

I18N - a short story

MOHSEN SHAFIEI

Frontend Engineer @ Shopee

Github: mohsenshafiei

WHAT IS IN IT?

EXPERIENCE

PROBLEM

SOLUTION

MAKE OUR SOLUTION BETTER

WHAT IS NEXT?


Once Upon a Time i18n

Manager:

“Add a new language today.”



Are You Kidding Me?



```
export default class Translator {
  constructor(locale, dictionary) {
    this.locale = locale ? locale : 'en';
    this.dictionary = dictionary;
  }
  t = (key) => {
    if (dictionary && dictionary[this.locale]) {
      return dictionary[this.locale][key]
    }
    return (key)
  }
  setLocale = (locale) => this.locale = locale
  getLocale = () => this.locale
}
```



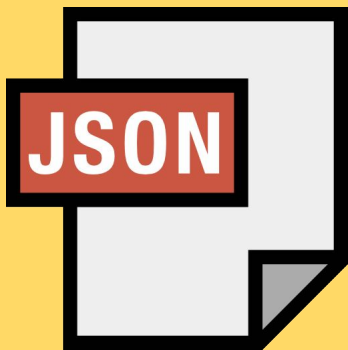

```
const i18nFactory = (locale, dictionary) => {  
  const client = new Translator(locale, dictionary)  
  const t = client.t  
  const setLocale = client.setLocale  
  const getLocale = client.getLocale  
  return {  
    t,  
    getLocale,  
    setLocale  
  }  
}
```

```
const dictionary = {
  en: {
    "label_ok": 'ok'
  },
  fa: {
    "label_ok": 'خوبه'
  }
}

const locale = window.localStorage &&
  window.localStorage.getItem &&
  window.localStorage.getItem('locale') &&

export default i18n = i18nFactory(locale, dictionary);
const Component = () => <h1>{i18n.t('label_ok')}</h1>
```

There are different approaches.



JSON



Webpack



Gatsby

Pains & Gains

Gains:

- They are fast to implement
- They are good enough when your team is small
- They are suitable for small and mid-size projects

Pains:

- **Managing keys & translations is hard**
- **Merge Conflict**
- **Slow Delivery**

