WEBAPP勉強会 第1回目 解説

- WEBAPP勉強会 第1回目 解説
  - プロジェクト作成
    - 起動する
  - NUXTの解説
    - NUXTのフォルダ構成について
    - 画面構成について
    - Vueの構成について
    - リアクティブとは？
  - 次回以降の準備
    - 実習用ページの追加
    - メニューの追加
    - 参考までに
  - webapp ソースの取得
    - historyから圧縮ソースを取得
    - git hub からソースを取得
    - モジュールダウンロード

プロジェクト作成

作業ディレクトリで下記のように入力します。

今回はテザリング使用しないとモジュールがダウンロードできないので、一旦ベースを作成したものから始めます。

右のリンクからダウンロードしてください。　<ベースのダウンロード>

以下は、コマンドラインからプロジェクトを作成する方法です。

```
CMD> npx create-nuxt-app webapp
```

いくつかの質問に答えていきます。

- programming language: **JavaScript**
- package manager: **Yarn**
- I framework: **Element**
- Nuxt.js modules: **Axios**
- linting tools: **ESLint, Prettier**
- rendering mode: **Single Page App**

```
create-nuxt-app v2.15.0
✨  Generating Nuxt.js project in webapp
? Project name webapp
```

```
? Project description My first NUXT
? Author name Hisakazu KATO
? Choose programming language JavaScript
? Choose the package manager Yarn
? Choose UI framework Element
? Choose custom server framework None (Recommended)
? Choose Nuxt.js modules Axios
? Choose linting tools ESLint, Prettier
? Choose test framework None
? Choose rendering mode Single Page App
? Choose development tools jsconfig.json (Recommended for VS Code)
```

プロジェクトが作成されました。

起動する

早速起動してみます。
webappのディレクトリに移動して、yarn dev で起動します。

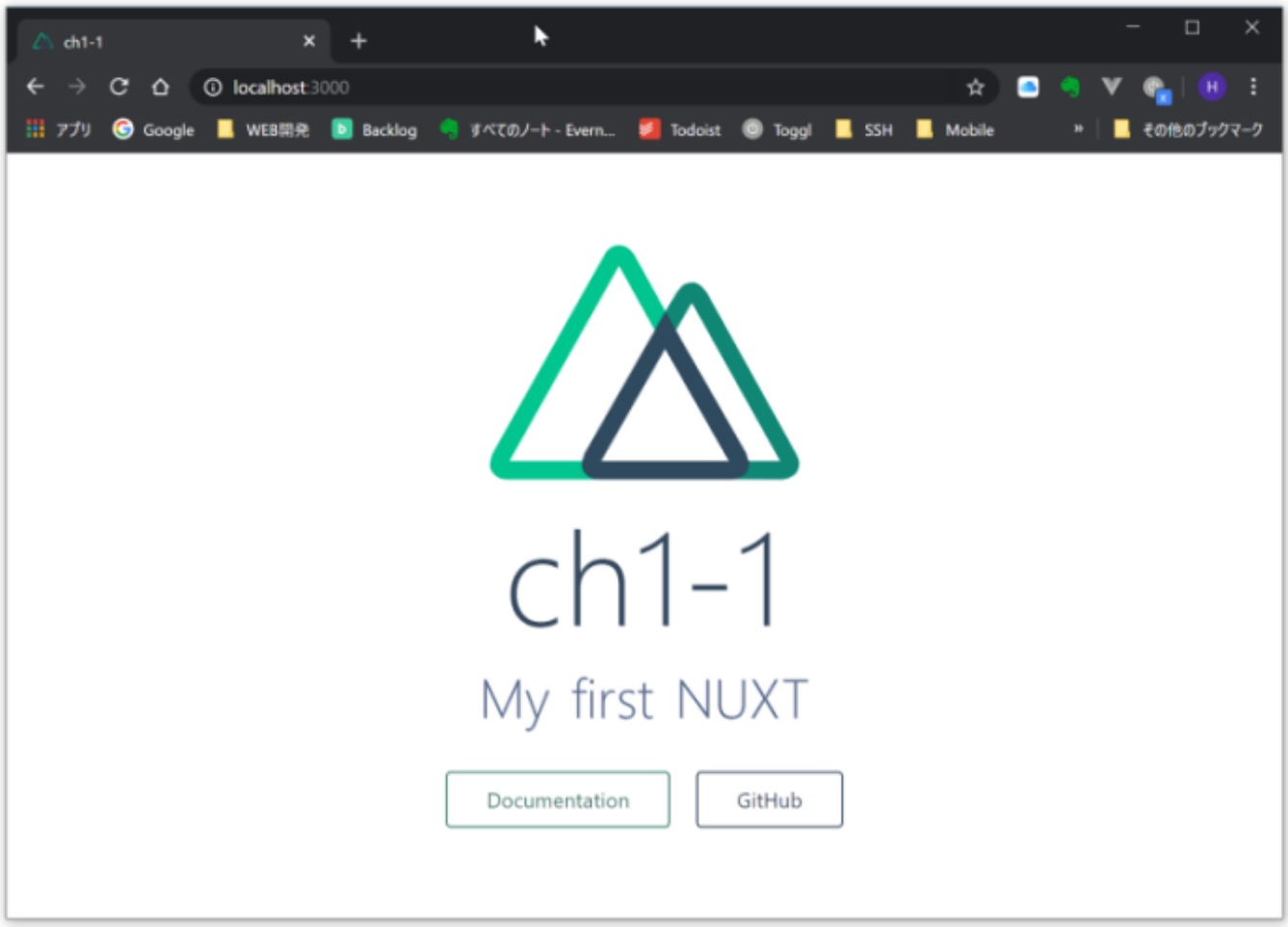
```
CMD> cd webapp
CMD> yarn dev
```

```
    :
    :
i Listening on: http://localhost:3000/
```

