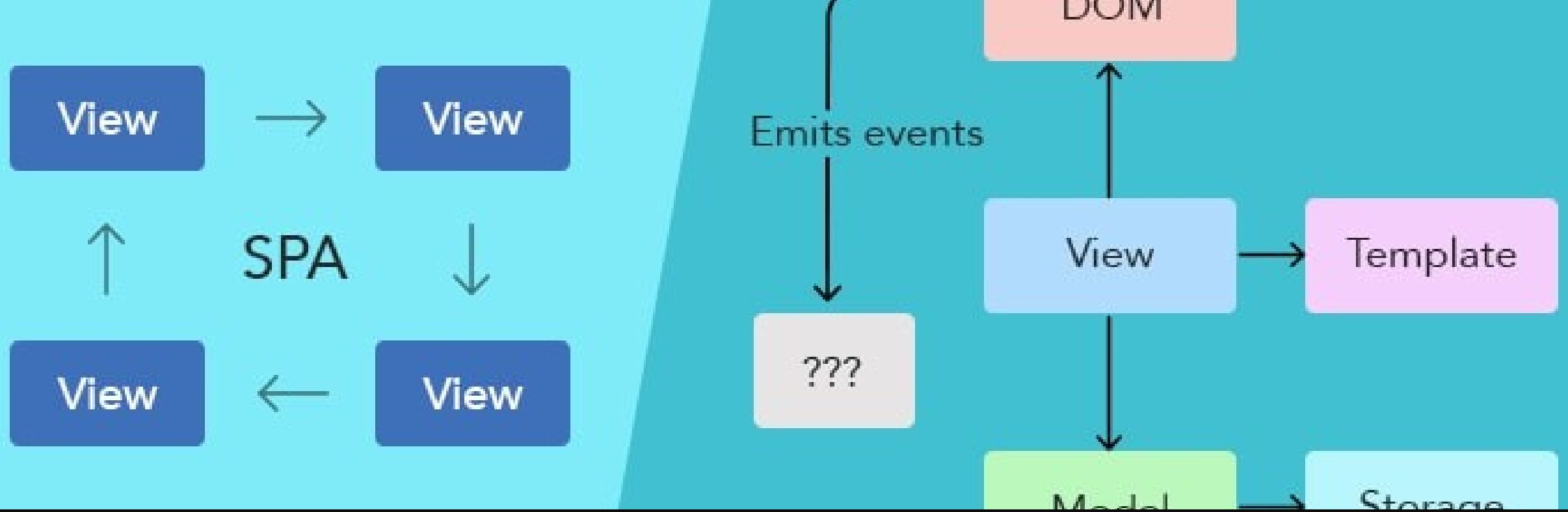


GUI Programming & Single-page App – Vue

Frank Rosbak

Faglærer

E-mail: fros@techcollege.dk



Single-Page Application introduktion

Hvordan virker Vue?

Lad os tage et kig ned under motorhjelm

Hvordan virker Vue?

- Lad os undersøge hvad der egentlig sker, når vi starter en Vue app

Vues virkemåde

1. Hvordan virker magien?



DATAKOMMUNIKATION
EN UDDANNELSE PÅ **TECH**COLLEGE

Hvordan virker Vue?

```
index.html > html > body
1 <html lang="en">
2 <head>
3   <meta charset="UTF-8">
4   <meta name="viewport" content="width=device-width, initial-scale=1">
5   <title>Learning Vue Programming</title>
6   <script src="https://unpkg.com/vue@3.0.2"></script>
7 </head>
8 <body>
9   <h1>Hello, Vue</h1>
10  <div id="app">
11    <p>{{ title }}</p>
12  </div>
13  <script src="app.js"></script>
14 </body>
15 </html>
```

Vi tager udgangspunkt i youtube videoens start, hvor vi laver en Vue app via CDN'en (Content Delivery Network), hvor selve Vue hentes via script linjen i line 6

I line 10 har vi en div med id'et "app"

```
JS app.js > app
1 const app = Vue.createApp({
2   data() {
3     return {
4       title: 'The Final Empire'
5     }
6   }
7 })
8
9 app.mount('#app')
```

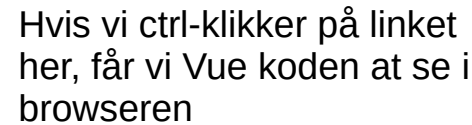
I bodyen loader vi "app.js", Som erklærer en const app der sættes til *Vue.createApp*

