Pinia

Statemanagement made easy

Pinia

Pinia

Evan You @youyuxi

Replying to @olemarius @VueDose and @vuejs

Pinia is de facto Vuex 5! At this point it's really a naming/branding issue.

10:49 AM · Nov 24, 2021 · Twitter for iPhone

17 Retweets   **4** Quote Tweets   **79** Likes

Pinia

Credits: To keep it simple, I will use a slightly changed Version of https://blog.logrocket.com/complex-vue-3-state-management-pinia/ which is an awesome tutorial.

Pinia

Although Pinia can be considered Vuex 5, there are some important differences between the two you should bear in mind:

- In Pinia, mutations are removed because of their extreme verbosity
- Pinia fully supports TypeScript and offers auto-completion for JavaScript code
- Pinia does not need nested modules, but if one store uses another store, this can be considered implicit nesting
- In Pinia, there is no need to namespace app stores like for Vuex modules
- Pinia **uses Composition API, but can be used with Options API** too
- **Pinia offers server-side rendering (SSR) support**
- Vue 2 or Vue 3 can use Pinia (both with devtools support)

Pinia

The Pinia API is maximally simplified. Here is an example of a basic Pinia store:

```
1  import { defineStore } from 'pinia'
2
3  export const useCounterStore = defineStore({
4    id: 'counter',
5    state: () => ({
6      counter: 0
7    }),
8    getters: {
9      doubleCount: (state) => state.counter * 2
10   },
11   actions: {
12     increment() {
13       this.counter++
14     }
15   }
16 })
```
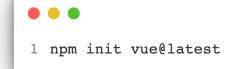
Pinia

To define a store, we use the defineStore function. Here, the word define is used instead of create because a store is not created until it's actually used in a component/page.

Pinia

Pinia also uses the **`state, getters, and actions concepts`**, which are equivalent to **`data, computed, and methods`** in components:

- The state is defined as a function returning the initial state
- The getters are functions that receive the state as a first argument
- The actions are functions that can be asynchronous

Pinia Exercise

To demonstrate Pinia's features, we'll build a basic blog engine with the following features:

- A list of all posts
- A single post page with the post's comments
- A list of all post authors
- A single author page with the author's written posts

```
1  npm init vue@latest
```

Pinia Exercise

```
$ npm init vue@latest

Vue.js - The Progressive JavaScript Framework

√ Project name: ... vue-project
√ Add TypeScript? ... No / Yes
√ Add JSX Support? ... No / Yes
√ Add Vue Router for Single Page Application development? ... No / Yes  ←
√ Add Pinia for state management? ... No / Yes  ←
√ Add Vitest for Unit Testing? ... No / Yes
√ Add Cypress for both Unit and End-to-End testing? ... No / Yes
√ Add ESLint for code quality? ... No / Yes  ←
√ Add Prettier for code formatting? ... No / Yes  ←

Scaffolding project in C:\WEBDEV\Vue\Vue Pinia\Project\vue-project...

Done. Now run:

  cd vue-project
  npm install
  npm run lint
  npm run dev
```

Pinia Exercise

It should print the following to the command line:

```
1 cd 09_pinia/vue-pinia
2 npm install
3
4 npm run dev
```

Pinia Exercise

Let us open main.js to see what has been created:

```
1  import { createApp } from 'vue'
2  import { createPinia } from 'pinia' // Import
3
4  import App from './App.vue'
5  import router from './router'
6
7  const app = createApp(App)
8
9  app.use(createPinia()) // Create the root store
10 app.use(router)
11
12 app.mount('#app')
```
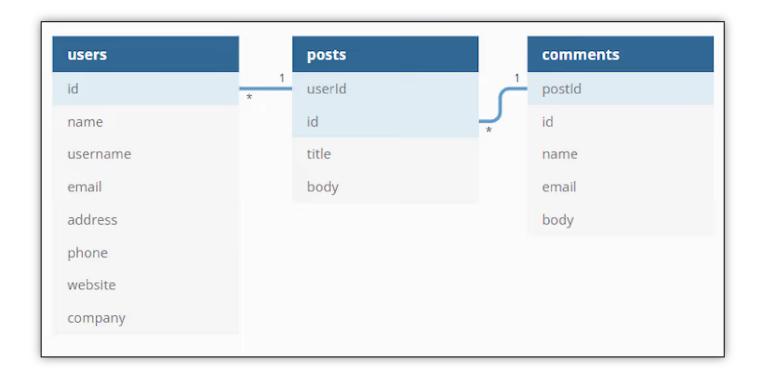
As you can see, the **createPinia** function is imported, creates the Pinia store, and passes it to the app.