**What do we need to make
everything better?**

Let's review the process

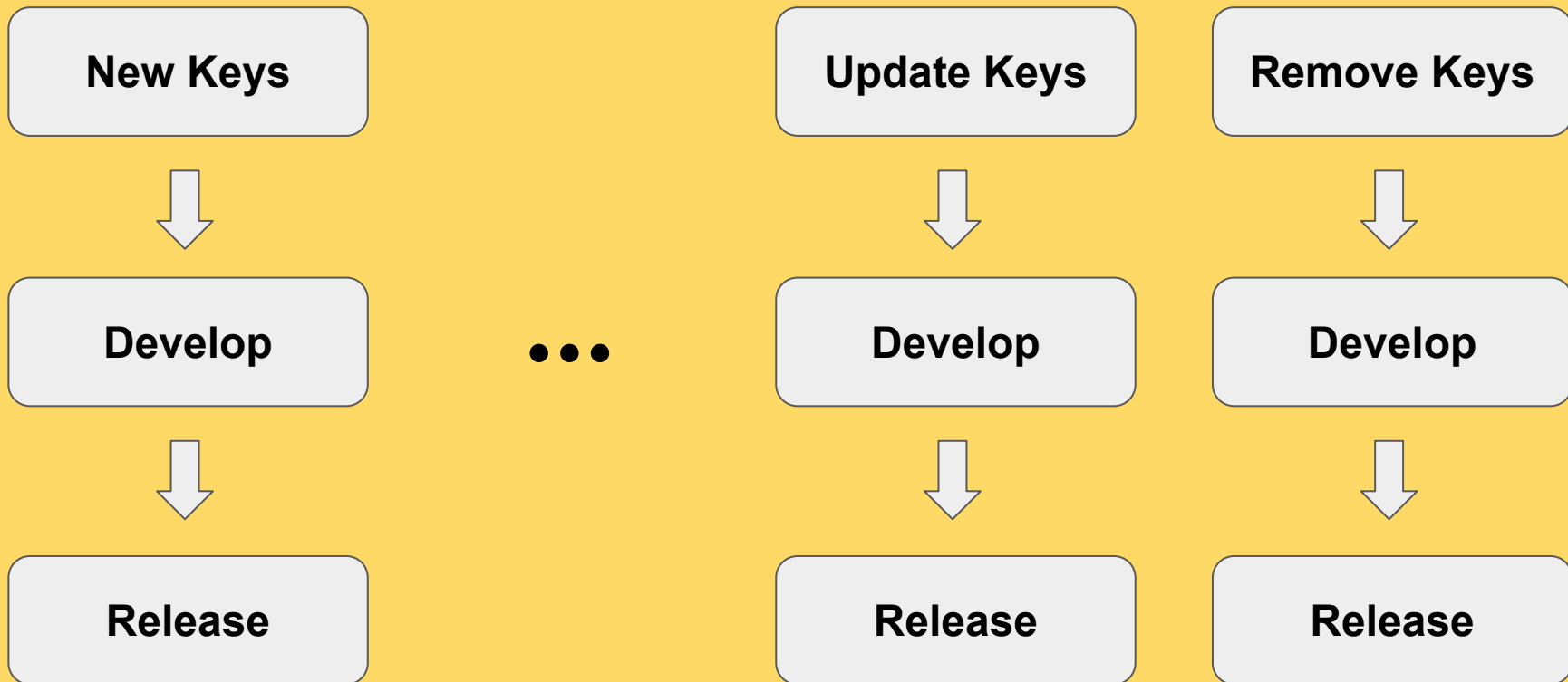
New Keys

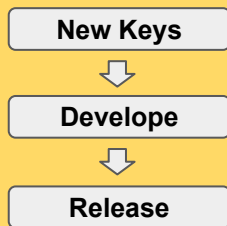


Develop

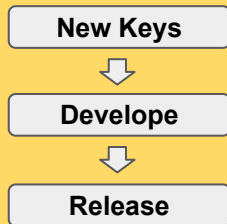
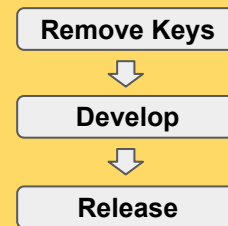
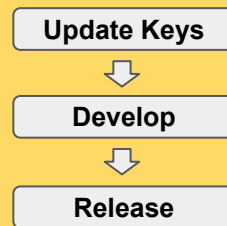


Release

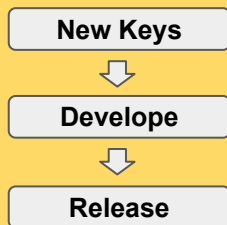
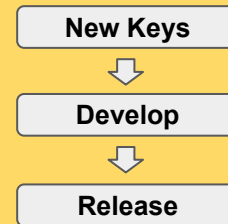
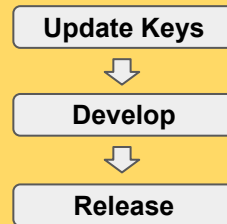




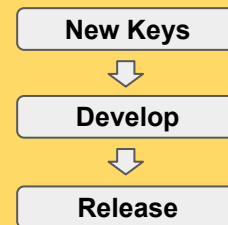
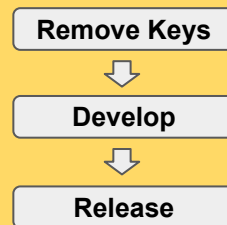
...



...



...



**EXHAUSTED.
JUST EXHAUSTED.**



TOO EXHAUSTED.

What is the solution?

BOOOOSTER

Booster Goals:

- Fast Translation Delivery
- Fast Translation Development (Add, Remove, Update)
- Better Collaboration On Translation
- Less Merge Conflict
- Easy To Release

But How?

You need a dictionary



Which cases we solved?

Goals:

- Fast Translation Delivery ✓
- Fast Translation Development (Add, Remove, Update) ✓
- Better Collaboration On Translation ✓
- Less Merge Conflict ✓
- Easy To Release ✓

Do you think that's all?

NO

Let's review it again

First: Repeat this process again and again

New Keys



Development



Release

What Is happening?

We have to deploy the entire project to a live environment many times because of translation updates.