Listening on ... と表示されれば、起動は完了です。
ブラウザを立ち上げで、以下のURLにアクセスします。
(ブラウザはフロントエンドのデバッグが容易なchromeをお勧めします)

```
http://localhost:3000/
```

無事、画面が表示されたでしょうか？



終了するにはコマンドラインから `Ctrl+C` を入力します。

NUXTの解説

NUXTのフォルダ構成について

フォルダ構成を確認します。

```
webapp
    +- .nuxt          : 本番で公開される内容
    +- assets         : コンパイルが必要な静的がファイル(CSSなど)
    +- components     : VUE部品を管理(サンプルではLogo)
    +- layouts        : ページのテンプレートを管理
    +- middleware     : 認証などページを表示する共通前処理を定義
    +- node_modules   : 必要なモジュールを保存(パッケージマネージャが管理)
    +- pages          : 画面モジュールを管理
    +- plugins        : 共通ファンクションなどを管理
    +- static         : コンパイル不要なイメージなどの静的ファイルを管理
    +- store          : Vuexという全体で利用可能なメモリを管理
```

画面構成について

サンプル画面は次の様な構成となっています。

```
layout/default.vue
    +- pages/index.vue
        +- componets/Logo.vue
```

テンプレートのソースコード `layouts/default.vue` で確認します。

```
<template>
  <div>
    <nuxt />     ... このタグにpages/内のページが読み込まれます。
  </div>
</template>
```

`pages/index.vue` を確認します。
先ほどの `<nuxt />` 部分に下記の `<template>~</template>` が展開されます。

```
<template>
  <div class="container">
    <div>
      <logo />        ... components/logo.vue のコンポーネントが展開されます。
      <h1 class="title">
        webapp
      </h1>
      <h2 class="subtitle">
        My first NUXT
      </h2>
      <div class="links">
        <a href="https://nuxtjs.org/" target="_blank" class="button--green">
          Documentation
        </a>
        <a href="https://github.com/nuxt/nuxt.js" target="_blank" class="button--
grey">
          GitHub
        </a>
      </div>
    </div>
  </div>
</template>
```