Pinia Exercise

Now, open the App.vue file and replace its content with the following:

```
1  <script setup>
2  import { RouterLink, RouterView } from 'vue-router'
3  </script>
4
5  <template>
6    <header class="navbar">
7     <div>
8       <nav>
9         <RouterLink to="/">Posts</RouterLink> -
10        <RouterLink to="/authors">Authors</RouterLink>
11      </nav>
12     </div>
13   </header>
14
15   <RouterView />
16 </template>
```

Pinia Exercise

For our small app, we'll use the JSONPlaceholder service as a data source and these three resources: users, posts, and comments.

Pinia Exercise

To understand how we'll create the app stores better, let's see how these resources relate to each other. Take a look at the following diagram:

Pinia Exercise

The first store we'll define is for blog posts. In the stores directory, rename counter.js to post.js and replace its content with the following:

```javascript
import { defineStore } from 'pinia'
export const usePostStore = defineStore({
  id: 'post',
  state: () => ({
    posts: [],
    post: null,
    loading: false,
    error: null
  }),
  getters: {
    getPostsPerAuthor: (state) => {
      return (authorId) => state.posts.filter((post) => post.userId === authorId)
    }
  },
  actions: {
    async fetchPosts() {
      this.posts = []
      this.loading = true
      try {
        this.posts = await fetch('https://jsonplaceholder.typicode.com/posts')
        .then((response) => response.json())
      } catch (error) {
        this.error = error
      } finally {
        this.loading = false
      }
    },
    async fetchPost(id) {
      this.post = null
      this.loading = true
      try {
        this.post = await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`)
        .then((response) => response.json())
      } catch (error) {
        this.error = error
      } finally {
        this.loading = false
      }
    }
  }
})
```

Pinia Exercise

It is done already - have a look at it!

Also the author.js and comments.js has been done - they are much easier to read.

Pinia Exercise

In the views directory, rename HomeView.vue to PostsView.vue and replace its content with the following:
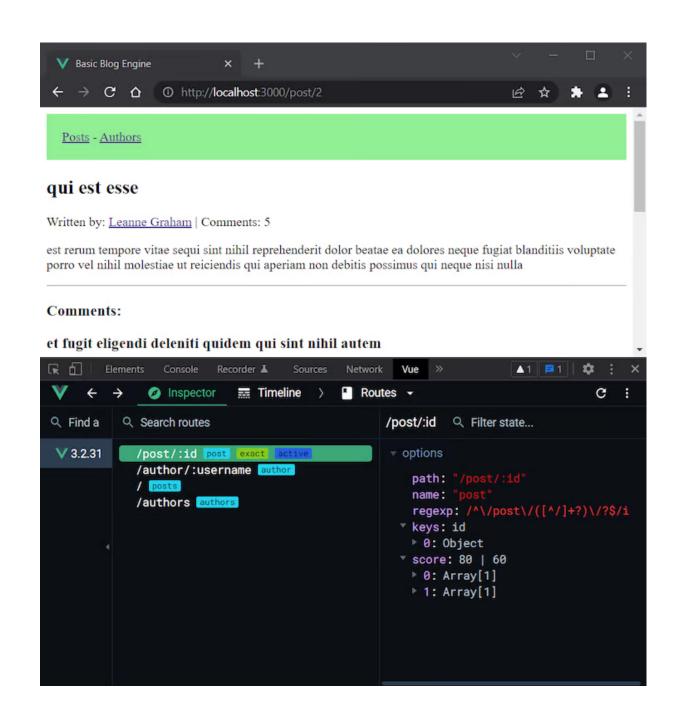
```
1  <script setup>
2    import { RouterLink } from 'vue-router'
3    import { storeToRefs } from 'pinia'
4    import { usePostStore } from '../stores/post'
5
6    const { posts, loading, error } = storeToRefs(usePostStore())
7    const { fetchPosts } = usePostStore()
8
9    fetchPosts()
10 </script>
11
12 <template>
13   <main>
14     <p v-if="loading">Loading posts...</p>
15     <p v-if="error">{{ error.message }}</p>
16     <p v-if="posts" v-for="post in posts" :key="post.id">
17       <RouterLink :to="`/post/${post.id}`">{{ post.title }}</RouterLink>
18       <p>{{ post.body }}</p>
19     </p>
20   </main>
21 </template>
```
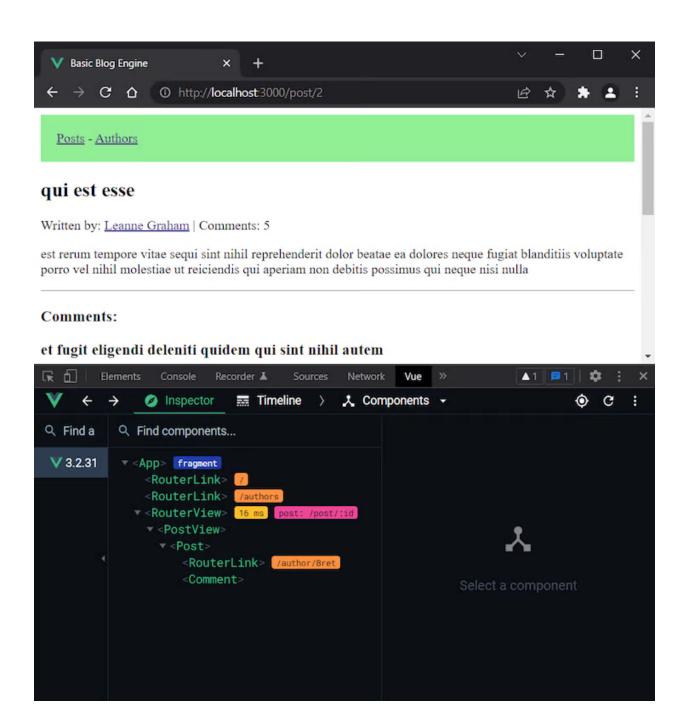
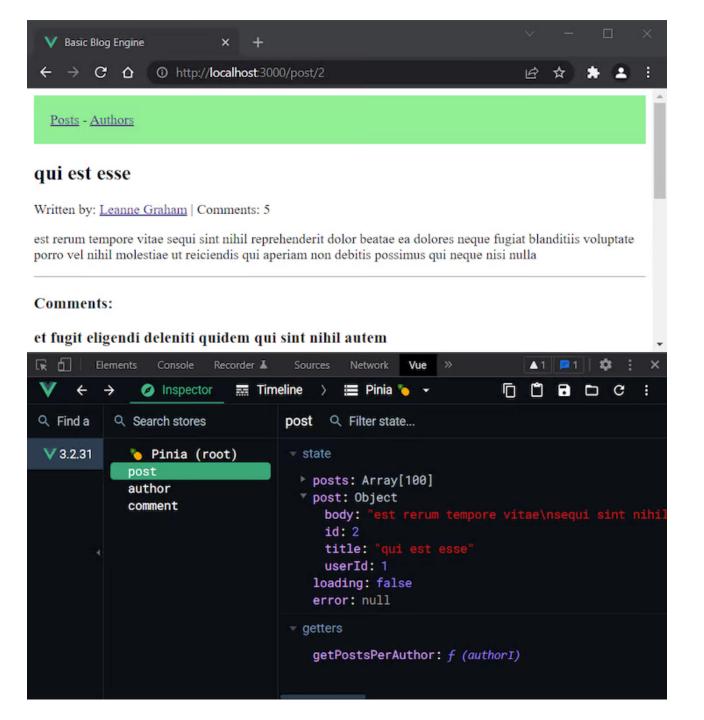# Pinia Exercise