Hele cirkuset startes med *app.mount('#app')* der mounter vue app'en i index.html der hvor vi har vores div med id'et "app"

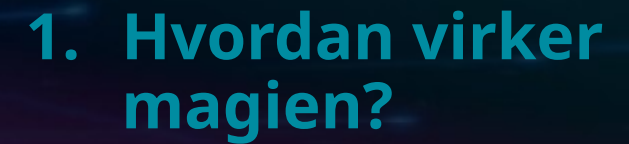
Vues virkemåde

1. Hvordan virker magien?

Vues virkemåde



Det første vi får øje på er 'Vue'



Hvordan virker Vue?

```
index.html > html > body
1 <html lang="en">
2 <head>
3   <meta charset="UTF-8">
4   <meta name="viewport" content="width=device-width, i
5   <title>Learning Vue Programming</title>
6   <script src="https://unpkg.com/vue@3.0.2"></script>
7 </head>
8 <body>
9   <h1>Hello, Vue</h1>
10  <div id="app">
11    <p>{{ title }}</p>
12  </div>
13  <script src="app.js"></script>
14 </body>
15 </html>
```

Hvis vi ctrl-klikker på linket her, får vi Vue koden at se i browseren

```
};
}
let uid$1 = 0;
function createAppAPI(render, hydrate) {
  return function createApp(rootComponent, rootProps = null) {
    if (rootProps != null && !isObject(rootProps)) {
      warn(`root props passed to app.mount() must be an object.`);
      rootProps = null;
    }
    const context = createAppContext();
    const installedPlugins = new Set();
    let isMounted = false;
    const app = (context.app = {
      _uid: uid$1++,
      _component: rootComponent,
      _props: rootProps,
      _container: null,
      _context: context,
      version,
      get config() {
        return context.config;
      },
      set config(v) {

```

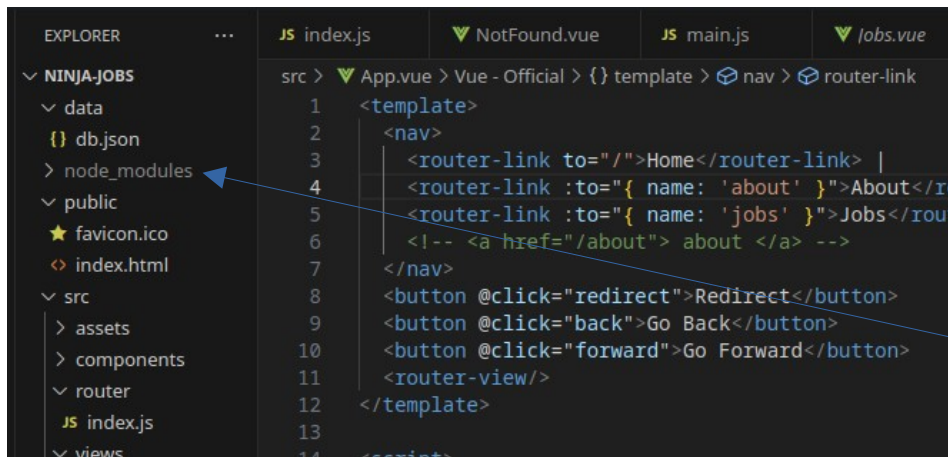
Og hvis vi kigger lidt i vue.global.js, som vi får tilbage, så kan vi se hvor 'createApp' kommer fra.

Hvis vi gider, kan vi se hvordan Vue fungerer, ved at granske hele koden.

Vues virkemåde

1. Hvordan virker magien?

Hvordan virker Vue?



Det var opstarts app'en.
Hvis vi kigger i et projekt der
er en "rigtig" vue app, der er
skabt med 'vue create'.

