

Pinia

State management made easy

Pinia



Pinia



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
```

name it: **vue-pinia**

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 21_pinia/vue-pinia
2 npm install
3
4 npm run dev
```

Pinia Exercise

Let us open **main.ts** 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>
```

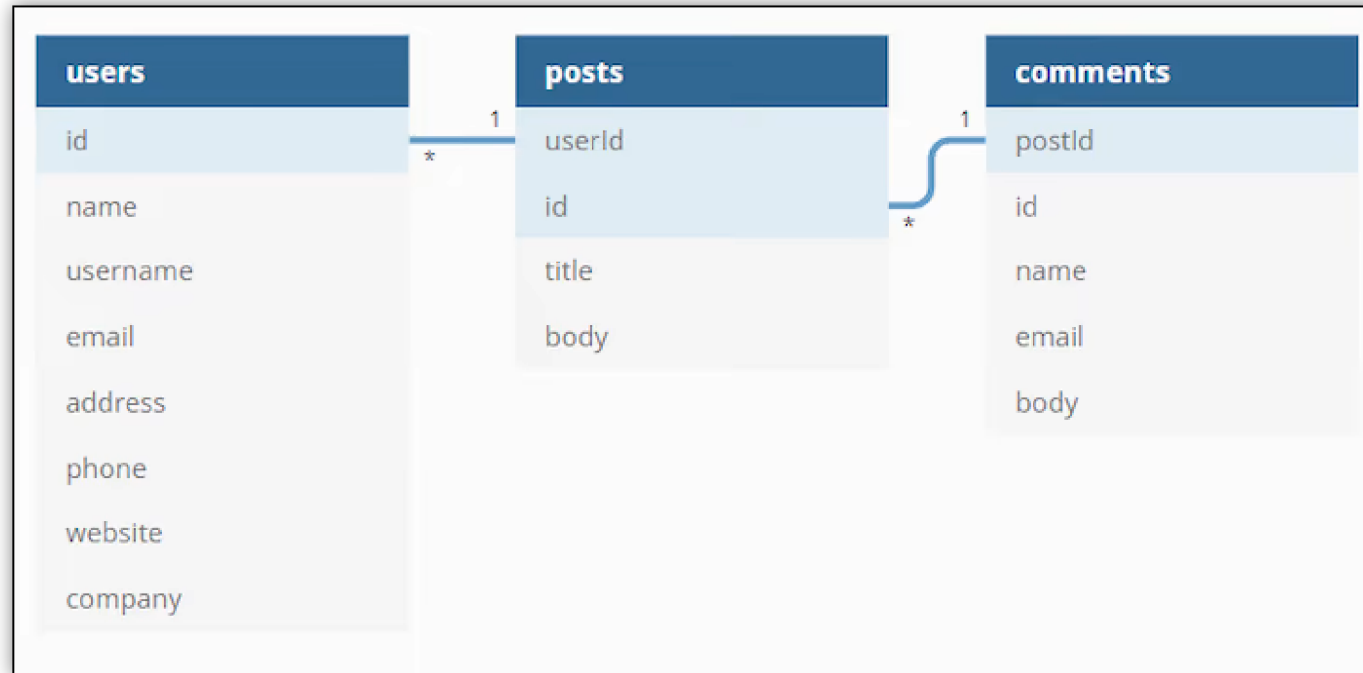
Please delete everything else.

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 thing we need, is a store for an author:

We create author.ts in stores:

```
1 import { defineStore } from 'pinia'
2 import { usePostStore } from '../post'
3
4 export const useAuthorStore = defineStore({
5   id: 'author',
6   state: () => ({
7     authors: []
8   }),
9   getters: {
10    getPostAuthor: (state) => {
11      const postStore = usePostStore()
12      return state.authors.find((author) => author.id === postStore.post.userId)
13    }
14  },
15   actions: {
16    async fetchAuthors() {
17      this.authors = await fetch('https://jsonplaceholder.typicode.com/users')
18        .then((response) => response.json())
19    }
20  }
21 })
```


Pinia Exercise

Next, we will
create
Authors.vue in
components:

```
1 <script setup>
2 import { RouterLink } from 'vue-router'
3
4 defineProps(['author', 'posts'])
5 </script>
6
7 <template>
8   <div>
9     <h1>{{author.name}}</h1>
10    <p>{{posts.length}} posts written.</p>
11    <p v-for="post in posts" :key="post.id">
12      <RouterLink :to="/post/${post.id}">{{ post.title }}</RouterLink>
13    </p>
14  </div>
15 </template>
```

Pinia Exercise

Next, we will fill
AuthorView.vue
in views:

