Let's Start!

CarRadar®

CarRadar®

Mar 17, 2022

Designing the App with scale

Agenda

1. Who I am
2. Yesterday and Today
3. What is SPA
4. Why React
5. App Controller
6. Architecture
7. DALs
8. Pros and Cons
9. CI/CD + Automation
10. Backend
11. Legends

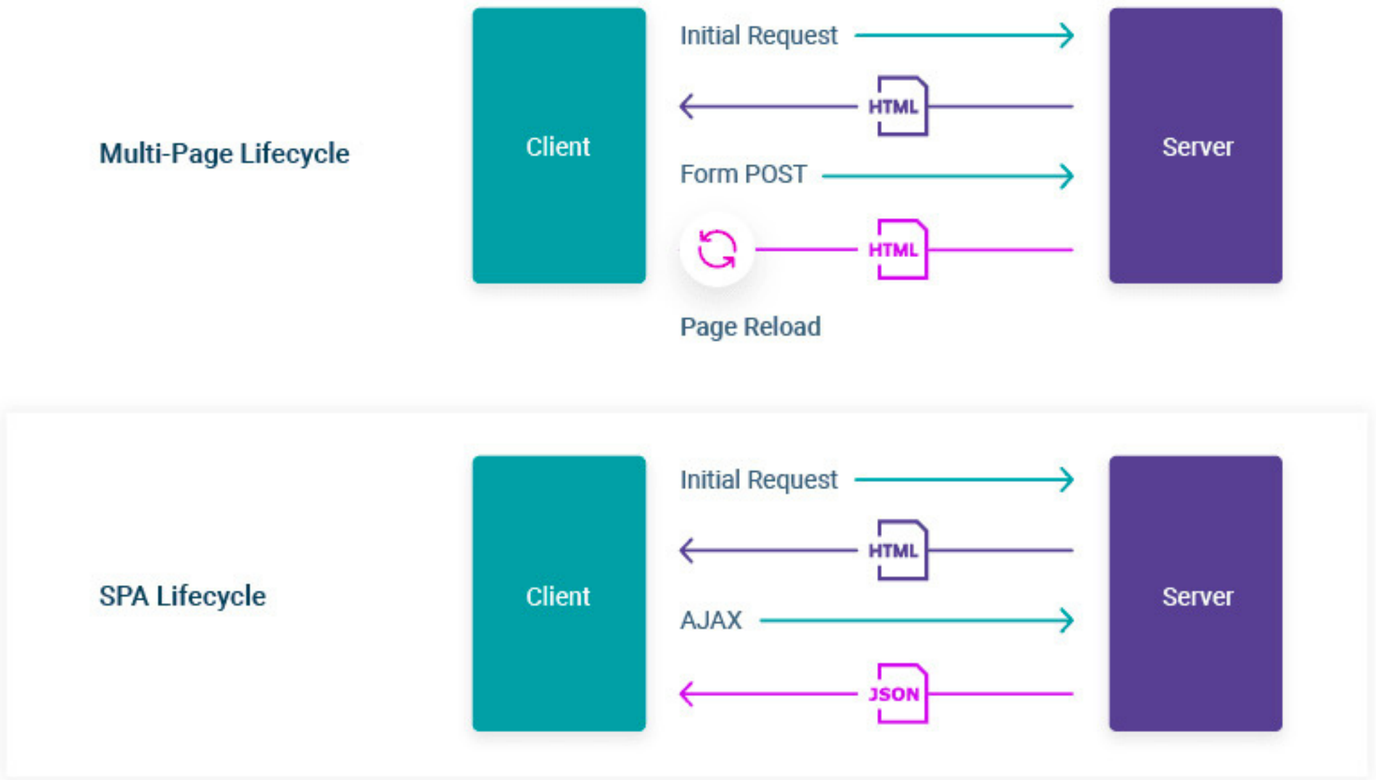
What is SPA

What is SPA

A single-page application is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the webserver, instead of the default method of a web browser loading entire new pages.

This, therefore, allows users to use websites without loading whole new pages from the server, which can result in performance gains and a more dynamic experience.

With a SPA the new data doesn't require the entire page to be re-rendered; instead, only new pieces of data are requested through an API, and are refreshed on the web page.



Who am I?

Sviatoslav Kuzhelev

position: Front-End Developer; rank: Senior/Lead.

What I do:
- create awesome Front-end things;
- scale up FE codebase;
- isolated React Development;

What I do OOO:
- write Dev articles on Medium;
- contribute to open-source;
- hike, spartan, travels all the year;

@BiosBoy

sviat-kuzhelev.medium.com

@svyat770

Favorite books:

"Code Complete: *A Practical Handbook of Software Construction*",
- by Steve McConnell

"You Don't Know JS: *series of books*"
- by Kyle Simpson

Yesterday and Today

Before

./app/views/email_templates/_denial_letter_feb.html.erb

```erb
<body style="margin: 0; padding: 0;">
  <table>
    <tr class="header" align="center">
      <td>
        <p>CreditNinja</p>
        <p>27 N. Wacker Drive #404</p>
        <p>Chicago, IL 60606</p>
      </td>
    </tr>
    <tr>
      <td>
        <p>Date: <%= @date_str || Time.now.strftime("%m/%d/%Y") %></p>
      </td>
    </tr>
  </table>
</body>
```

- Writting HTML, JS , CSS separatelly
- Building without isolation testing in mind
- Pretty hard to reuse the view logic site-wide

Today

./app/frontend/pages/EmailTemplates/routes/DenialLetterFeb.jsx

```jsx
import React from 'react';

class Body extends React.Component {
    render() {
        const { date_str, time } = this.props

        return (
            <body style="margin: 0; padding: 0;">
                <table>
                    <tr class="header" align="center">
                        <td>
                            <p>CreditNinja</p>
                            <p>27 N. Wacker Drive #404</p>
                            <p>Chicago, IL 60606</p>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <p>Date: {date_str || time}</p>
                        </td>
                    </tr>
                </table>
            </body>
        )
    }
}
```

- Each piece of code is a reusable module
- Ability to create centralized UI Kit library

Why React

Developers love it

Open-Source Project since 2011 - **165k+ stars** on **GitHub**

Huge, friendly community - **468k+ followers** at **Twitter**

React - a frontend open-source library created by Facebook. Aimed to deliver faster, more dynamic web pages on top of flexible ENV. Makes it painless to create interactive UIs.