Second: Developers add more and more keys



Add Key

Add Key

...

Add Key



Add Key

Add Key

...

Add Key



Add Key

Add Key

...

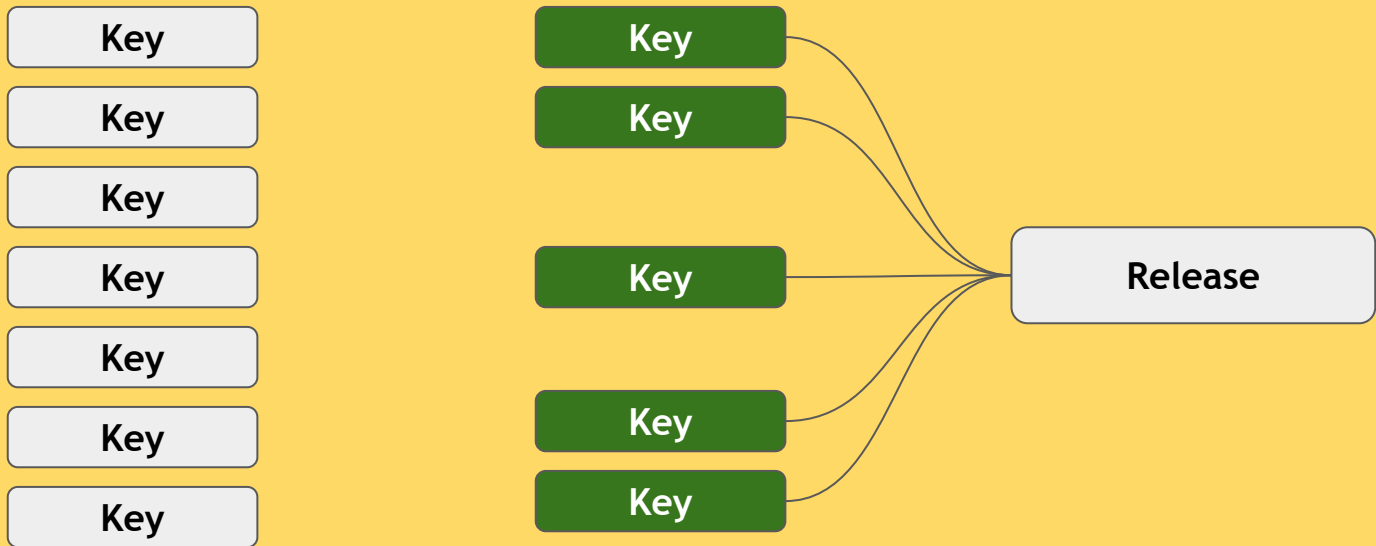
Add Key

IMPORTANT QUESTION

Do we need all those keys?