Next we update the router

```
1  import { createRouter, createWebHistory } from 'vue-router'
2  import PostsView from '../views/PostsView.vue'
3
4  const router = createRouter({
5    history: createWebHistory(),
6    routes: [
7      {
8        path: '/',
9        name: 'posts',
10       component: PostsView
11     },
12     {
13       path: '/about',
14       name: 'about',
15       // route level code-splitting
16       // this generates a separate chunk (About.[hash].js) for this route
17       // which is lazy-loaded when the route is visited.
18       component: () => import('../views/AboutView.vue')
19     }
20   ]
21 })
22
23 export default router
```
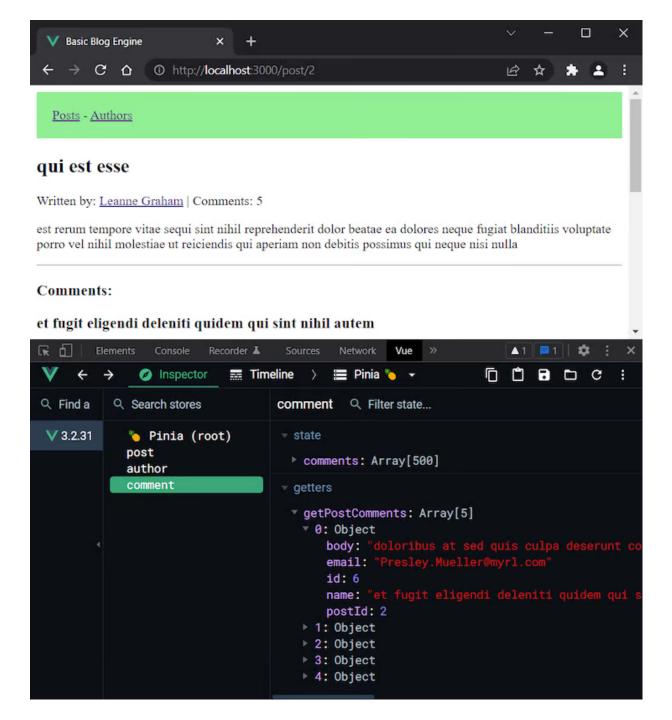
Pinia Exercise

The next few parts are skipped for shortening the process.

Pinia Exercise

The next few parts are skipped for shortening the process.

Pinia Exercise

Pinia Exercise

Pinia Exercise

Pinia Exercise

End

That was all for this chapter