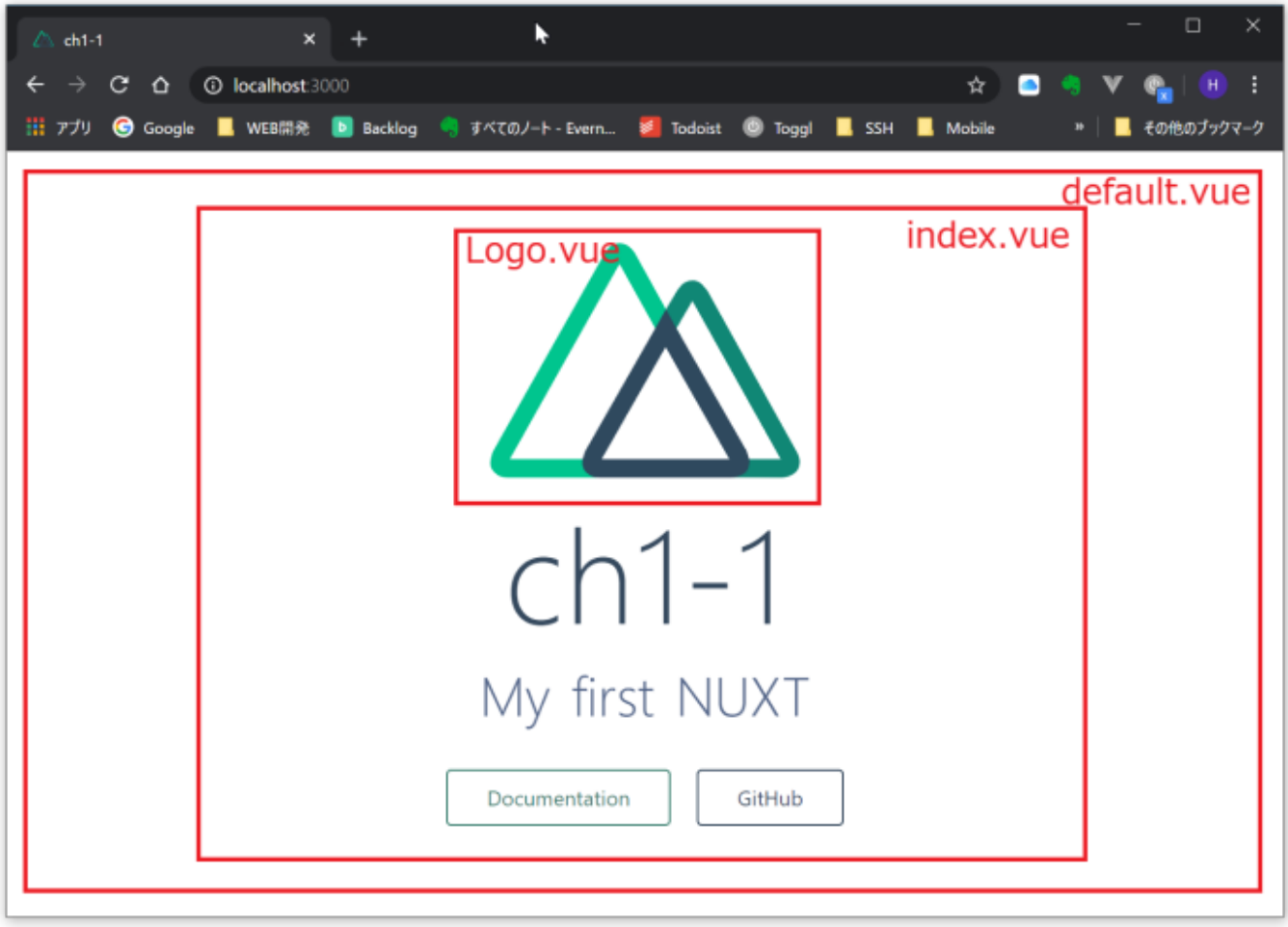
このように、ページの構成要素を分解し、階層構造を持ったコンポーネントに分解することで、 共通性の高い部品・処理を再利用できるようになります。
また、ElementUIのようなUIフレームワークを導入すると、汎用性・機能性の高い部品を簡単に組み込むことができます。
さらに、それらを組み合わせて、例えばファイルアップロード用のダイアログを作れば、その部品を使うだけで、サーバーにファイルをアップロードするところまではコーディングレスで実現することさえ出来ます。

コンポーネント設計をうまく行うことで、非常に高い生産性を実現する事が出来ます。(設計が難しいですが...)

