React Front-end Implementation Plan

About this document

[Placeholder]

Reference documents

- https://patternlab.io/

- https://github.docusignhq.com/WWW/gibson/wiki
- http://cssinjs.org/benefits
- https://emotion.sh
- https://storybook.js.org/

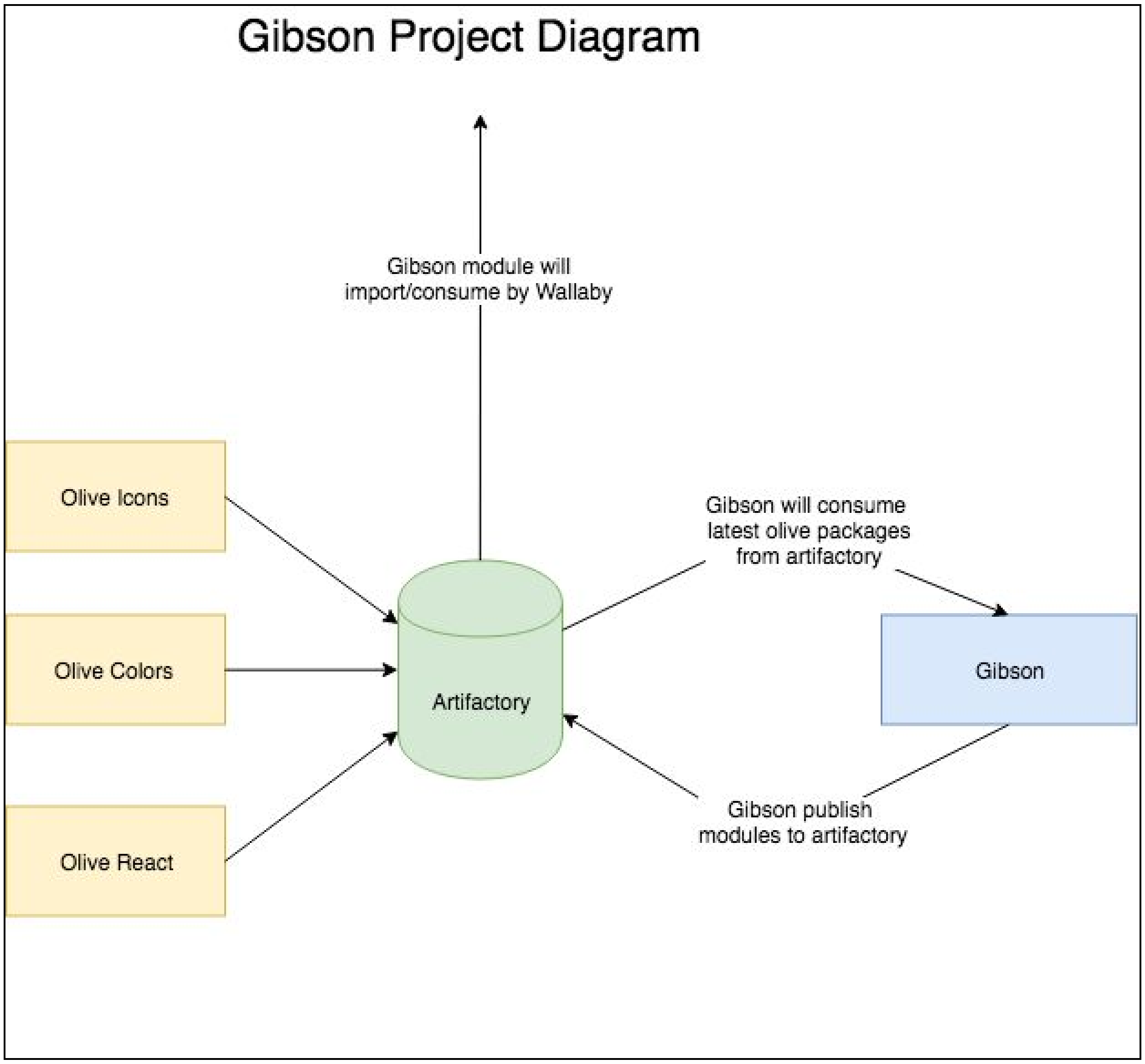
Implementation plan

Storybook

Storybook is a way to play with our front-end Gibson components, wIthout the Drupal backend.