og kører 'grep -r createApp'
i `node_modules` folderen

```
@vue/cli-service/generator/template/src/main.js:import { createApp } from 'vue'
@vue/cli-service/generator/template/src/main.js:createApp(App).mount('#app')
@vue/cli-service/lib/commands/build/demo-lib.html:Vue.createApp({
@vue/cli-plugin-vuex/generator/injectUseStore.js:  if (j.Identifier.check(node.callee) && node.callee.name === 'createApp'
@vue/cli-plugin-vuex/generator/injectUseStore.js:    node.callee.property.name === 'createApp'
@vue/cli-plugin-vuex/generator/injectUseStore.js:  appRoots.replaceWith(({ node: createAppCall }) => {
@vue/cli-plugin-vuex/generator/injectUseStore.js:    j.memberExpression(createAppCall, j.identifier('use')),
@vue/cli-plugin-router/generator/injectUseRouter.js:  if (j.Identifier.check(node.callee) && node.callee.name === 'createApp'
@vue/cli-plugin-router/generator/injectUseRouter.js:    node.callee.property.name === 'createApp'
@vue/cli-plugin-router/generator/injectUseRouter.js:  appRoots.replaceWith(({ node: createAppCall }) => {
@vue/cli-plugin-router/generator/injectUseRouter.js:    j.memberExpression(createAppCall, j.identifier('use')),
@vue/runtime-core/dist/runtime-core.cjs.js:function createAppContext() {
@vue/runtime-core/dist/runtime-core.cjs.js:function createAppAPI(render, hydrate) {
@vue/runtime-core/dist/runtime-core.cjs.js:  return function createApp(rootComponent, rootProps = null) {
@vue/runtime-core/dist/runtime-core.cjs.js:    const context = createAppContext();
@vue/runtime-core/dist/runtime-core.cjs.js:    If you want to remount the same app, move your app creation logic into a factory
nd create fresh app instances for each mount - e.g. 'const createMyApp = () => createApp(App)``
@vue/runtime-core/dist/runtime-core.cjs.js:  createApp: createAppAPI(render, hydrate)
@vue/runtime-core/dist/runtime-core.cjs.js:const emptyAppContext = createAppContext();
@vue/runtime-core/dist/runtime-core.cjs.prod.js:function createAppContext() {
@vue/runtime-core/dist/runtime-core.cjs.prod.js:  return function createAppAPI(render, hydrate) {
@vue/runtime-core/dist/runtime-core.cjs.prod.js:    return function createApp(rootComponent, rootProps = null) {
@vue/runtime-core/dist/runtime-core.cjs.prod.js:      const context = createAppContext();
@vue/runtime-core/dist/runtime-core.cjs.prod.js:      createApp: createAppAPI(render, hydrate)
@vue/runtime-core/dist/runtime-core.cjs.prod.js:const emptyAppContext = createAppContext();
@vue/runtime-core/dist/runtime-core.cjs.prod.js:function createAppContext() {
```

Kan vi se, at metoden også
er at finde her, og at det er
her vi kal snage, hvis vi vil
kigge under motorhjælmen
på Vue, og se hvordan det
fungerer.

Vues virkemåde

1. Hvordan virker magien?

Hvordan virker Vue?

```
index.html > html > body
1 <html lang="en">
2 <head>
3   <meta charset="UTF-8">
4   <meta name="viewport" content="width=device-width, i
5   <title>Learning Vue Programming</title>
6   <script src="https://unpkg.com/vue@3.0.2"></script>
7 </head>
8 <body>
9   <h1>Hello, Vue</h1>
10  <div id="app">
11    <p>{{ title }}</p>
12  </div>
13  <script src="app.js"></script>
14 </body>
15 </html>
```

```
JS app.js > [ ] app
1 const app = Vue.createApp({
2   data() {
3     return {
4       title: 'The Final Empire'
5     }
6   }
7 })
8
9 app.mount('#app')
```

Men hvordan virker det så?

Jo Den del af dommen der har selectoren id="app",

bliver transpileret af Vue koden, når vi kører koden.

Kører vi CDN udgaven, så sker det når vi vælger 'open with live server', og browseren starter siden med app.js, og app.mount('#app')

Det samme sker når vi bruger 'npm run serve', og vi gemmer ændringer, men her gennemløbes filerne, og vi får fejlmeddelelser hvis der er fejl.

Vues virkemåde

1. Hvordan virker magien?



DATAKOMMUNIKATION
EN UDDANNELSE PÅ **TECH**COLLEGE

Hvordan virker Vue?

```
index.html > html > body
1 <html lang="en">
2 <head>
3   <meta charset="UTF-8">
4   <meta name="viewport" content="width=device-width, initial-scale=1">
5   <title>Learning Vue Programming</title>
6   <script src="https://unpkg.com/vue@3.0.2"></script>
7 </head>
8 <body>
9   <h1>Hello, Vue</h1>
10  <div id="app">
11    <p>{{ title }}</p>
12  </div>
13  <script src="app.js"></script>
14 </body>
15 </html>
```

```
JS app.js > app
1 const app = Vue.createApp({
2   data() {
3     return {
4       title: 'The Final Empire'
5     }
6   }
7 })
8
9 app.mount('#app')
```

Men hvordan virker det så?