ページ内で使用するコンポーネントは、 `<script>~</script>` 部分で利用を宣言します。

```
<script>
import Logo from '~/components/Logo.vue'     ...    コンポーネントの実体

export default {
  components: {
    Logo       ... 何というタグで利用するかを宣言
  }
}
</script>
```



Vueの構成について

Vueの構成についてまとめると、以下のようになります。

```
<template>
    :            画面に表示する内容をタグ形式で表現します
</template>
```

```
<script>
    :           バックエンドとのやりとりや、画面内のデータを
    :           JavaScriptで記述します
</script>

<style>
    :           画面表示する要素の色や大きさ、配置をCSSで指定します
</stye>
```

画面のデザインを意図した通りに作りこむには、CSSの知識も必要になります。

リアクティブとは？

NUXTのベースであるVueの特徴 **「リアクティブ」** の概念について初期ページに手を加えて確認します。
リアクティブとは簡単に言えば、「変化する変数の内容をリアルタイムに画面表示する」ものです。

これにより、例えばテーブルのデータとしてバインドした変数に、RESTAPIから取得した内容を渡してやると、即座にデータが反映されるようになります。

どのような感じなのか、簡単な例で見てみましょう。

h2要素の"My first NUX"の下に、次のように要素を追加します。

```
<div class="links">
  <p>{{ message }}</p>
  <input v-model="message" />
</div>
```

script部分にmessageの定義を追加します。

```
export default {
  components: {
    Logo
  },
  data() {
    return {
      message: ''
    }
  }
}
```

入力欄に入力した文字がリアルタイムで上部のメッセージに反映されるのがわかります。

従来のWEB画面では表示内容を書き換えるのにリロードが必要でしたが、VueではDOMの状態を管理しており、バインドされた変数に変化があればその部分だけを書き換えるように出来ています。

次回以降の準備

実習用ページの追加

実習用ページとして、**「pages/items.vue」** を追加します。

```
<template>
  <div>
    <el-row type="flex" justify="center" style="margin-top: 2rem;">
      <el-col :span="2">
        <div></div>
      </el-col>
      <el-col :span="20">
        <div>
          <h2>Items</h2>
        </div>
      </el-col>
      <el-col :span="2">
        <div></div>
      </el-col>
    </el-row>
  </div>
</template>
```

参考URL: ElementUI/Layout
https://element.eleme.io/#/en-US/component/layout

ElementUIのグリッドシステムは、行を `<el-row>` で表現し、その行中での列を `<el-col>` として表現します。
行内を24個のグリッドと考えて、spanの合計が24になるようにレイアウトします。(足りなくても問題ないです)。
上記の例では、2:20:2に3列作成しています。
また、ページ上部に少し余裕を持たせるために、CSSで次のようにスタイル指定しています。
`style="margin-top: 2rem;"`
共通部品として作成する場合は、`<style>~</style>` 内に記述するほうが良いです。

メニューの追加

Single Page Application ですが、一応メニューも追加してみます。

メニューシステムは各ページが共通して利用するため、default.vueで定義します。

「layout/default.vue」 ElementUIの水平メニューを追加します。

`<templeta>~</template>` 内に、`<el-menu>~</el-menu>` 部分を追加します。

```
<template>
  <div>
    <el-menu :default-active="activeIndex" mode="horizontal" router>
      <el-menu-item index="/">About</el-menu-item>
      <el-menu-item index="/items">Items</el-menu-item>
```

```
      </el-menu>
      <nuxt />
    </div>
  </template>
```