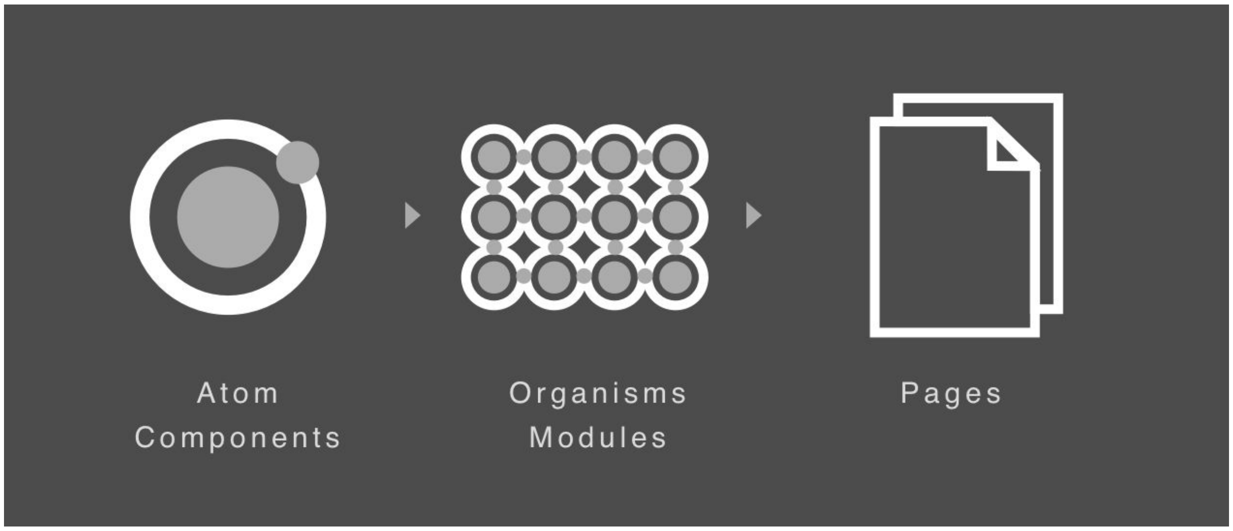
Front-End (Gibson)

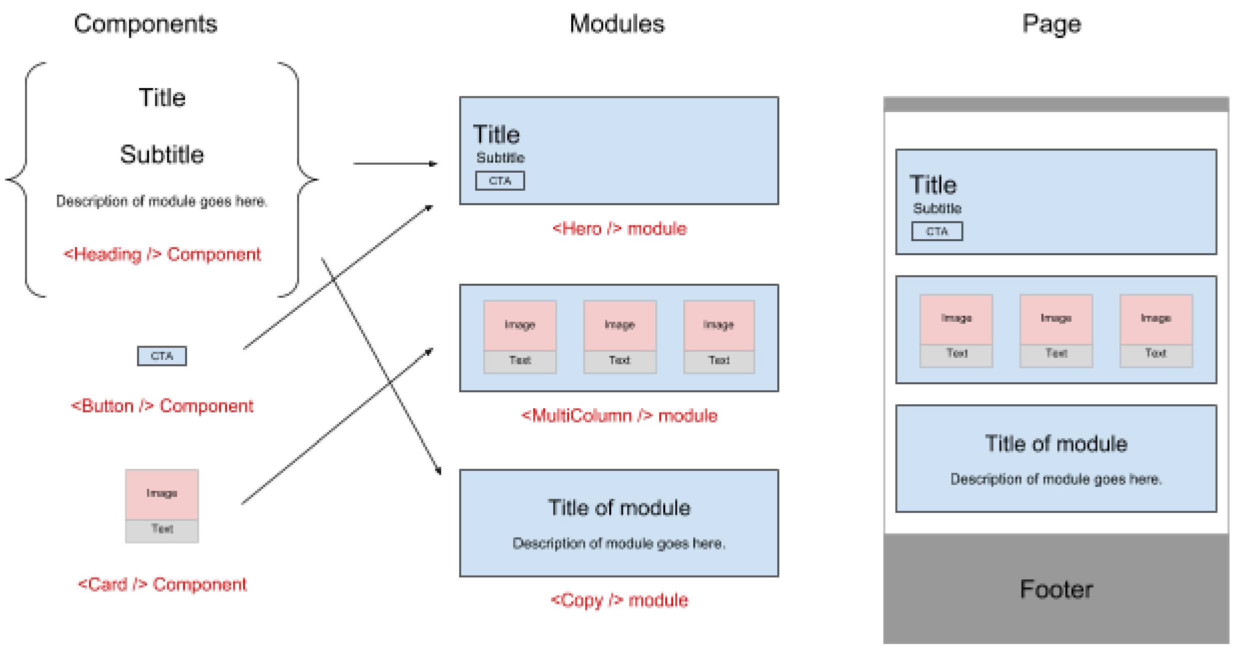
Gibson is React modular library for DocuSign web properties. Gibson will publish using npm on Artifactory and will be consumed by other project using es6 imports.