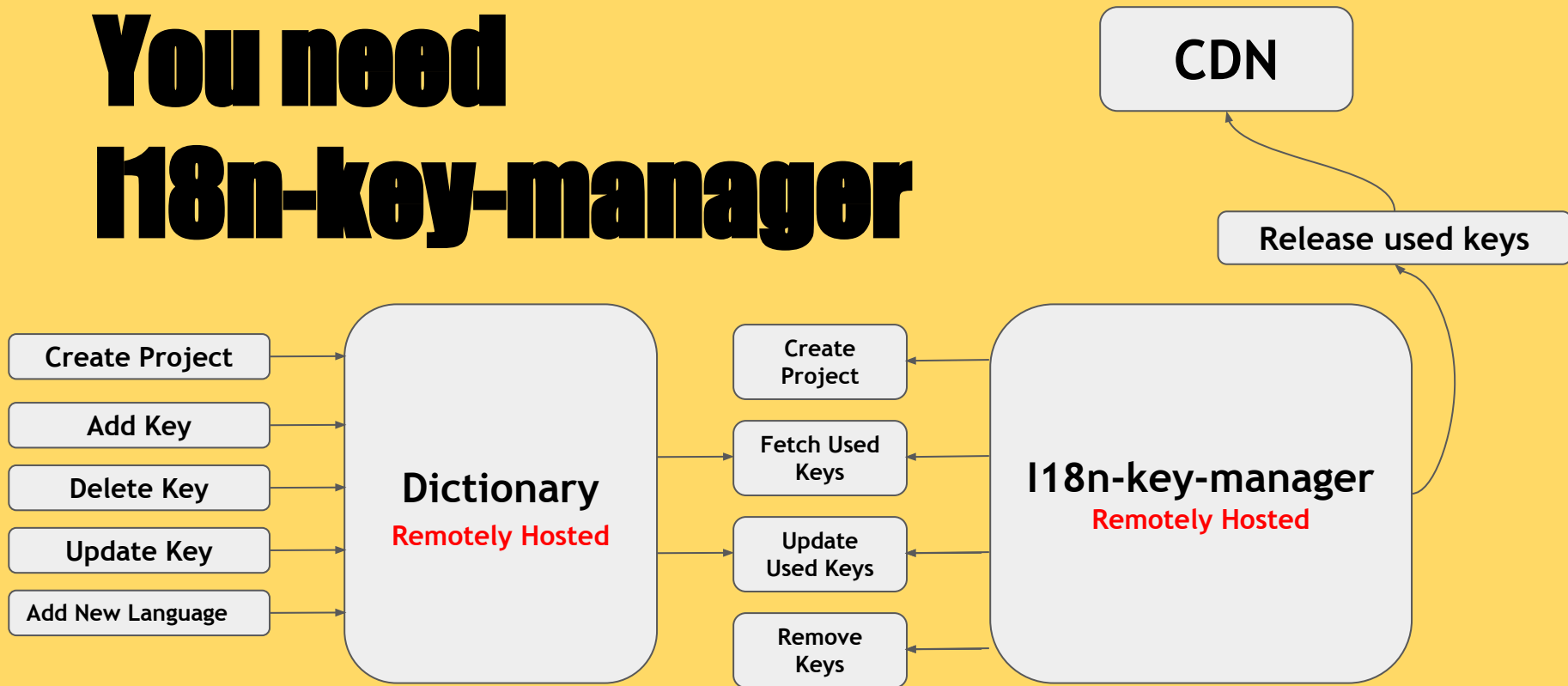
What Is the solution?

We need something to control releasing of used keys



But How?

You need I18n-key-manager



My Frontend Project

Add New Key

Release

key_label_1

translation

Update

Remove

key_label_2

translation

Update

Remove

key_label_3

translation

Update

Remove

key_label_4

translation

Update

Remove

key_label_5

translation

Update

Remove

key_label_6

translation

Update

Remove

key_label_7

translation

Update

Remove

key_label_8

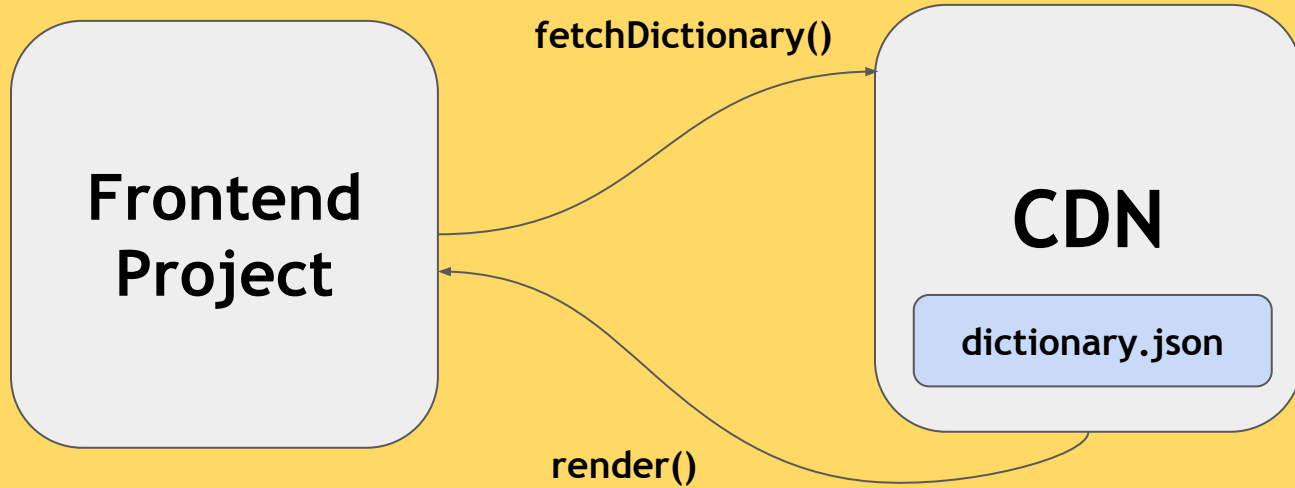
translation

Update

Remove

How we can use it in a real project?