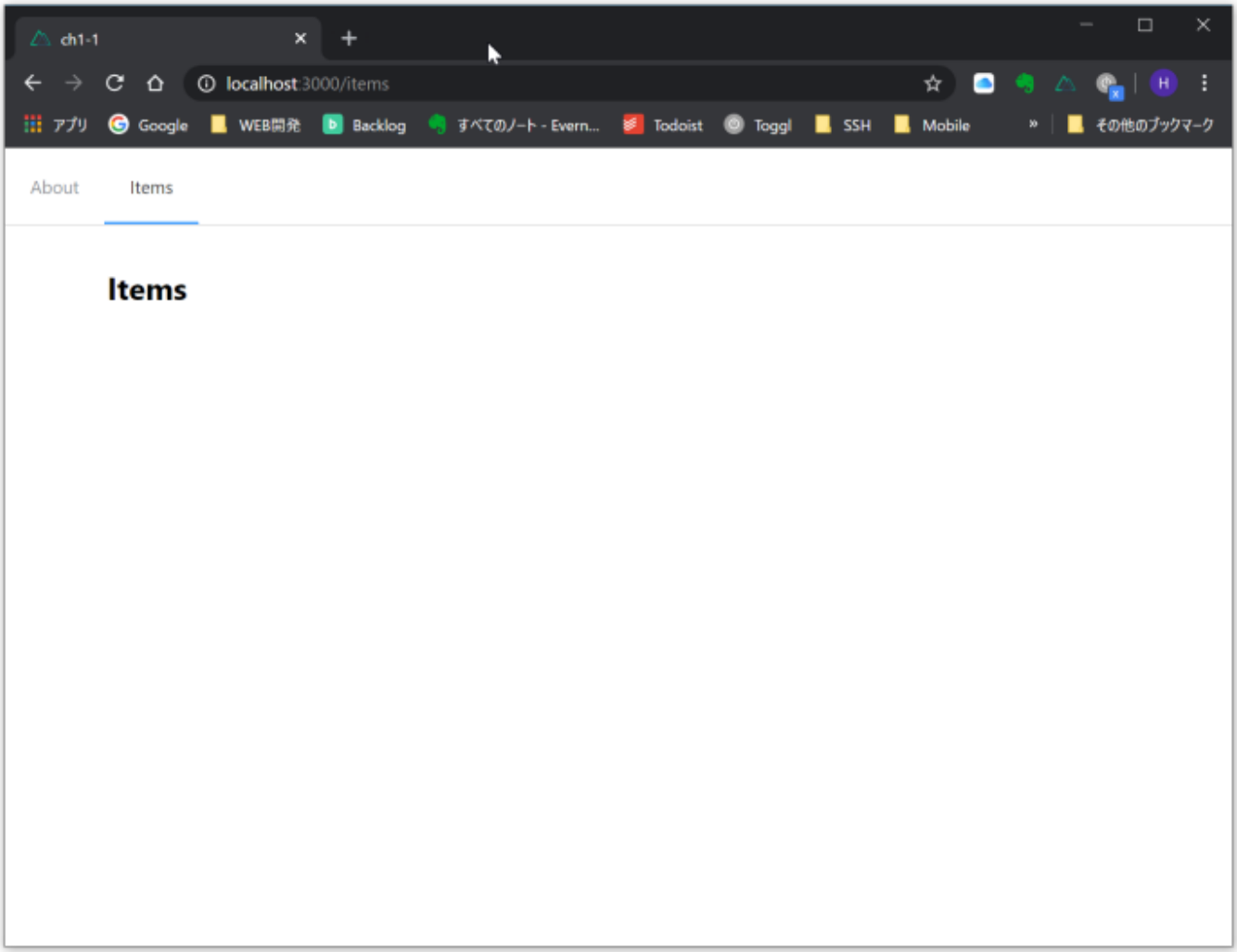
router オプションを定義することで、indexに指定したページに遷移できるようになります。
default-active が、現在選択中のメニューとなります。ここでは、その属性に、activeIndex 変数の内容を設定しています。
これは、遷移先のページメニューを選択状態にするために必要になります。

script部分に下記を追加

```
<script>
export default {
  data() {
    return {
      activeIndex: '/'
    }
  },
  mounted() {
    this.activeIndex = this.$route.path
  }
}
</script>
```

参考URL: ElementUI/Navimenu
https://element.eleme.io/#/en-US/component/menu

data() {...} 内にこのページで利用する変数を定義していきます。
現在のページを表す変数 activeIndex を初期値 '/' で定義しています。
mounted() {...} でページがレンダリングされた後処理として、this.$route.path を取得し、activeIndex に設定しています。
this.$route.pathは現在のページのパスを表し、'/', '/items', '/about' などが入ります。