1

Simplicity & Reusablity

Start develop new features in React without rewriting existing codebase

2

React + React Native

90% equality btw React and React Native coding approaches

3

Declarative abstraction

Declarative views make your code more predictable and easier to debug

4

Learn Once, Write Anywhere

React can also render on the server-side for better SEO

Controller

Redux – Redux is a predictable state container for JavaScript apps. It helps you write applications that behave consistently and are centralized in mind.

1

Predictable

Redux helps you write applications that behave consistently, run in different environments client/server.

2

Centralized

Centralizing your application's state and logic enables powerful capabilities like undo/redo, state persistence, and much more.

3

Debuggable

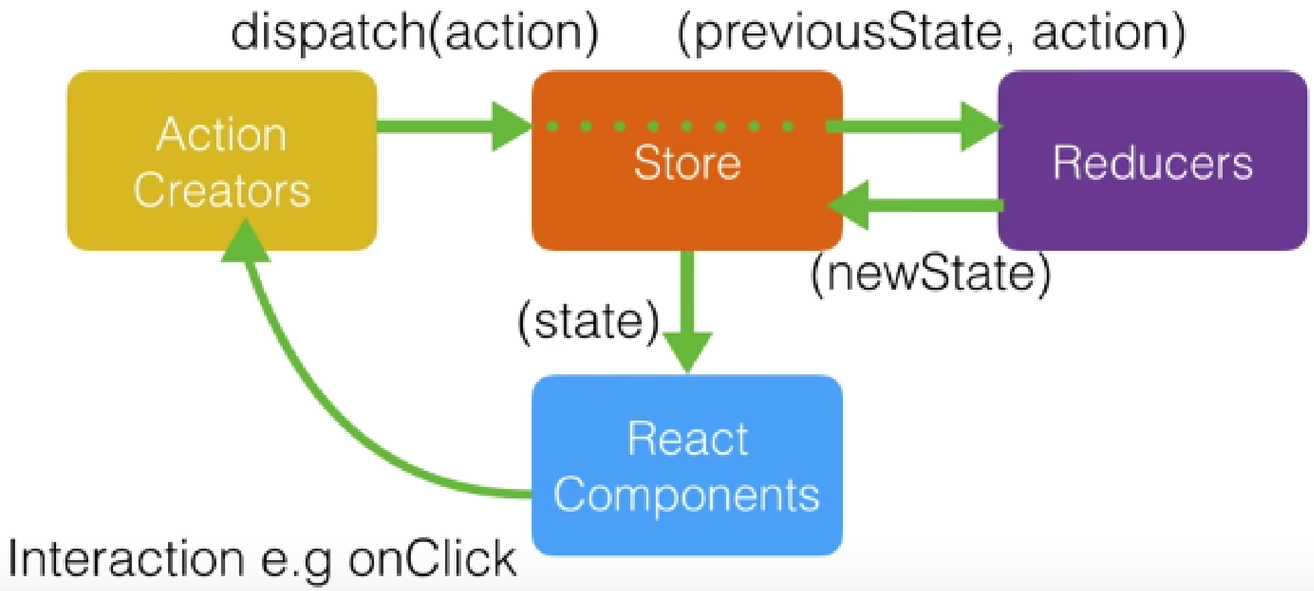
The Redux DevTools make it easy to trace when, where, why, and how your application's state changed.

4

Flexible

Redux works with any UI layer, and has a large ecosystem of addons to fit your needs.

Redux Flow


dispatch(action)
(previousState, action)
Action Creators
Store
Reducers
(newState)
(state)
React Components
Interaction e.g onClick

React-Router

React Router - is a standard library system built on top of React and used to create routing in the React application using React Router Package. It provides the synchronous URL on the browser with data that will be displayed on the web page. It maintains the standard structure and behavior of the application and is mainly used for developing single-page web applications.

1

SPA Navigation

Easy to track over SPA navigation and Redux controller state manager

2

History tracking

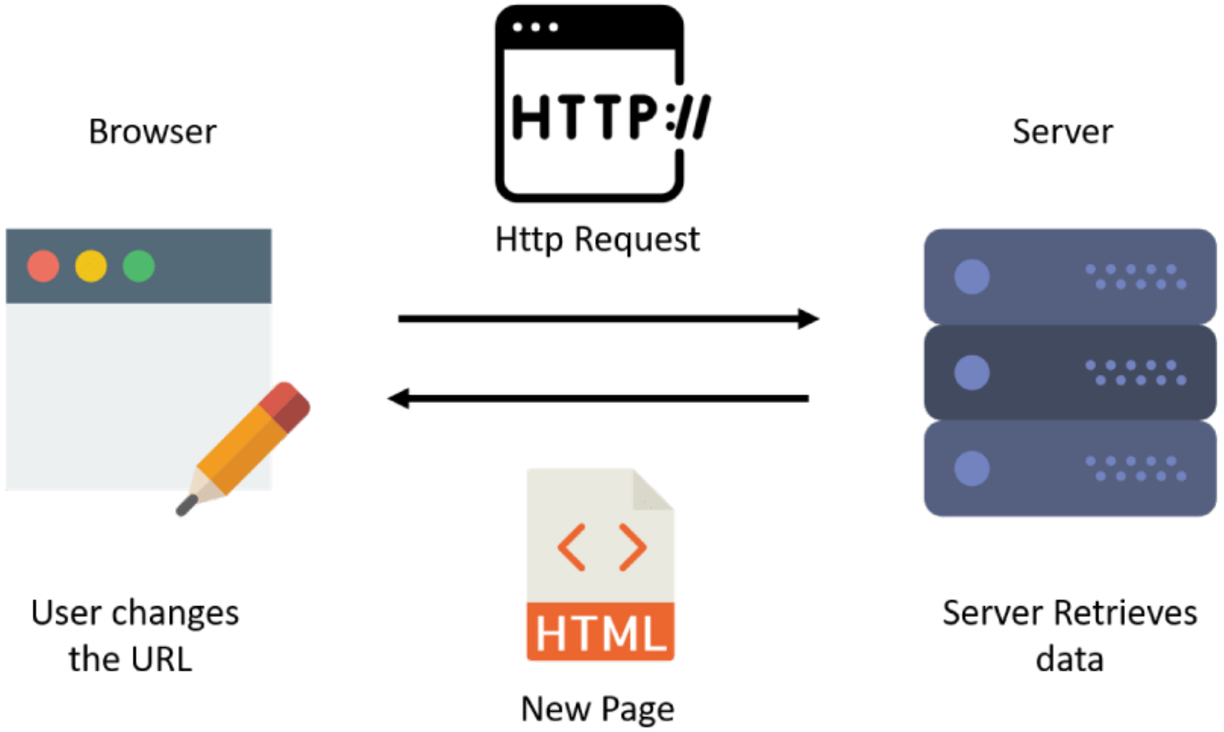
Navigation with centralized controller in Redux state