It is simple



```
const fetchDictionary = async (path) => {
  const result = await fetch(path);
  return result;
}

const initDictionary = async () => {
  const locale = 'en'
  const dictionaryPath = 'https://my-dictionary-path';
  const dictionary = await fetchDictionary(dictionaryPath)
  const i18n = i18nFactory(locale, dictionary);
  return i18n;
}

const App = async () => {
  const i18n = await initMyDictionary();
  return (<h1>{i18n.t('label_ok')}</h1>)
}
```

Can we make it better?

There are three approaches:

- **Fetch on render**
- **Fetch then render**
- **Render as you fetch**

Fetch on render

```
export default class Translator {
  constructor(locale) {
    this.locale = locale ? locale : 'en';
    this.dictionary = {};
  }
  t = (key) => {
    if (isEmpty(this.dictionary)) return key;
    if (this.dictionary.hasOwnProperty(key)) {
      return this.dictionary[key]
    }
    return key;
  }
  fetchDictionary = async (path) => {
    const result = await fetch(path);
    if (result.status === 200) {
      Object.assign(this.dictionary, result.dictionary)
    }
    return {}
  }
  hasDictionary = () => !isEmpty(this.dictionary)
}
```



```
const i18nFactory = (locale) => {  
  const client = new Translator(locale)  
  const t = client.t  
  const fetchDictionary = client.fetchDictionary  
  return {  
    t,  
    fetchDictionary,  
    hasDictionary,  
  }  
}  
  
const i18n = i18nFactory('en')
```




```
const App = async () => {  
  return (<Content />)  
}  
  
const Content = async () => {  
  const [isReady, setIsReady] = useState(false);  
  const path = `https://dictionary.json`  
  useEffect(() => {  
    i18n.fetchDictionary(path).then(() => setIsReady(true))  
  }, [])  
  if (isReady && i18n.hasDictionary()) {  
    return (<h1>{i18n.t('label_ok')}</h1>)  
  }  
  return <h1>Loading Page1 ... </h1>  
}
```

Fetch then render



```
const i18n = i18nFactory('en')

const init = async () => {
  const path = `https://dictionary.json`
  await i18n.fetchDictionary(path);
  render();
}

const render = () => {
  ReactDOM.render(<App />)
}

const App = async () => {
  return (<h1>{i18n.t('label_ok')}</h1>)
}

init();
```