Jo Den del af dommen der har selectoren id="app",

Som vi kan se passer vi et objekt til Vue.createApp.

Objektet kan indeholde mange foruddefinerede funktioner/metoder. Typiske funktioner, vi møder i kurset er:

- '**data()**' : data vi vil passe til app'en
- '**methods()**' : metoder vi kalder, f.eks. Når en knap klikkes
- '**computed()**' : hvis vi skal have genberegnet noget på baggrund af data der ændres
- '**watch()**' : kode der skal udføres når noget ændrer sig
- '**component()**' : registrerer en komponent der kan have sine egne *data*, *methods*, *computed*, *template* erklæringer

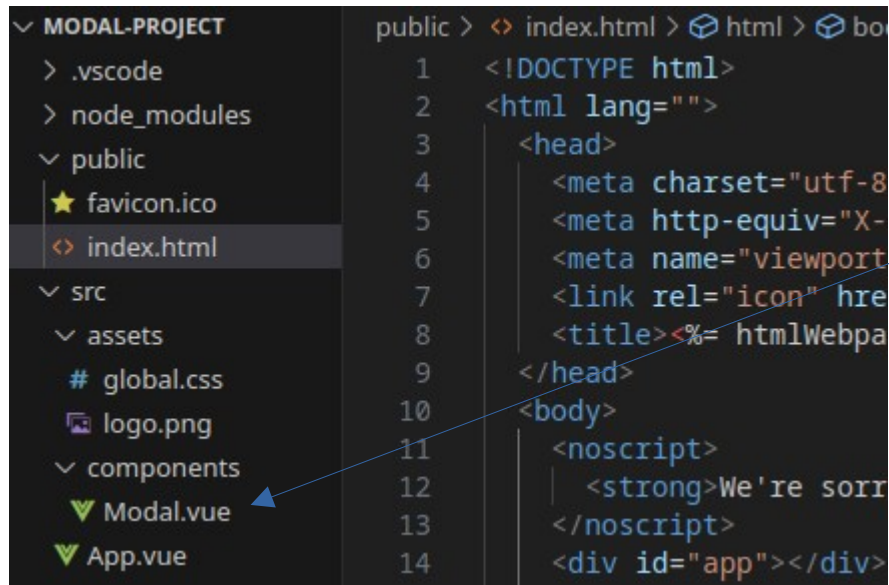
Vues virkemåde

1. Hvordan virker magien?

Hvordan med components?

Målepindende siger noget om 'komponenter'
Lad os se hvordan det fungerer i Vue

Hvordan virker komponenter?



```
public > index.html > html > bo
1  <!DOCTYPE html>
2  <html lang="">
3    <head>
4      <meta charset="utf-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6      <meta name="viewport" content="width=device-width, initial-scale=1" />
7      <link rel="icon" href="favicon.ico" />
8      <title><%= htmlWebpackPlugin.options.title %></title>
9    </head>
10   <body>
11     <noscript>
12       <strong>We're sorry, but your browser does not support JavaScript.</strong>
13     </noscript>
14     <div id="app"></div>
```

Her har vi et projekt hvor der indgår et komponent kaldet Modal. Det er defineret i filen 'Modal.vue'

1. Hvordan virker magien?

Hvordan virker komponenter?

```
c > App.vue > Vetur > {} "App.vue"
1  <template>
2    <h1>{{ title }}</h1>
3    <p>Welcome ...</p>
4    <teleport to="#modals" v-if="showModal">
5      <Modal @close="toggleModal">
6        <template v-slot:links>
7          <a href="#">sign up now</a>
8          <a href="#">more info</a>
9        </template>
10       <h1>Ninja Giveaway!</h1>
11       <p>Get your Ninja swag at half price!</p>
12       <p>Web development is for pussies!</p>
13     </Modal>
14   </teleport>
15
16   <teleport to="#modals" v-if="showModalTwo">
17     <Modal @close="toggleModalTwo">
18       <h1>Sign up for the newsletter!</h1>
19       <p>For updates and promo codes</p>
20     </Modal>
21   </teleport>
22   <button @click="toggleModal">open modal</button>
23   <button @click="toggleModalTwo">open modal Two</button>
24 </template>
25
26 <script>
27 import Modal from './components/Modal.vue'
28
29 export default {
30   name: 'App',
31   components: { Modal },
32   data() {
33     return {
```