3

Flexible

Easy to inject/unplug any brand new route

React-Router flow



Redux-Saga - intuitive Redux-state side effect manager. Easy to manage, test, and executes efficiently. For any E2E request/async operation.

1

Asynchronous

ES6 generators make asynchronous flows easy to read, write, and test. Create complex side effects without getting bogged down by the details.

2

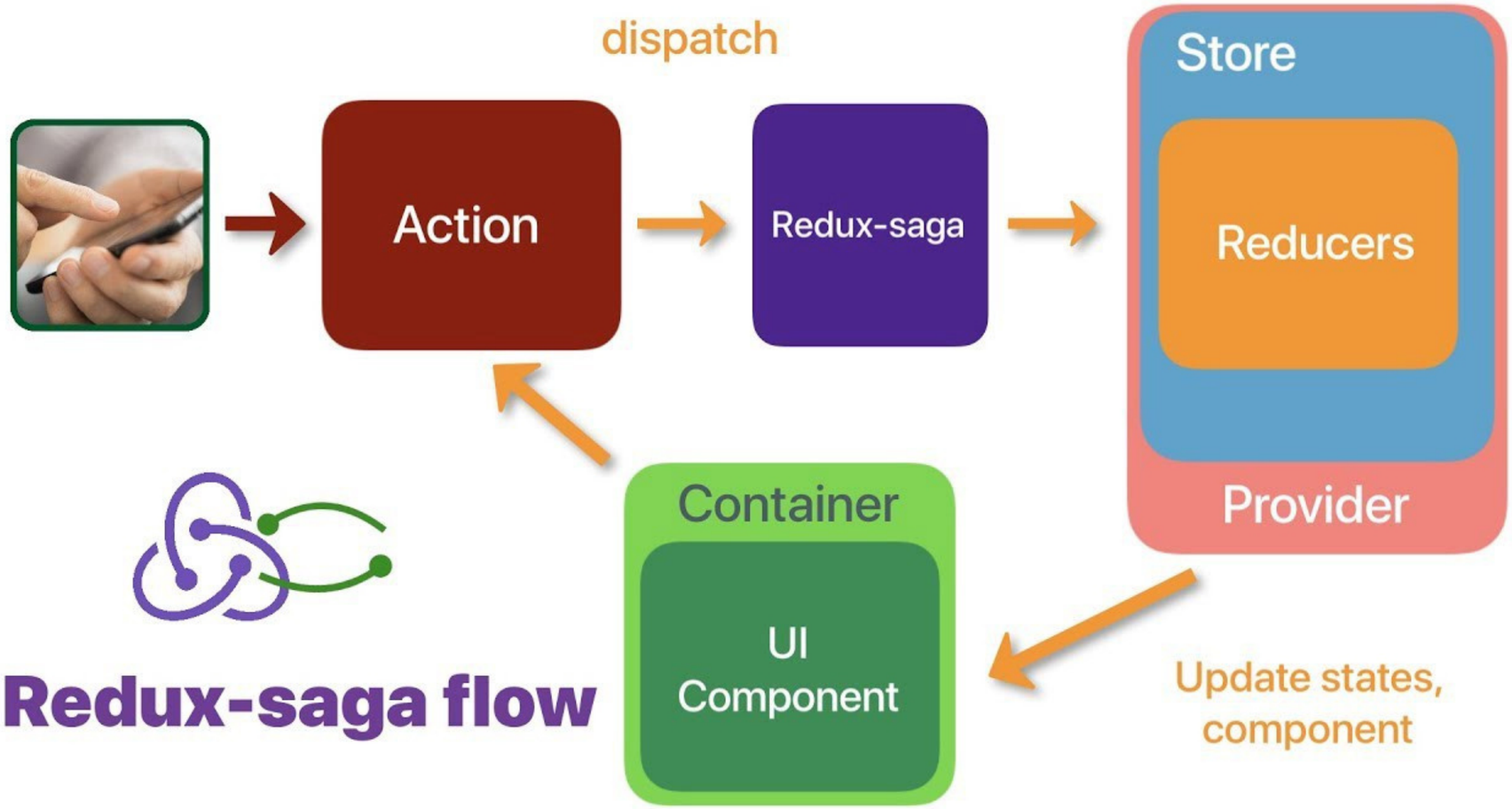
Composition-focused

Sagas enable numerous approaches to tackling parallel execution, task concurrency, task racing, task cancellation, and more. Keep total control over the flow of your code.

3

Easy To Test

Assert results at each step of a generator or for a saga as a whole. Either way, side effect testing is quick, concise, and painless, as testing should be.