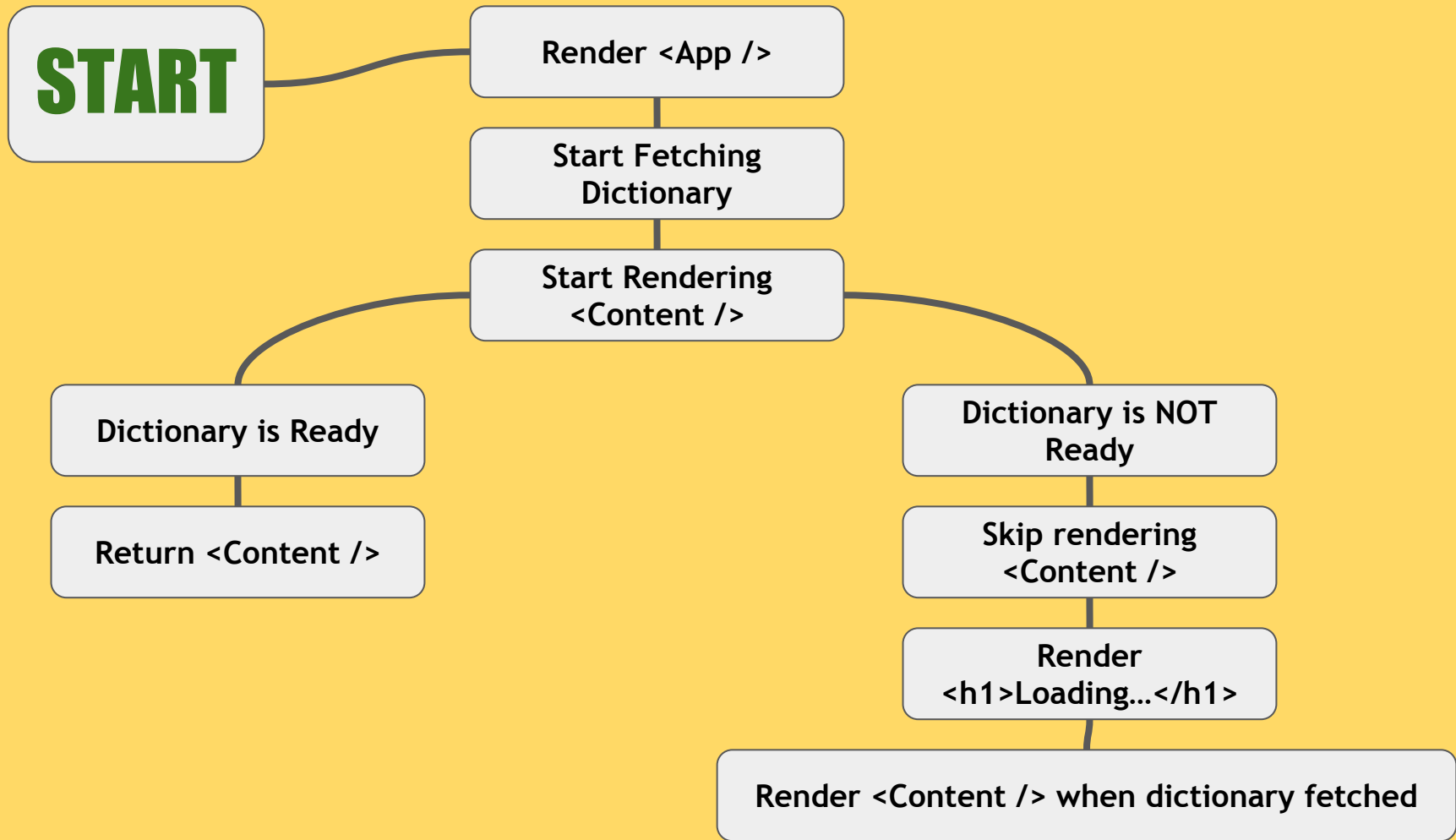
Render as you fetch

```
export default class Translator {
  constructor(locale) {
    this.locale = locale ? locale : 'en';
    this.dictionary = {};
  }
  t = (key) => {
    if (isEmpty(this.dictionary)) return key;
    if (this.dictionary.hasOwnProperty(key)) {
      return this.dictionary[key]
    }
    return key;
  }
  fetchDictionary = async (path) => {
    const result = await fetch(path);
    if (result.status === 200) {
      Object.assign(this.dictionary, result.dictionary)
    }
    return {}
  }
  useDictionary = async (path) => {
    if (this.dictionary) return
    throw new Promise(fetchDictionary(path))
  }
}
```

```
const i18n = i18nFactory('en')

const App = () => {
  const path = `https://dictionary.json`
  useDictionary(path);
  return (
    <Suspense fallback={<h1>Loading Page1 ... </h1>}>
      <Content />
    </Suspense>
  )
}

const Content = () => {
  return (<h1>{i18n.t('label_ok')}</h1>)
}
```



Do you think it is good enough?

**What will happen if the size of
used keys became huge?**

JSON = 100 KB

SLOW

What do we do when a file is big?