Gibson Project Diagram
Gibson module will import/consume by Wallaby
Olive Icons
Olive Colors
Olive React
Artifactory
Gibson will consume latest olive packages from artifactory
Gibson
Gibson publish modules to artifactory

React-Components structure

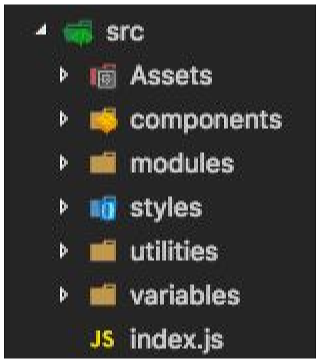
- Review components and modules requirement documentation at https://github.docusignhq.com/WWW/gibson/wiki
- It will use atom(components) → Organisms(Modules) → Pages(Web Pages) analogy to build web pages.
- Gibson react project will build reusable modules using components.





Directory structure

- Gibson src will have components/modules/assets directory structure.

- 

```
src
  Assets
  components
  modules
  styles
  utilities
  variables
  index.js
```

Components

- Components are basic atoms which will be use to build Gibson modules.
- For Example: Heading, Button, Section, Card, and, more
- DO NOT PUBLISH THIS COMPONENTS while you build Gibson.

Assets

- This directory will hold Images, Fonts, Icons, Videos to build modules locally for storybook.
- DO NOT PUBLISH THIS ASSETS while you build Gibson.

Modules

- Modules are reusable react component.
- Gibson will publish modules to npm artifactory
- For Example: Hero, Multicolumn, Copy

Pages

- “Emmet” compiler will use Gibson modules to build web pages.

Styling method

- Use CSS in JS, JSS styling method for Gibson css styling.
- Why JSS ?
    - JSS is a more powerful abstraction over CSS.
    - It uses JavaScript as a language to describe styles in a declarative and maintainable way.
    - It is a high performance JS to CSS compiler which works at runtime and server-side.
    - This core library is low level and framework agnostic.

    - More...
- Using https://emotion.sh as CSS-in-JS library.
- Olive, Martini also use emotion js as solution

Component Previewer Site

- Storybook : https://storybook.js.org/
- Publish Storybook on Git-Pages
- You can review published Gibson site at : https://github.docusignhq.com/pages/WWW/gibson/

API integration

-

State management

[Placeholder-State Tree diagram? Redux information?]

Routing

Code splitting

[Placeholder]

Server side rendering

[Placeholder]

Quality assurance

Here are QA observations on existing automation test scripts.

- Any url change or navigation change automation test scripts should be updated.
- As long as the existing unique ids for the text fields/web elements are not altered, automation test scripts should be working good.

- When any new text field is added or any existing text field is removed, its required to update the same in the test scripts.
- There are few fields/elements which do not have unique ids for which I have used Xpath based on the location of the element, these elements will require an update if there is any change in the dom.

At this moment I do not see any more scenarios that will require to modify existing automation scripts when we use React for Front-end development.

Open questions

Is Gatsby better than rolling a custom front-end?
Gatsby would overlap what is provided with the internal olive/react front end.
Olive and our variant, Gibson, would conform more strictly to DocuSign standards for both design, code, and accessibility.

Is create-react-app sufficient or do we need to maintain our own webpack config?

- Create our own static site compiler “Emmet” or use Gatsby as static site generator.

Future phases

- ECOM, DevCenter, Blog will use Gibson modules for unique web experience for customer.