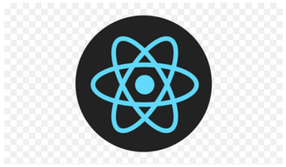

dispatch
Action
Redux-saga
Store
Reducers
Provider
Container
UI
Component
Update states,
component
Redux-saga flow

Architecture

How to create a flawless architecture

1. Balance loose coupling with high cohesion
2. Omit circular dependencies
3. Establishing environment with scale in mind
4. Choose reliable tech stack
5. Create a SOLID foundation
6. A heavy pinch of self-experience Developer and Community best practices

What do we need



ReactJS

UI-writing pattern library

Babel

Writing code on peak features

TypeScript

Typization for JS codebase

Storybook

Isolated UI Kit creation

NodeJS Server

Apps local run out of ruby ENV

Webpack

Optimizing code on the output

Linting

Code-checking (aka rubokop)

Jest Testing

Writing Unit test ENV

Why do we need

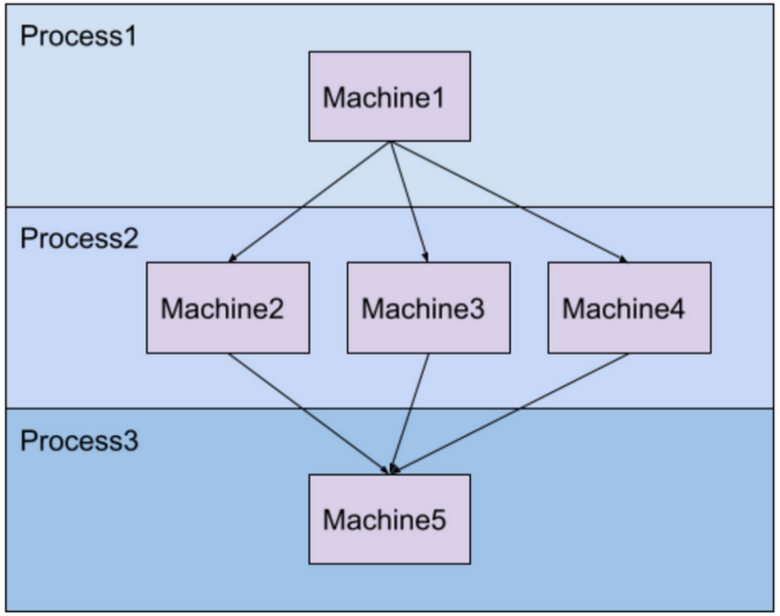
- Invent low coupling and high cohesion btw back-end and front-end.

- Establish a single source of truth for front-end codebase.

- Ability to write front-end on top of the latest language abilities.

DALs

Core Level



react-entrypoint

↑

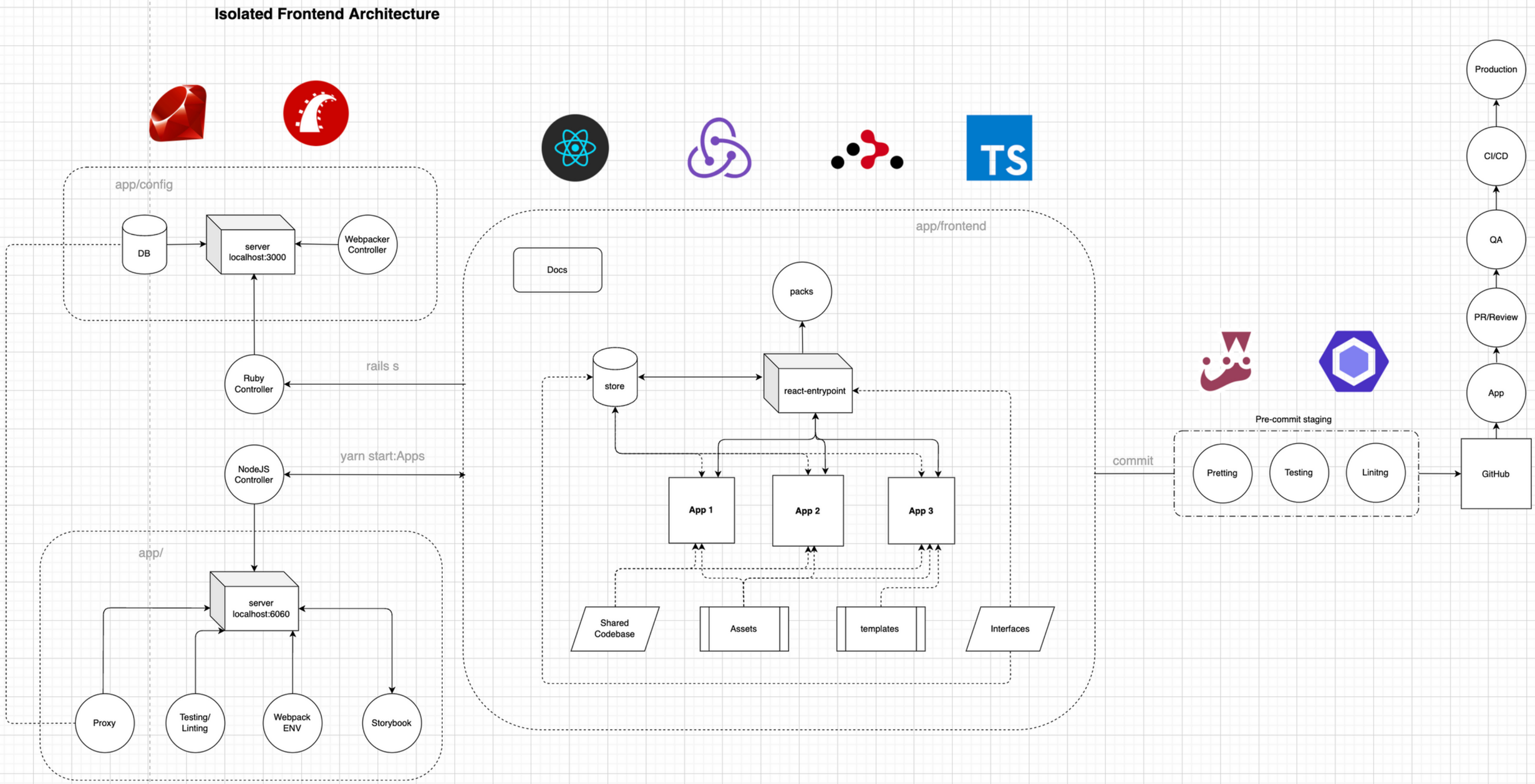
App 1 / App 2 / App 3

↑

Reusable codebase
(UI-Kit/DB/Utils)