Split The File

D

IC

TIO

NA

RY

Let's split the `dictionary.json` file into small files

**You can decide which way is the right way
for you to split the json file**

First:

**Our i18n-key-manager should support
collections**



GOOD

**You only need to fetch a
collection when you use it**

```
export default class Translator {
  ...
  fetchCollection = async (collectionId) => {
    if (this.collections.has(collectionId)) return;
    if (!this.pendingCollections.has(collectionId))
      this.pendingCollections.set(
        collectionId,
        _downloadTranslationCollection(collectionId)
      );
    return pendingCollections.get(collectionId);
  }

  _downloadCollection = async (collectionId) => {
    const path = createPath(collectionId)
    const result = await fetch(path);
    if (result.status === 200) {
      Object.assign(dictionary, result.dictionary);
      this.collections.add(collectionId);
    }
    this.pendingCollections.delete(collectionId);
  }

  useCollections = (collectionId) => {
    if (this.collections.has(collectionId)) return
    throw new Promise(fetchCollection(collectionId))
  };
}
```

```
const i18n = i18nFactory('en')
const App = async () => {
  const locale = 'en'
  return (
    <
      <Suspense fallback={<h1>Loading Page1 ... </h1>}>
        <Page1 />
      </Suspense>
      <Suspense fallback={<h1>Loading Page2 ... </h1>}>
        <Page2 />
      </Suspense>
      <Suspense fallback={<h1>Loading Page3 ... </h1>}>
        <Page3 />
      </Suspense>
    </>
  )
}
```



```
const Page1 = () => {  
  useCollections(1);  
  return (<h1>{i18n.t('label_1')}</h1>)  
}
```

```
const Page2 = () => {  
  useCollections(2);  
  return (<h1>{i18n.t('label_2')}</h1>)  
}
```

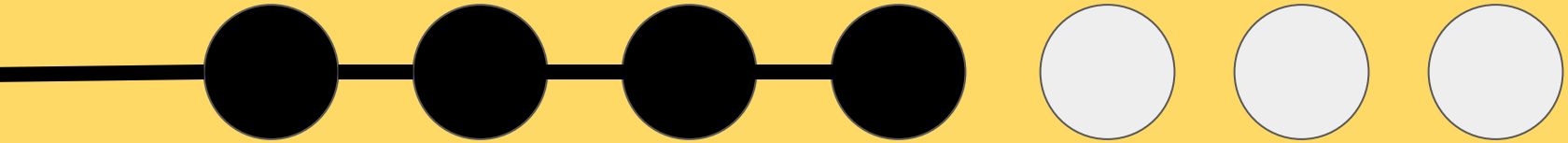
```
const Page3 = () => {  
  useCollections(3);  
  return (<h1>{i18n.t('label_3')}</h1>)  
}
```

**Why we don't automate
creating collections?**

Collections should be

CUSTOMIZABLE

What Is next?



Developer Experience Utilities

- Help developers to understand how to use keys more efficiently
- To create some IDE extensions for auto-complete keys
- etc

Compress Dictionary Keys

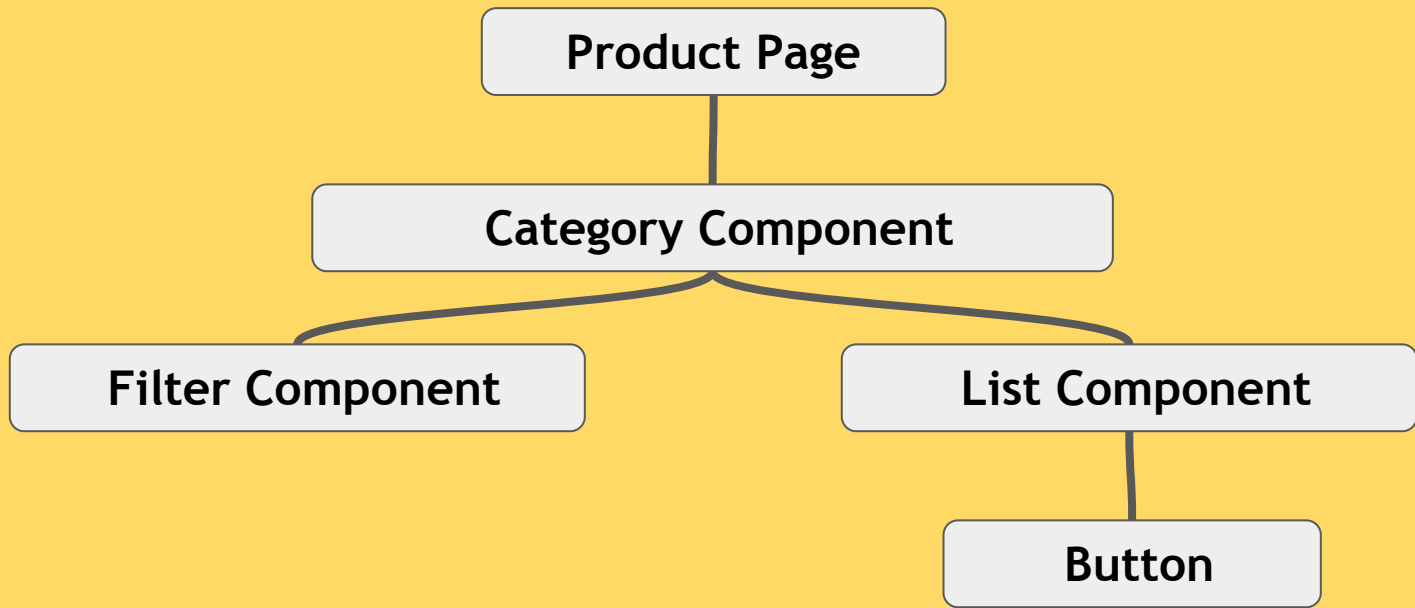
“loooooooooooooong-hello-key-label”: “hello”