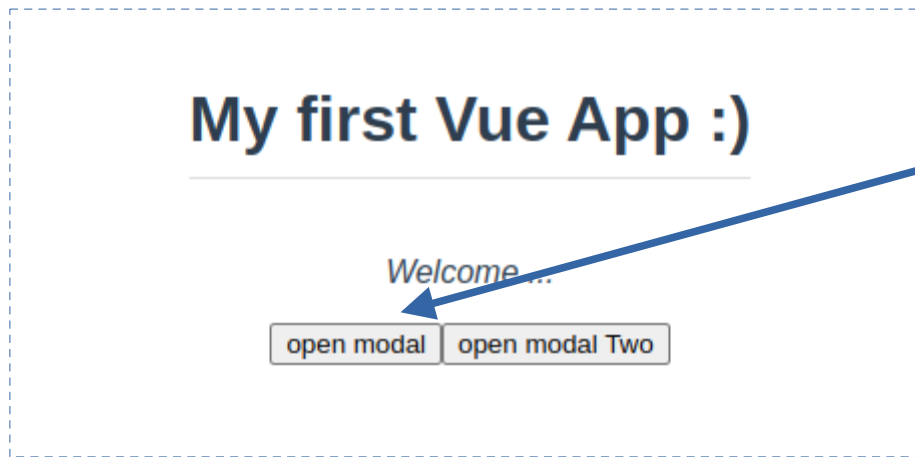
Komponenten 'Modal' (en modal dialog) bruges her ved blot at lave et tag med dens navn

For at have komponenten til rådighed skal den importeres til scriptet

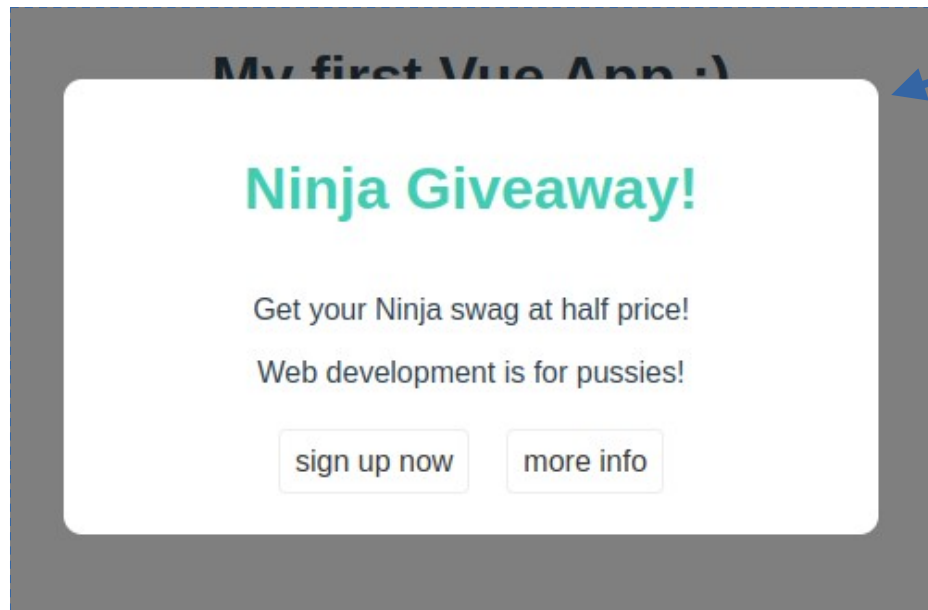
for derefter at exporteres via default objektet

1. Hvordan virker magien?

Hvordan virker komponenter?



Her er knappen



Og når jeg klikker på den får jeg modale dialog (komponenten)

Vues virkemåde

1. Hvordan virker magien?

Hvordan med components? (forts.)

Hvad nu hvis man vil vide når en event indtræffer i et child
component?

undertitel: "kan forældre lære at lytte til børnene :-)"

Events i children?