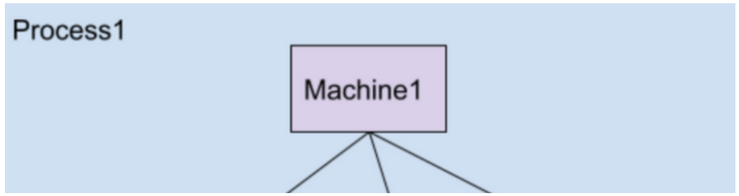
Why this abstraction works

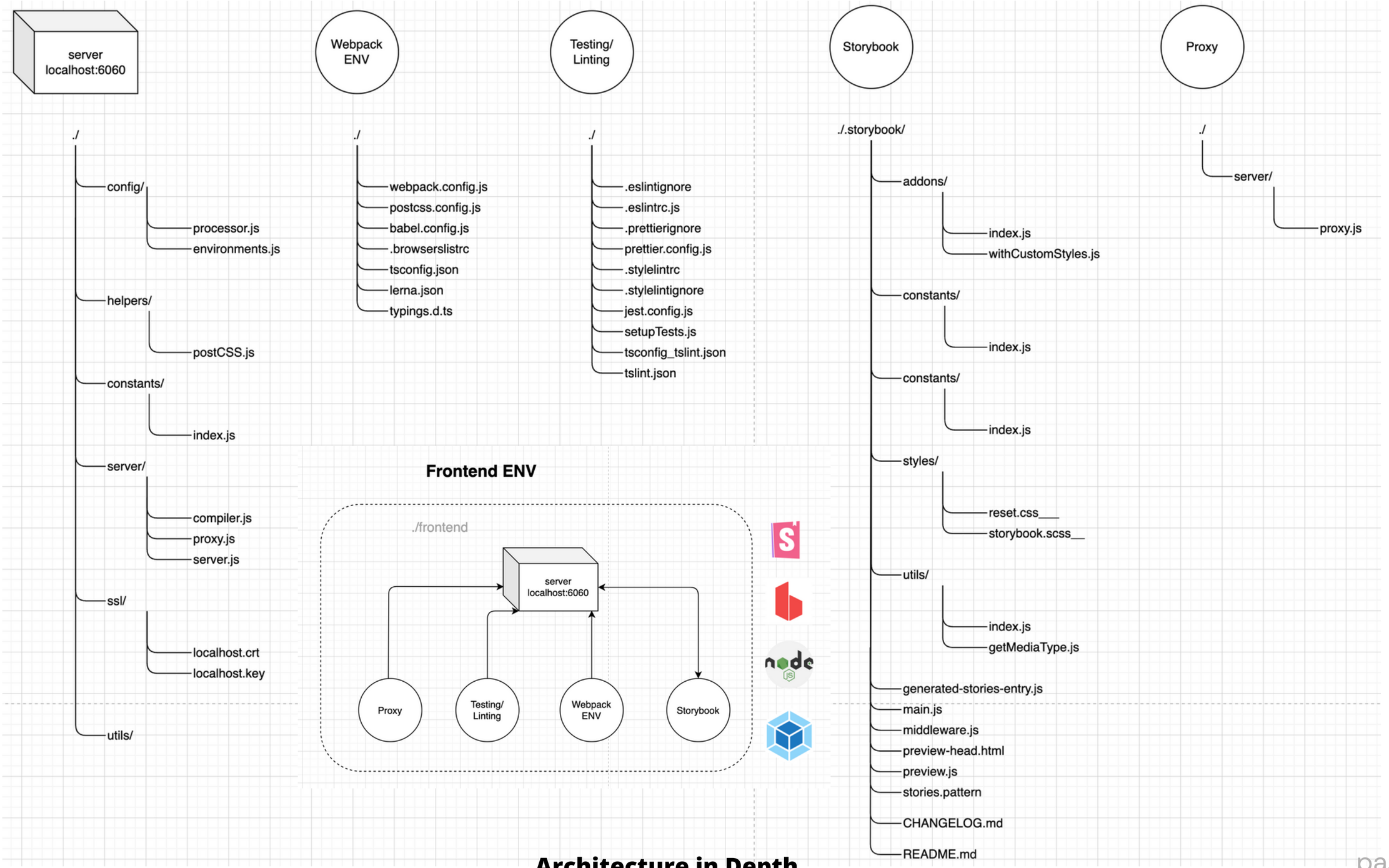
The modern frontend has been involved. The times when we could build the page on top of pure CSS/HTML/JS were gone. Nowadays it requires huge knowledge.
A bunch of reliable packages (aka gems). Of course, the ability to write scalable Frontend architecture stands on principles of Reusability, Single Source of Truth, and Scalability in mind.