“k7c”: “hello”

Solving Waterfall Problem

We are still figuring out



Each node fetch its own translation file

Product Page

40 ms

Category Component

60 ms

List Component

80 ms

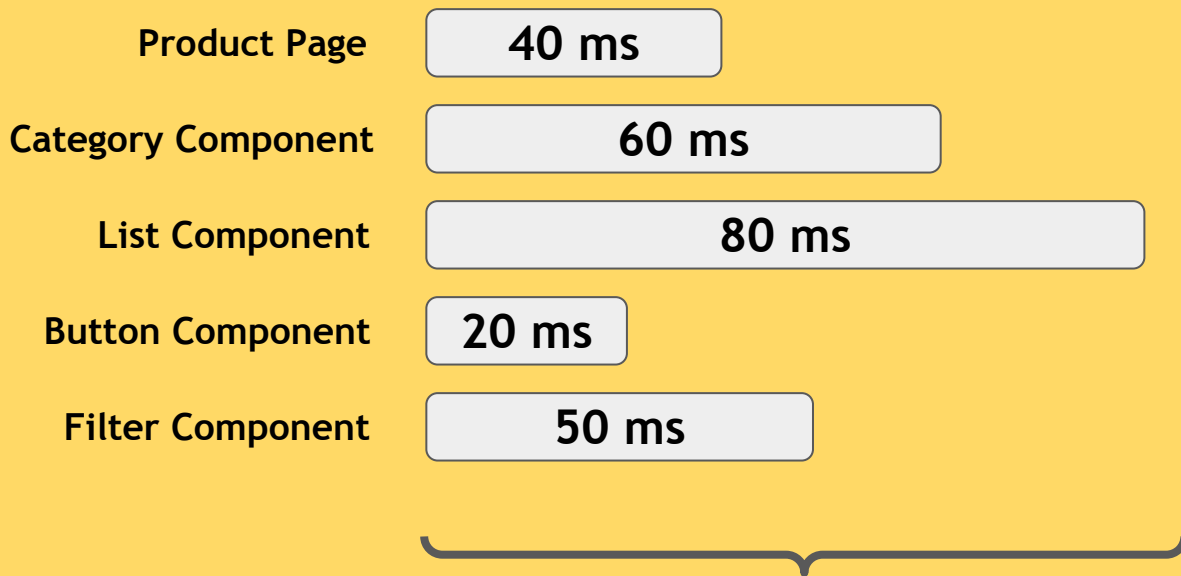
Button Component

20 ms

Filter Component

50 ms

We need **200 ms** to fetch all translation files



We need **80 ms** to fetch all translation files

Let's review everything

What we have done together?

- What are the different approaches of i18n?
- How to collaborate on translations to have fast delivery?
- How to manage used keys in our application?
- How to decrease the size of our translation file?
- How to make our user experience better?
- What we can do next?
- What are our challenges?

Questions?

Thank you