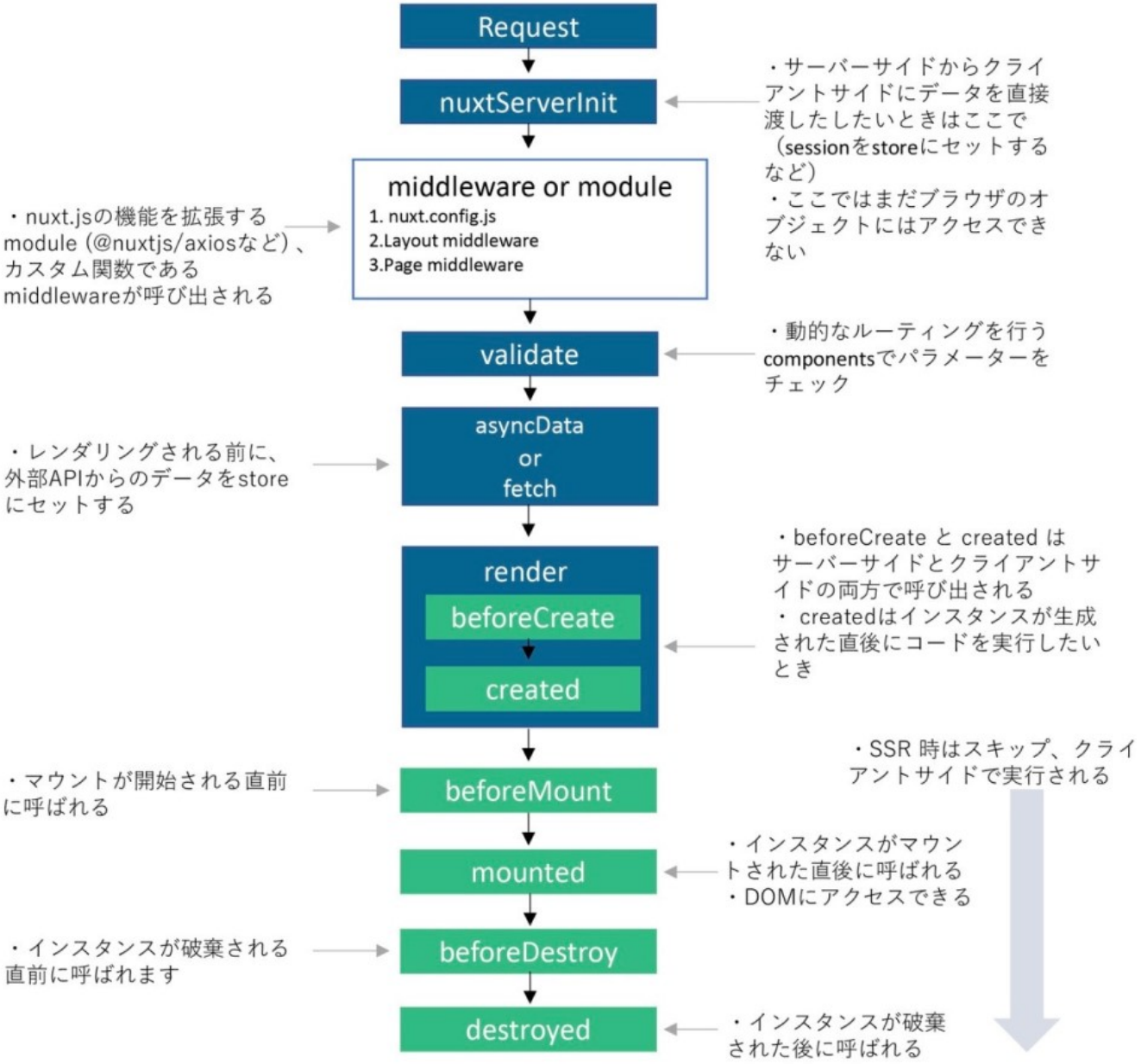


比較的少ない手間で、メニューバーが追加できることを体験頂きました。

参考までに

NUXTはライフサイクルのどの部分で処理したいかによって、先の`mounted`のようなポイントがあります。

Nuxのライフサイクル： Nuxt.jsのライフサイクル覚書より



webapp ソースの取得

historyから圧縮ソースを取得

ソースファイル：勉強会>WEBAPP>50_ETC>app>ch1-1.zip

zipファイルは必要モジュールを同梱していますので、展開後、直ぐに「`yarn dev`」頂けます。(おすすめ)

git hub からソースを取得

git hub使ってみたい人向け(gitインストールが必要)
今回(webapp終了時点)のソースを取得するには、以下のコマンドを実行します。

```
git clone https://github.com/hiszuk/webapp.git -b webapp --depth 1 webapp
```

モジュールダウンロード

git hub から落とした場合は、画面を起動するには必要モジュールのダウンロードが必要です。

```
yarn install
```

ダウンロードできない場合、以下のフォルダよりnode_modulesをコピーしてください。

50_ETC\node_modules