Main Back-end/Front-end Architecture

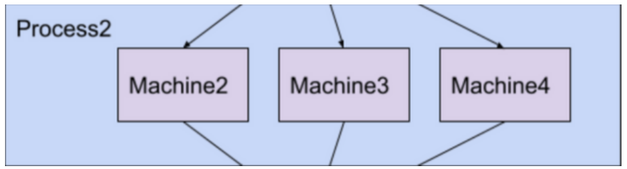
Main Level





Architecture in Depth

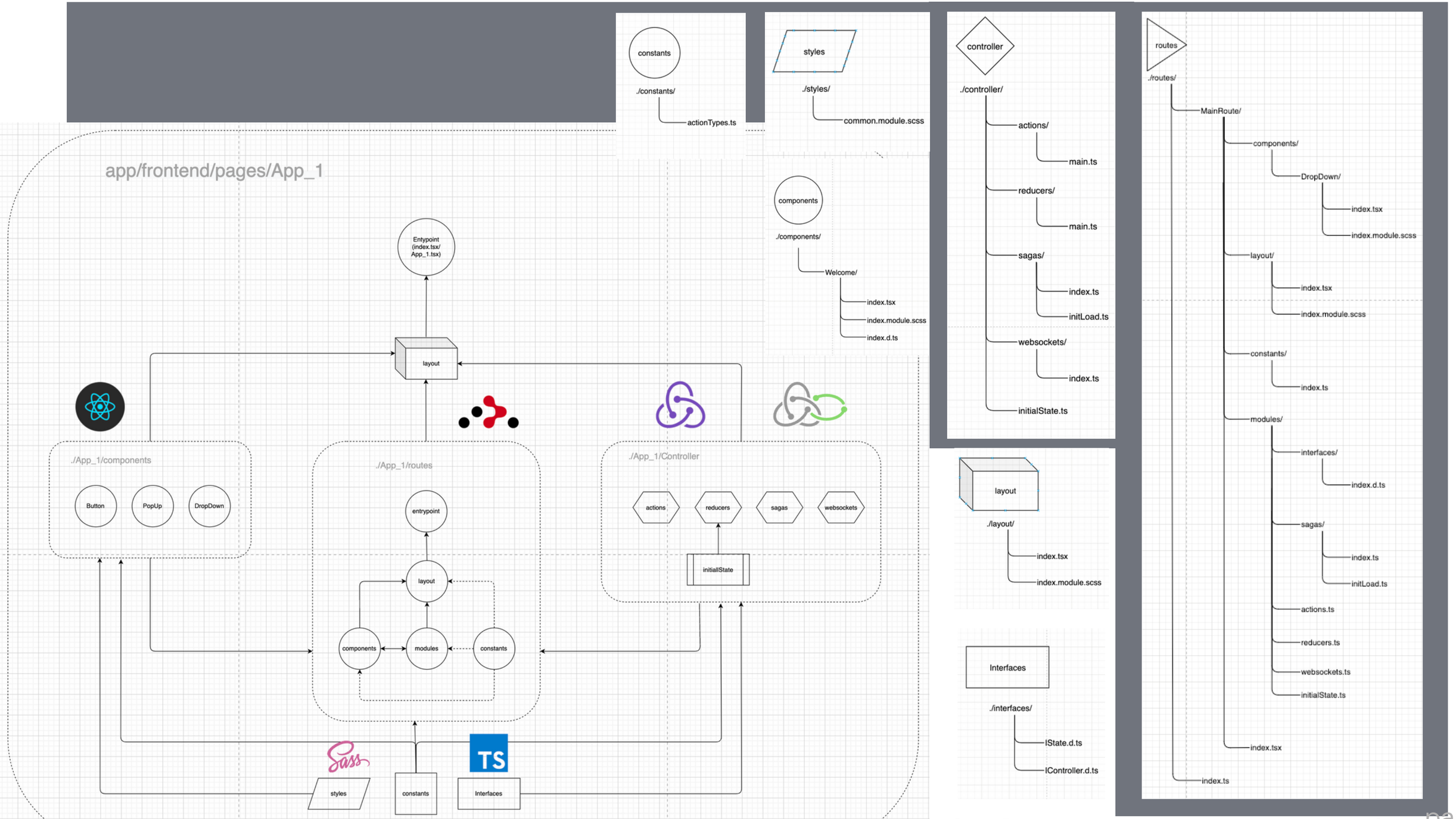
App Level



Easy App Roll Out

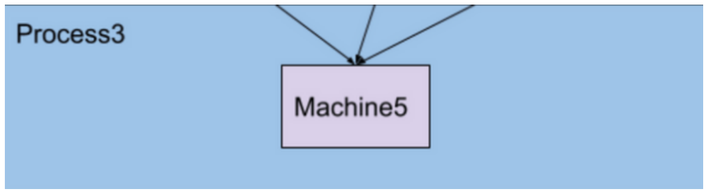
1. **Start Frontend:**
   - Run **start:frontend** to run the SPA app apart from backend

2. **Start backend:**
   - Run **start:backend** to run the backend through proxy to get/post real-world data with the frotnend



Particular App Front-end Architecture

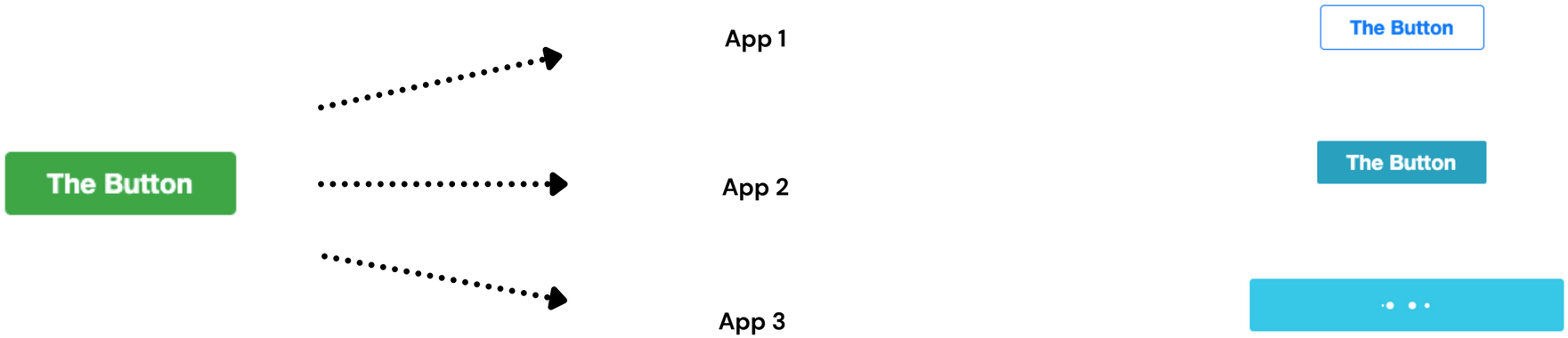
Child Level



Mar 17, 2022

Reusable Codebase

Centralized UI Kit – a single source of truth for Designer/Dev/QA Teams.



1

Write Once

Create a single Element based on the designers mocks

2

Use Everywhere

Integrate the same Element across the projects App_1/App_2/App_3

3

Extend it

Add new shapes, colors, sizes while referring to the same Element.

Pros and Cons

Pros

1. Independent, out from back-end scope development.
2. Modern frontend environment for day-by-day coding.
3. Reusable front-end codebase across all projects.
4. Faster immersion for beginners based on pure front-end architecture.
5. An ability to level up front-end coding at all.