I eksemplet 'reactiontimer' fra NetNinja, skal vi i App.vue, have at vide hvornår timeren, der findes i cild komponenten 'Block_'.
↓



Vues virkemåde

1. **Behov for at reagere på en event i et child**
2. 'emit()' af eventen og reaktion på den

Events i children?

I Block_.vue, emit'er vi eventen når timeren udløber.

```
28     },
29     stopTimer() {
30       clearInterval(this.timer)
31       this.$emit('end', this.reactionTime)
32     }
33   }
34 }
```

Og vi consumer og reagerer på eventen i App.vue, i template delen

```
1 <template>
2   <h1>Ninja reaction timer</h1>
3   <button @click="start" :disabled="isPlaying">Play!</button>
4   <Block_ v-if="isPlaying" :delay="delay" @end="endGame"/>
5   <Game_Results v-if="showResults" :score="score"/>
6 </template>
7
```

I 'endGame()' sættes 'showResults' true og dermed vises den endelige score.

```
endGame(reactionTime) {
  // reactionTime passed as parameter with 'end' event from Block component
  this.score = reactionTime
  this.isPlaying = false
  this.showResults = true
}
```

Vues virkemåde

1. Behov for at reagere på en event i et child
2. 'emit()' af eventen og reaktion på den

Hvordan med routning?

Målepindende siger også noget om routning
Lad os tage et kig på hvordan det virker i Vue

Hvordan virker routning?

```
Vue CLI v5.0.8
? Please pick a preset: Manually select feat
? Check the features needed for your project
  proceed)
  • Babel
  ○ TypeScript
  ○ Progressive Web App (PWA) Support
  > • Router
  ○ Vuex
  ○ CSS Pre-processors
  • Linter / Formatter
  ○ Unit Testing
  ○ E2E Testing
```

Når projektet oprettes med 'vue create', skal man tilvælge 'router'

```
▼ src
  > assets
  > components
  ▼ router
    JS index.js
  ▼ views
```

Så får man folderen 'router' forærende, med en 'index.js' fil.

1. Hvordan virker magien?

Hvordan virker routning?

```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'
import AboutView from '../views/AboutView.vue'
import NotFound from '../views/NotFound.vue'
import Jobs from '../views/Jobs/Jobs.vue'
import JobDetails from '@views/Jobs/JobDetails.vue'

const routes = [
  {
    path: '/',
    name: 'home',
    component: HomeView
  },
  {
    path: '/about',
    name: 'about',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for the
    // about view which is lazy-loaded when the route is visited.
    component: AboutView
  },
  {
    path: '/jobs/jobs',
    name: 'jobs',
    component: Jobs
  },
  {
    path: '/jobs/jobs/:id',
    name: 'JobDetails',
    component: JobDetails,
    props: true
  },
  {
    // redirect
    path: '/all-jobs',
    redirect: '/jobs/jobs'
  },
  {
    // Catchall 404
  }
]
```

I 'index.js' filen definerer man så sine "endpoints"

Vues virkemåde

1. Hvordan virker magien?

Hvordan virker routning?

```
<template>
  <nav>
    <router-link to="/">Home</router-link> |
    <router-link :to="{ name: 'about' }">About</router-link>
    <router-link :to="{ name: 'jobs' }">Jobs</router-link>
    <!-- <a href="/about"> about </a> -->
  </nav>
  <button @click="redirect">Redirect</button>
  <button @click="back">Go Back</button>
  <button @click="forward">Go Forward</button>
  <router-view/>
</template>

<script>
import HomeView from './views/HomeView.vue';

export default {
  methods: {
    redirect() {
      this.$router.push({ name: 'home' })
    },
    back() {
      this.$router.go(-1)
    },
    forward() {
      this.$router.go(1)
    }
  },
}
</script>
```

I 'App.vue' kan vi nu linke til vores router. Øverst er links til About og Jobs View'ene.

Nedenunder knapper til *home*, *back*, og *forward*

Vues virkemåde

1. Hvordan virker magien?

Hvordan virker routning?

Vues virkemåde

1. Hvordan virker magien?

I 'Jobs.vue' kaldes 'mounted()' når DOM er mounted på app'en, og den læser i det her tilfælde jobs ind fra en json data kilde.

