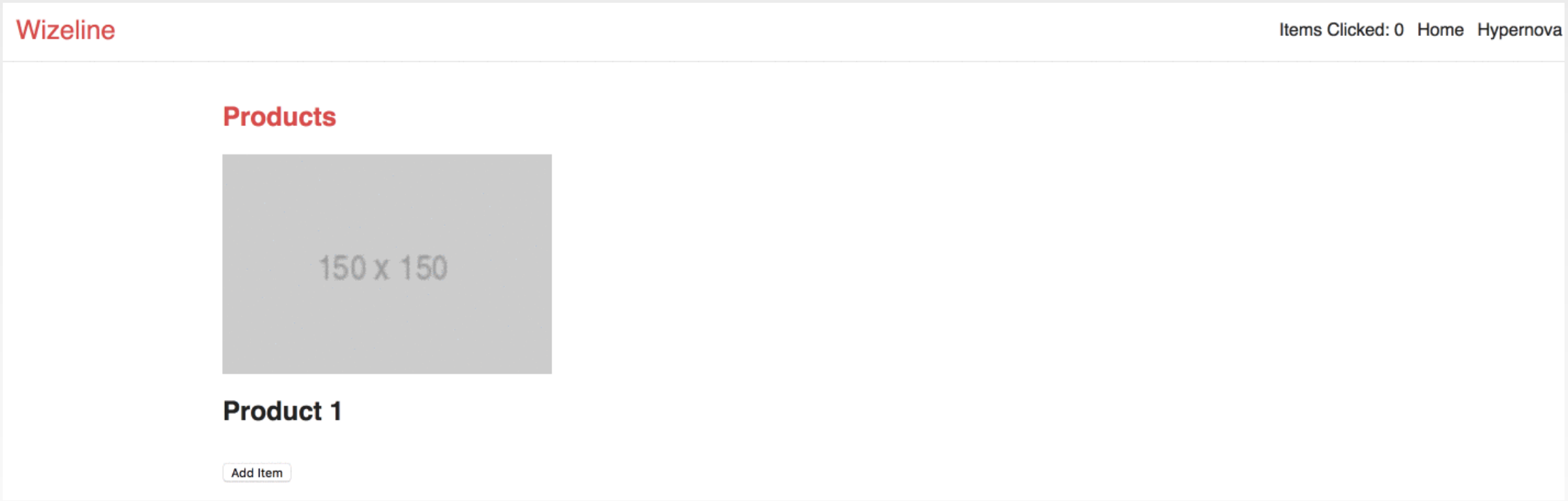
Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
  <link rel="stylesheet" href="/style.css">
</head>
<body>
  <div data-hypernova-key="Header" data-hypernova-id="ad34216c-c5fc-47ff-9000-0632cd3031ba"><header class="k-header" data-reactroot=""><div class="k-header__brand">Wizeline</div><span class="k-header__space"></span><span>Items Clicked: 0</span><nav class="k-navbar"><ul><li class="k-navbar__item"><a href="/">Home</a></li><li class="k-navbar__item"><a href="https://github.com/airbnb/hypernova">Hypernova</a></li></ul></nav></header></div>
<script type="application/json" data-hypernova-key="Header" data-hypernova-id="ad34216c-c5fc-47ff-9000-0632cd3031ba"><!--{"title":"Wizeline","links":[{"url":"/","text":"Home"},{"url":"https://github.com/airbnb/hypernova","text":"Hypernova"}]}--></script>
  <div data-hypernova-key="ProductList" data-hypernova-id="2d188e2b-a59b-4d4f-ba81-2218983270a2"><div data-server-rendered="true" class="k-product-list"><h2 class="k-product-list__header">Products</h2> <ul><li class="k-product-item"><img src="https://via.placeholder.com/150" class="k-product-item__image"> <h4 class="k-product-item__title">Product 1</h4></li></ul> <button>Add Item</button></div></div>
<script type="application/json" data-hypernova-key="ProductList" data-hypernova-id="2d188e2b-a59b-4d4f-ba81-2218983270a2"><!--{"title":"Products","items":[{"title":"Product 1","imageUrl":"https://via.placeholder.com/150"}]}--></script>
  <script src="http://localhost:3031/client.js"></script>
  <script src="http://localhost:3030/client.js"></script>
</body>
</html>
```

Micro-frontends using Vue.js, React.js, and Hypernova

INTRASOFT INTERNATIONAL



Server Side Rendering
vs
Client Side Rendering

SSR - CSR

INTRASOFT INTERNATIONAL

Server-Side Rendering:

- A user makes a request to a webpage
- The server prepares an HTML page by fetching user-specific data and sends it to the user's machine over the internet.
- The browser then construes the content and displays the page. This entire process of fetching data from the database, creating an HTML page and sending it to client happens in mere milliseconds.

- A user clicks a link on the page
- The browser sends a request to the server and the entire process is carried out by the server again.
- This process not only increases the load on the server but also consumes unnecessary internet bandwidth.

SSR - CSR

INTRASOFT INTERNATIONAL

Client-Side Rendering:
Client-side rendering is a reasonably new approach to rendering websites, and it didn't really become popular until JavaScript libraries started incorporating it.

- It is about rendering content in the browser using JavaScript.
- So instead of getting all the content from the HTML document itself, a bare-bones HTML document with a JavaScript file in initial loading itself is received, which renders the rest of the site using the browser.
- With client-side rendering, the initial page load is naturally a bit slow.
- After that, every subsequent page load is very fast.
- Communication with server happens only for getting the run-time data.
- There is no need to reload the entire UI after every call to the server.
- The client-side framework manages to update UI with changed data by re-rendering only that particular DOM element.

SSR - CSR

INTRASOFT INTERNATIONAL

Server-side pros:
- Search engines can crawl the site for better SEO.
- The initial page load is faster.
- Great for static sites.

Server-side cons:
- Frequent server requests.
- An overall slow page rendering.
- Full page reloads.
- Non-rich site interactions.

Client-side pros:
- Rich site interactions
- Fast website rendering after the initial load.
- Great for web applications.
- Robust selection of JavaScript libraries.

Client-side cons:
- Low SEO if not implemented correctly.
- Initial load might require more time.
- In most cases, requires an external library.

References

INTRASOFT INTERNATIONAL

- https://micro-frontends.org/
- https://github.com/chrisvfritz/vue-enterprise-boilerplate
- https://dev.to/itnext/learn-how-you-can-build-enterprise-vue-js-applications-with-nuxt-5fp4
- https://innovation.enova.com/building-an-embeddable-micro-frontend-with-vue-js/
- https://www.trysmudford.com/blog/microfrontends-with-vue-js/

INTRASOFT INTERNATIONAL

Thank you for your attention

Follow us