Cons

1. Use E2E communication on top of REST only.
2. Start thinking about Frontend as an Independent Entity with its own ENV/Rules/Scope.
3. SEO Support became harder, but still possible

CI/CD + Automation

Code Quality

Write the code for human, not machine (c)

Prettier

Live-typing auto correction

ESLint

JavaScript/TypeScript code checking

Stylelint

CSS/SCSS stylesheets checking

TypeScript

JavaScript type checking

===

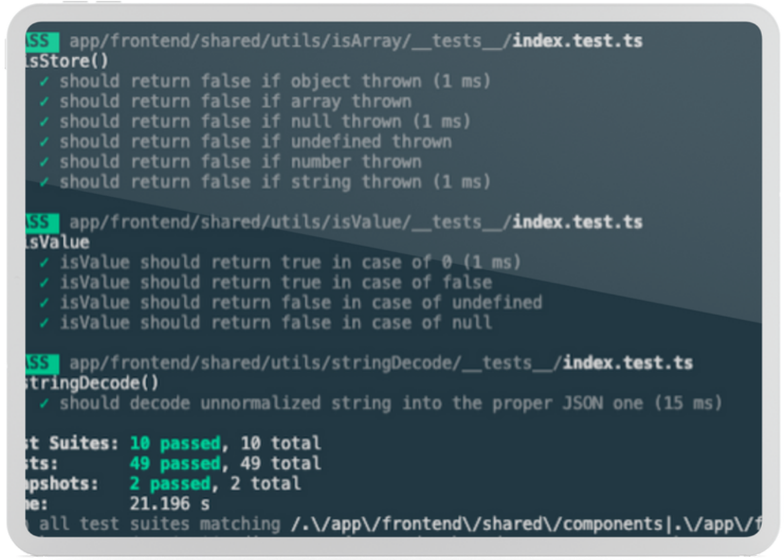
Automated codebase checking . Makes code writing consistent . Simplifies developers life ❤

Testing and QA

Automated and Semi-automated Code Testing

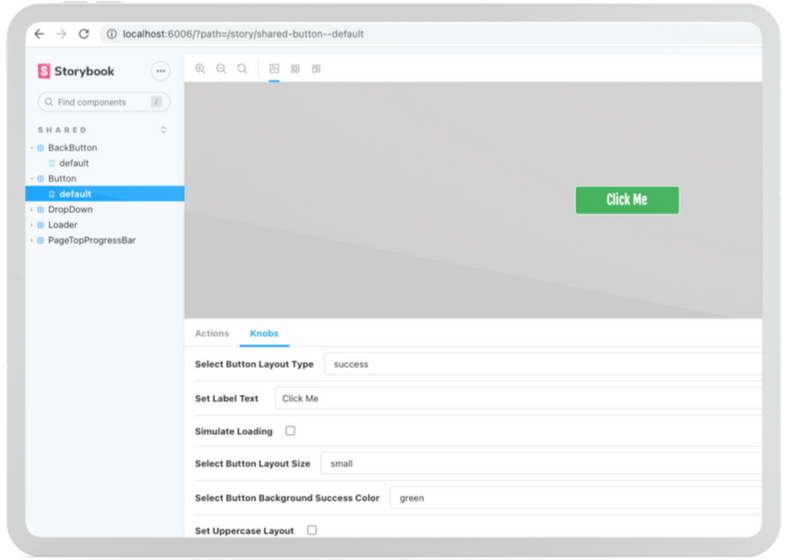
Jest

Declarative library for automated code testing.
Provides user-friendly API for Unit tests running.



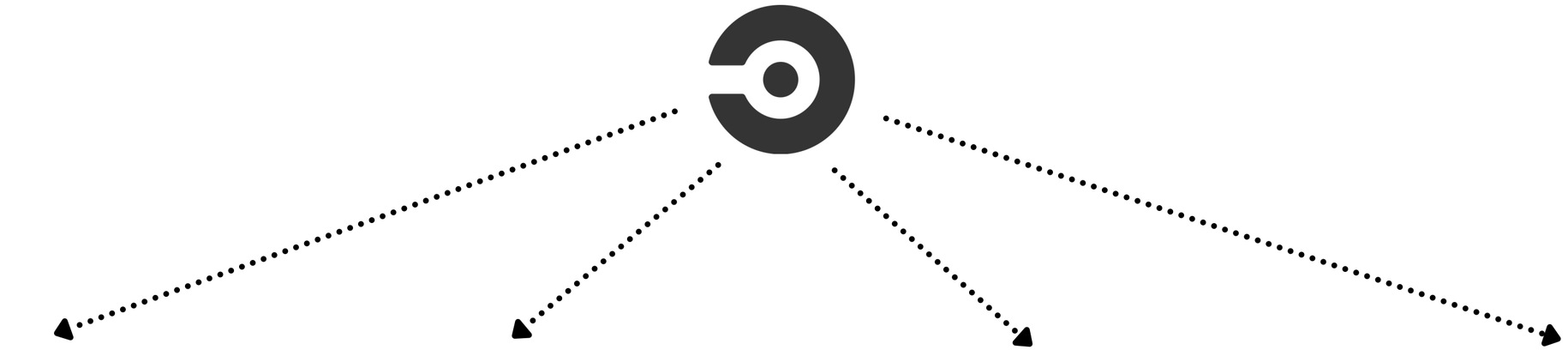
Stories

User-friendly UI for manual testing Components.
QA can check each entity through different ways.