På siden vises '... Loading jobs ...' indtil jobs arrayet har indhold. Herefter vises alle jobs med en Vue udgave af foreach.

```
src > views > Jobs > Jobs.vue > Vue - Official > {} script > default > mounted
1 <template>
2   <h1>Jobs</h1>
3   <div v-if="jobs.length">
4     <div v-for="job in jobs" :key="job.id" class="job">
5       <router-link :to="{ name: 'JobDetails', params: {id: job.id}}">
6         <h2>{{ job.title }}</h2>
7       </router-link>
8     </div>
9   </div>
10  <div v-else>
11    ... Loading jobs ...
12  </div>
13 </template>
14
15 <script>
16 export default {
17   data() {
18     return {
19       jobs: [
20       ]
21     },
22   },
23   mounted(){
24     fetch('http://localhost:3300/jobs')
25       .then(res => res.json())
26       .then(data => this.jobs = data)
27       .catch(err => console.log(err.message))
28   }
29 }
30
31 </script>
```

Hvordan virker routing?

```
.vscode
data
db.json
node_modules
public
favicon.ico
index.html
src
assets
components
router
index.js
views
Jobs
  JobDetails.vue
  Jobs.vue
AboutView.vue
HomeView.vue
NotFound.vue
App.vue
main.js
.browserslistrc
.gitignore
babel.config.js
package.json
```

```
1 <template>
2   <div v-if="job">
3     <h1>{{ job.title }}</h1>
4     <p>Job id is: {{ id }}</p>
5     <p>{{ job.details }}</p>
6   </div>
7   <div v-else>
8     ... Loading job details ...
9   </div>
10 </template>
11
12 <script>
13 export default {
14   props: ['id'],
15   data() {
16     return {
17       job: null
18     }
19   },
20   mounted() {
21     fetch('http://localhost:3300/jobs/' + this.id)
22       .then(res => res.json())
23       .then(data => this.job = data)
24       .catch(err => console.log(err.message))
25   }
26 }
27 </script>
```

I job detail view'et er 'id' erklæret som en 'props'. Det er en værdi child-komponenten JobDetails forventer at få fra sin parent

Her er det id'et på et job i vores json database

id bruges her i opslaget på vores json server

Vues virkemåde

1. Hvordan virker magien?

Hvordan virker routing?

Undrede du dig over at der både var en 'views' folder og en 'components' folder.

'views' – benyttes til komponenter der indgår i et routnings setup. De kan være ret omfattende indholdsmæssigt

'komponenter' – er mere tænkt som delkomponenter beregnet på at blive genbrugt i hele app'en, for at hylde DRY princippet.

Vues virkemåde

1. Hvordan virker magien?



DATAKOMMUNIKATION
EN UDDANNELSE PÅ **TECH**COLLEGE

Hvordan med asynk?

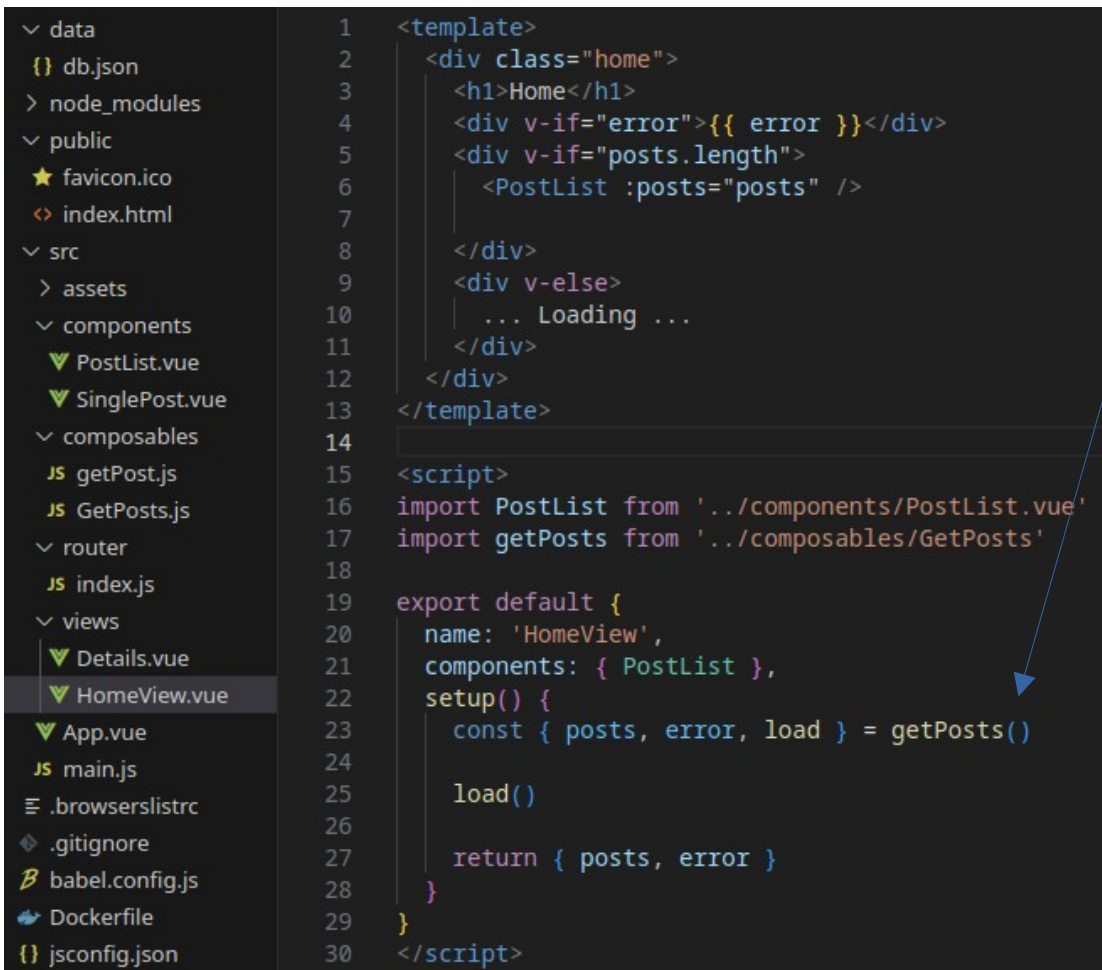
Der står også i målepindende at vi skal
Kunne lave asynkrone kald til en webservice.
Lad os tage et kig på det ...