```
1 <script setup>
2 import { RouterLink } from 'vue-router'
3 import { storeToRefs } from 'pinia'
4 import { useAuthorStore } from '../stores/author'
5
6 const { authors } = storeToRefs(useAuthorStore())
7 const { fetchAuthors } = useAuthorStore()
8
9 fetchAuthors()
10 </script>
11
12 <template>
13   <div>
14     <p v-if="authors" v-for="author in authors" :key="author.id">
15       <RouterLink :to="`/author/${author.username}`">{{ author.name }}</RouterLink>
16     </p>
17   </div>
18 </template>
```

Pinia Exercise

Next, we will fill
AuthorsView.vue
in views:

```
1 <script setup>
2 import { computed } from 'vue'
3 import { useRoute } from 'vue-router'
4 import { storeToRefs } from 'pinia'
5 import { useAuthorStore } from '../stores/author'
6 import { usePostStore } from '../stores/post'
7 import Author from '../components/Author.vue'
8
9 const route = useRoute()
10 const { authors } = storeToRefs(useAuthorStore())
11 const { getPostsPerAuthor } = storeToRefs(usePostStore())
12 const { fetchPosts } = usePostStore()
13
14 const getAuthorByUserName = computed(() => {
15   return authors.value.find((author) => author.username === route.params.username)
16 })
17
18 fetchPosts()
19 </script>
20
21 <template>
22   <div>
23     <author
24       :author="getAuthorByUserName"
25       :posts="getPostsPerAuthor(getAuthorByUserName.id)">
26     </author>
27   </div>
28 </template>
```

Pinia Exercise

Adapt the router

```
1 import { createRouter, createWebHistory } from 'vue-router'
2 import PostsView from '../views/PostsView.vue'
3 import PostView from '../views/PostView.vue'
4 import AuthorView from '../views/AuthorView.vue'
5
6 const router = createRouter({
7   history: createWebHistory(),
8   routes: [
9     {
10      path: '/',
11      name: 'posts',
12      component: PostsView
13    },
14    {
15      path: '/authors',
16      name: 'authors',
17      // route level code-splitting
18      // this generates a separate chunk (About.[hash].js) for this route
19      // which is lazy-loaded when the route is visited.
20      component: () => import('../views/AuthorsView.vue')
21    },
22    { path: '/post/:id', name: 'post', component: PostView },
23    { path: '/author/:username', name: 'author', component: AuthorView },
24  ]
25 })
26
27 export default router
```

Pinia Exercise

The next few parts are skipped for shortening the process.

Pinia Exercise

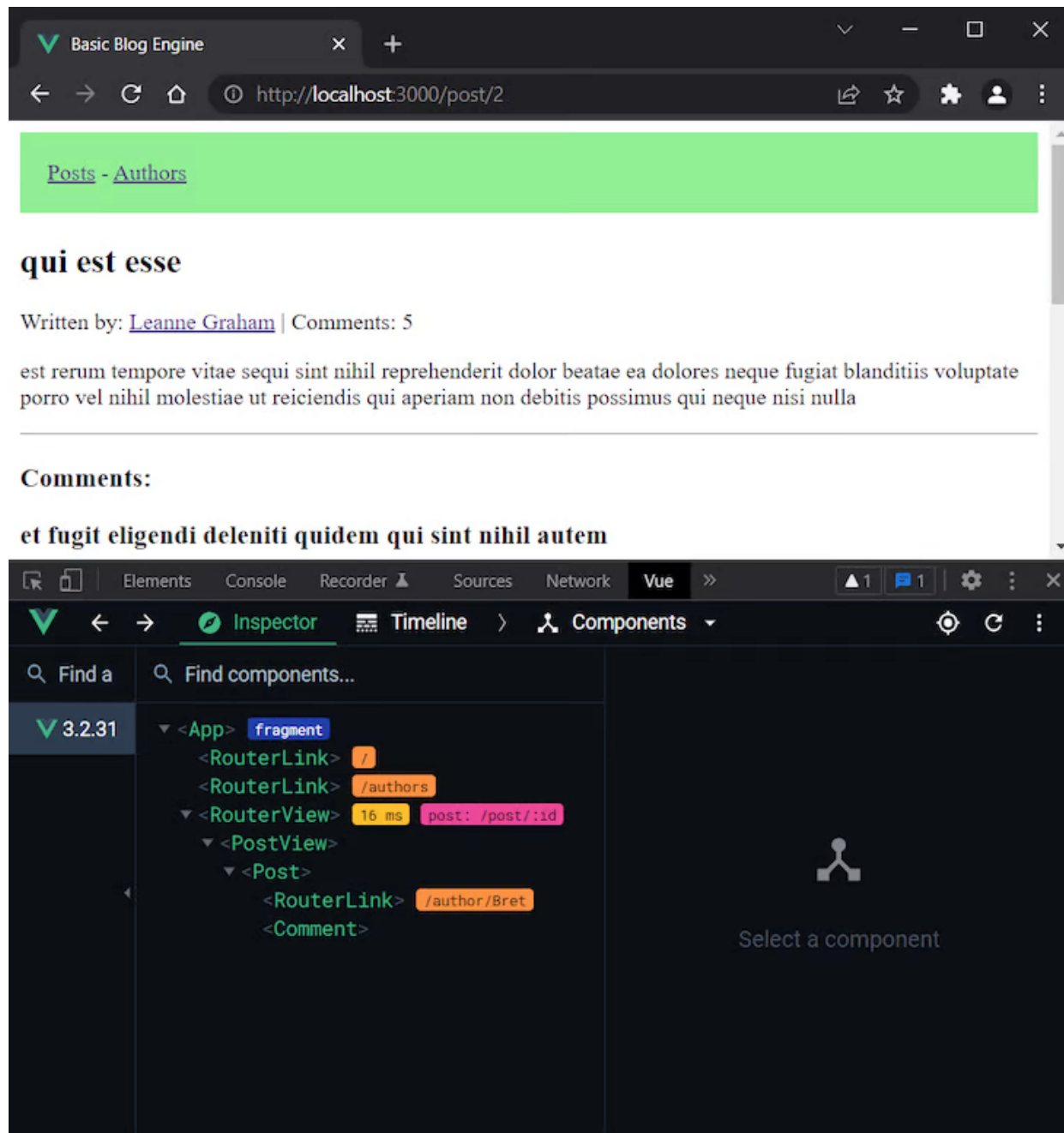
The screenshot displays a web browser window at `http://localhost:3000/post/2` showing a blog post titled "qui est esse" by "Leanne Graham". Below the post content, there is a "Comments:" section. The browser's developer tools are open to the "Routes" tab, showing the active route `/post/:id` with the following configuration:

```
{
  path: "/post/:id",
  name: "post",
  regexp: /^\/post\/(?:\/+)?\/?$/i,
  keys: {
    id: {
      type: "string",
      required: true,
    },
  },
  score: 80,
}
```

The "Inspector" tab also shows the component tree with the following structure:

```
3.2.31
├── /post/:id (post, exact, active)
│   ├── /author/:username (author)
│   └── /posts
│       └── /authors (authors)
```

Pinia Exercise



Pinia Exercise

The screenshot displays a web browser window at `http://localhost:3000/post/2` showing a blog post titled "qui est esse" by "Leanne Graham". The post content is a placeholder Lorem Ipsum text. Below the post, the "Comments" section is visible with the text "et fugit eligendi deleniti quidem qui sint nihil autem".

Below the browser window, the Vue DevTools Pinia Inspector is open, showing the state of the application. The left sidebar shows the "Pinia (root)" store with a tree view containing "post", "author", and "comment". The right pane shows the state of the "post" store, which includes a "state" object and a "getters" object.

```
state
  posts: Array[100]
  post: Object
    body: "est rerum tempore vitae\nsequi sint nihil"
    id: 2
    title: "qui est esse"
    userId: 1
    loading: false
    error: null
  getters
    getPostsPerAuthor: f (authorId)
```


Pinia Exercise

The screenshot displays a web browser window at `http://localhost:3000/post/2` showing a blog post titled "qui est esse" by "Leanne Graham". Below the post content, there is a "Comments:" section. The browser window is overlaid with the Vue DevTools interface, specifically the Pinia state management panel. The Pinia (root) store is visible, containing `post`, `author`, and `comment` properties. The `comment` property is selected, showing its state as an array of 500 comments. The first comment in the array is expanded, showing its details: `body`, `email`, `id`, and `name`.

Basic Blog Engine

[Posts](#) - [Authors](#)

qui est esse

Written by: [Leanne Graham](#) | Comments: 5

est rerum tempore vitae sequi sint nihil reprehenderit dolor beatae ea dolores neque fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis qui aperiam non debitis possimus qui neque nisi nulla

Comments:

et fugit eligendi deleniti quidem qui sint nihil autem

Inspector

Pinia (root)

- post
- author
- comment

comment

state

- comments: Array[500]

getters

- getPostComments: Array[5]
- 0: Object
 - body: "doloribus at sed quis culpa deserunt co"
 - email: "Presley.Mueller@myrl.com"
 - id: 6
 - name: "et fugit eligendi deleniti quidem qui s"
 - postId: 2
- 1: Object
- 2: Object
- 3: Object
- 4: Object

End

That was all for this chapter