CI/CD pipelines

We use CircleCI as a service for last stand checking and delivering code into production



1

Test it

Run Jest Unit test and Storybook build

2

Lint it

Check code quality, autocorrect and commit back into PR (if possible)

3

Build it

Try to check if the build ready for production without errors

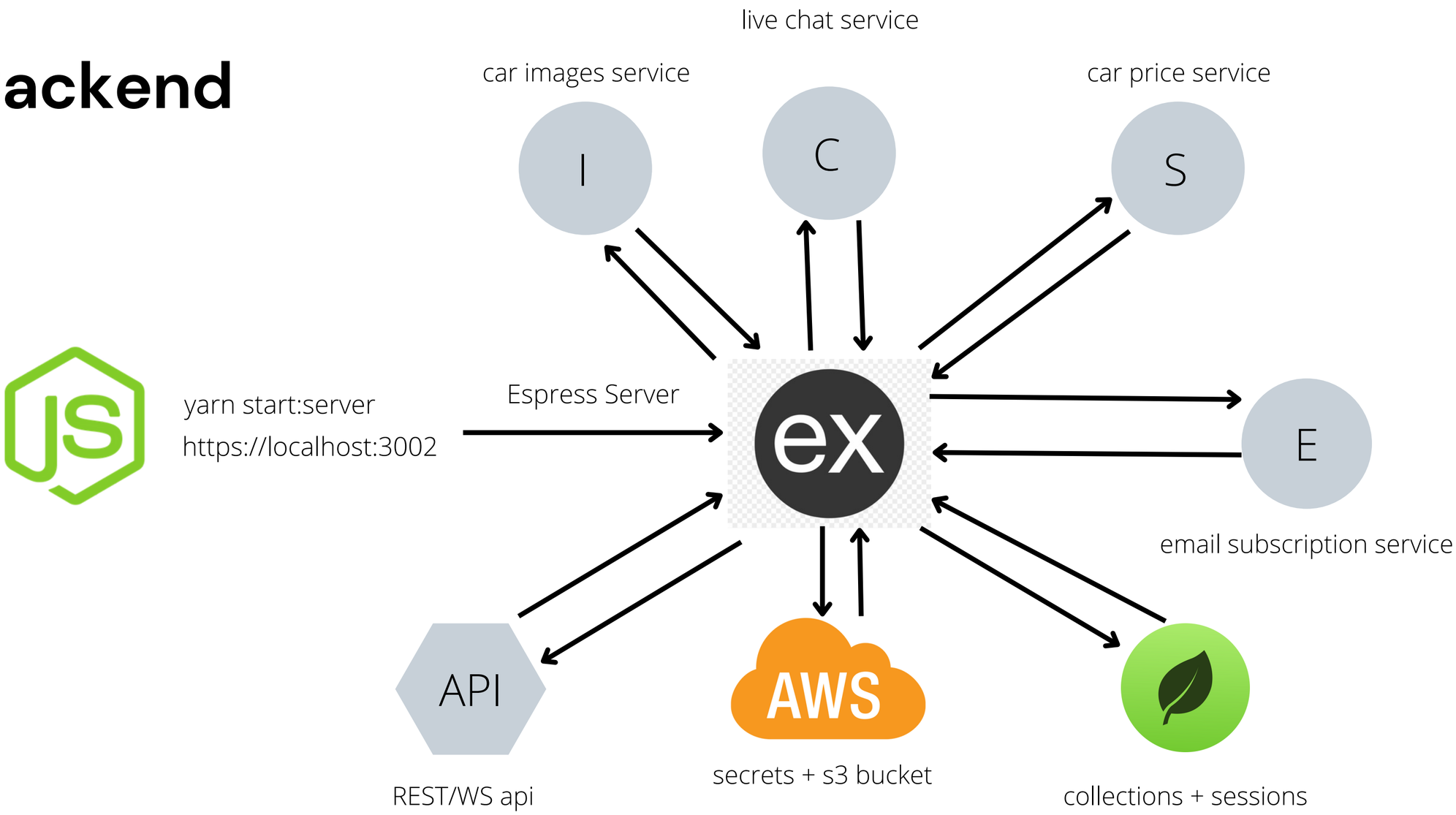
4

Deploy it

Automative deploy on NPM server for further usage

Backend

Backend


live chat service
car images service
car price service
I
C
S
yarn start:server
https://localhost:3002
Espress Server
ex
E
email subscription service
API
AWS
REST/WS api
secrets + s3 bucket
collections + sessions

Legends

Motivation

It's always been my passion to create scalable apps in mind. I believe, for sure, the present abstraction between the back-end and front-end is pretty nice at your end.

You already can manage HTML, CSS, JavaScript files all along under the present ENV. Meanwhile, all the abstractions you have currently in use are aimed at the back-end code writing in most. But do we think about convenience around the frontend codebase?

For example - when you need to fix some tiny bug located inside some deep page.
It cost you several hours just to navigate through each step of Application Flow to get there.

Here is a small portion of advantages that the Isolated frontend can bring us:

1. Deliver Front-end things in production faster.
2. Modern frontend ENV for day-by-day coding.
3. Reusable front-end codebase across all projects.

Diagrams

You can use the https://app.diagrams.net/ service to open the diagrams attached to the PDF file along.

The diagrams attached are:
- **isolated_app.drawio** describes the particular app architecture.
- **isolated_apps.drawio** describes the app's architecture on a middle layer.
- **isolated_env.drawio** describes the core level abstraction of isolated frontend development.
- **Isolated_Frontend.drawio** describes the whole picture of things required to be used to create a scalable Fronted architecture.

MIsconceptual Backend

I imagine we don't have NodeJS as a backend yet, so I found it wise to show how to combine heavy-weight Backend ENV with modern Frontend ENV.

NodeJS was used as an example of a backend. To demonstrate how things will work once the frontend will be established as a separate ENV. It's essential to understand how the assets & static will be served by a server. Don't pay much attention to why Ruby stands here, but how it can work together.

Packages & Services

Here I list the package we used on slides before. So you can easily surf them to understand what they stand for.

- Redux-Saga - https://redux-saga.js.org/
- ESLint - https://eslint.org/
- Jest - https://jestjs.io/
- Webpack - https://webpack.js.org/
- Babel - https://babeljs.io/
- TypeScript - https://www.typescriptlang.org/
- Sass - https://sass-lang.com/
- MongoDB - https://mongodb.com/
- AWS - https://aws.amazone.com/
- React - https://reactjs.org/
- React Router - https://reactrouter.com/
- Redux - https://redux.js.org/
- CircleCI - https://circleci.com/
- Stylelint - https://stylelint.com/
- Prettier - https://prettier.com/
- ExpressJS - https://expressjs.com/
- NodeJS - https://nodejs.org/
- D3JS - https://d3js.org/
- Storybook - https://storybook.js.org/

Mar 17, 2022

Thank you for participating! Have a great day ahead :)