Asynkrone kald til webservices

1. Hvordan virker magien?

Her får vi en liste af posts fra en blog. Disse posts vises på Home view'et.

Vi får dem vi kaldet til ved at kalde den asynkrone `'load()'` funktion, som bliver returneret til os fra `'getPost.js'` sammen med `'error'` og `'posts'`



```

1  <template>
2    <div class="home">
3      <h1>Home</h1>
4      <div v-if="error">{{ error }}</div>
5      <div v-if="posts.length">
6        <PostList :posts="posts" />
7      </div>
8      <div v-else>
9        ... Loading ...
10     </div>
11   </div>
12 </template>
13
14 <script>
15 import PostList from '../components/PostList.vue'
16 import getPosts from '../composables/GetPosts'
17
18 export default {
19   name: 'HomeView',
20   components: { PostList },
21   setup() {
22     const { posts, error, load } = getPosts()
23
24     load()
25
26     return { posts, error }
27   }
28 }
29 </script>
  
```


Asynkrone kald til webservices

1. Hvordan virker magien?

```
1 import { ref } from 'vue'
2
3 const getPosts = () => {
4   const posts = ref([])
5   const error = ref(null)
6
7   const load = async () => {
8     try {
9       let data = await fetch('http://localhost:3300/posts')
10      if (!data.ok) {
11        throw Error('no data available')
12      }
13      posts.value = await data.json()
14    }
15    catch (err) {
16      error.value = err.message
17    }
18  }
19  return { posts, error, load }
20 }
21
22 export default getPosts
```

I *'getPosts.js'* finder vi *'getPosts'* med sin asynkrone *'load()'* funktion, og i den ses hvordan vi benytter et *'await'* kald til, at hente vores blog posts fra serveren.

Når kaldet returnerer vil der enten være noget i *'error'*, eller der vil være data i *'posts'*

Asynkrone kald til webservices

```
views > ▼ HomeView.vue > Vetur > {} "HomeView.vue"
<template>
  <div class="home">
    <h1>Home</h1>
    <div v-if="error">{{ error }}</div>
    <div v-if="posts.length">
      <PostList :posts="posts" />
    </div>
    <div v-else>
      ... Loading ...
    </div>
  </div>
</template>

<script>
import PostList from './components/PostList'
```

Det benytter vi os af her i
'HomeView.vue'

Hvis der er en 'error' viser vi den.
Hvis der er data i 'posts' viser vi
komponenten *PostList* med alle posts,
eller (eller indtil der kommer posts
data) viser vi
'... Loading ...'

1. Hvordan virker magien?