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:

```
import { defineStore } from 'pinia'

export const useCounterStore = defineStore({
  id: 'counter',
  state: () => ({
    counter: 0
  }),
  getters: {
    doubleCount: (state) => state.counter * 2
  },
  actions: {
    increment() {
      this.counter++
    }
  }
})
```

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:

```
cd 21_pinia/vue-pinia
npm install

npm run dev
```

Pinia Exercise

Let us open **`main.ts`** to see what has been created:

```ts
import { createApp } from 'vue'
import { createPinia } from 'pinia' // Import

import App from './App.vue'
import router from './router'

const app = createApp(App)

app.use(createPinia()) // Create the root store
app.use(router)

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:

```
<script setup>
import { RouterLink, RouterView } from 'vue-router'
</script>

<template>
   <header class="navbar">
    <div>
      <nav>
        <RouterLink to="/">Posts</RouterLink> -
        <RouterLink to="/authors">Authors</RouterLink>
      </nav>
    </div>
  </header>

  <RouterView />
</template>
```

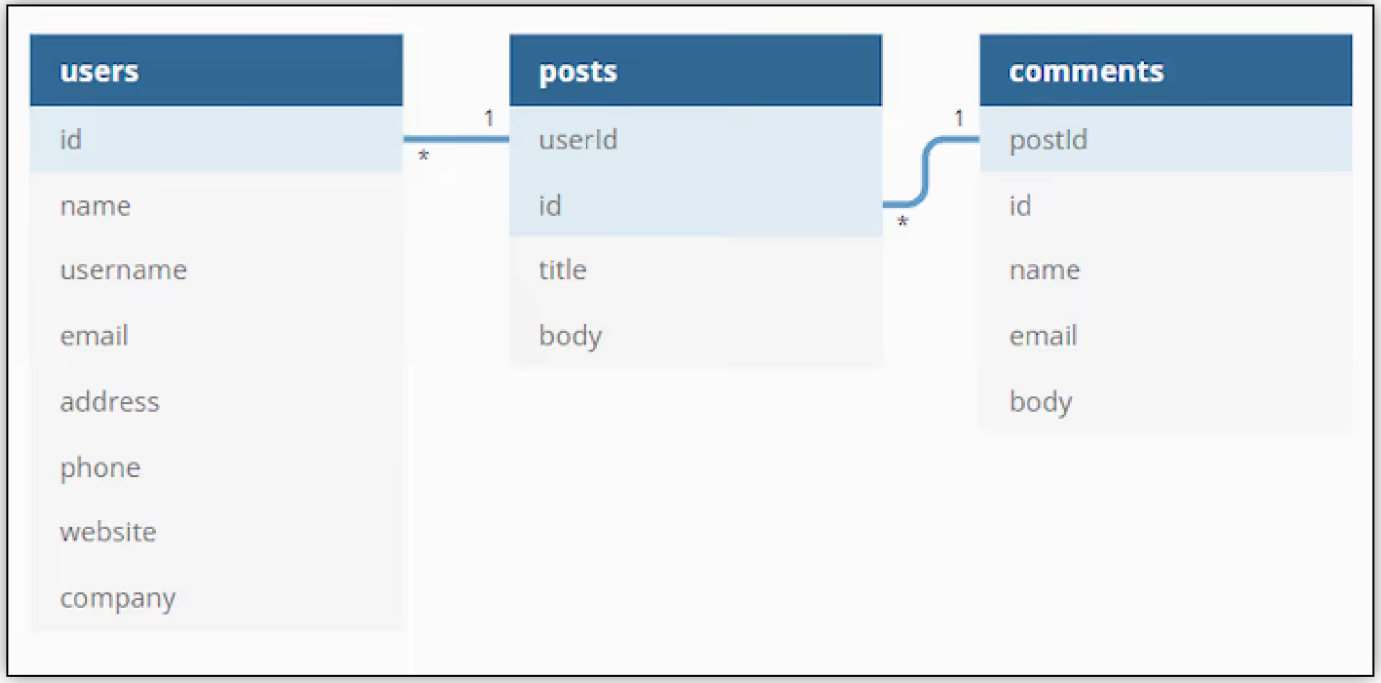
Please delete everything else.

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:

```
import { defineStore } from 'pinia'
import { usePostStore } from './post'

export const useAuthorStore = defineStore({
  id: 'author',
  state: () => ({
    authors: []
  }),
  getters: {
    getPostAuthor: (state) => {
      const postStore = usePostStore()
      return state.authors.find((author) => author.id === postStore.post.userId)
    }
  },
  actions: {
    async fetchAuthors() {
      this.authors = await fetch('https://jsonplaceholder.typicode.com/users')
          .then((response) => response.json())
    }
  }
})
```

Pinia Exercise

Next, we will create Authors.vue in components:

```vue
<script setup>
import { RouterLink } from 'vue-router'

defineProps(['author', 'posts'])
</script>

<template>
  <div>
    <h1>{{author.name}}</h1>
    <p>{{posts.length}} posts written.</p>
    <p v-for="post in posts" :key="post.id">
      <RouterLink :to="`/post/${post.id}`">{{ post.title }}</RouterLink>
    </p>
  </div>
</template>
```

Pinia Exercise

Next, we will fill AuthorView.vue in views:

```vue
<script setup>
import { RouterLink } from 'vue-router'
import { storeToRefs } from 'pinia'
import { useAuthorStore } from '../stores/author'

const { authors } = storeToRefs(useAuthorStore())
const { fetchAuthors } = useAuthorStore()

fetchAuthors()
</script>

<template>
  <div>
    <p v-if="authors" v-for="author in authors" :key="author.id">
      <RouterLink :to="`/author/${author.username}`">{{ author.name }}</RouterLink>
    </p>
  </div>
</template>
```

Pinia Exercise

Next, we will fill AuthorsView.vue in views:

```vue
<script setup>
import { computed } from 'vue'
import { useRoute } from 'vue-router'
import { storeToRefs } from 'pinia'
import { useAuthorStore } from '../stores/author'
import { usePostStore } from '../stores/post'
import Author from '../components/Author.vue'

const route = useRoute()
const { authors } = storeToRefs(useAuthorStore())
const { getPostsPerAuthor } = storeToRefs(usePostStore())
const { fetchPosts } = usePostStore()

const getAuthorByUserName = computed(() => {
  return authors.value.find((author) => author.username === route.params.username)
})

fetchPosts()
</script>

<template>
  <div>
    <author
        :author="getAuthorByUserName"
        :posts="getPostsPerAuthor(getAuthorByUserName.id)">
    </author>
  </div>
</template>
```

Pinia Exercise

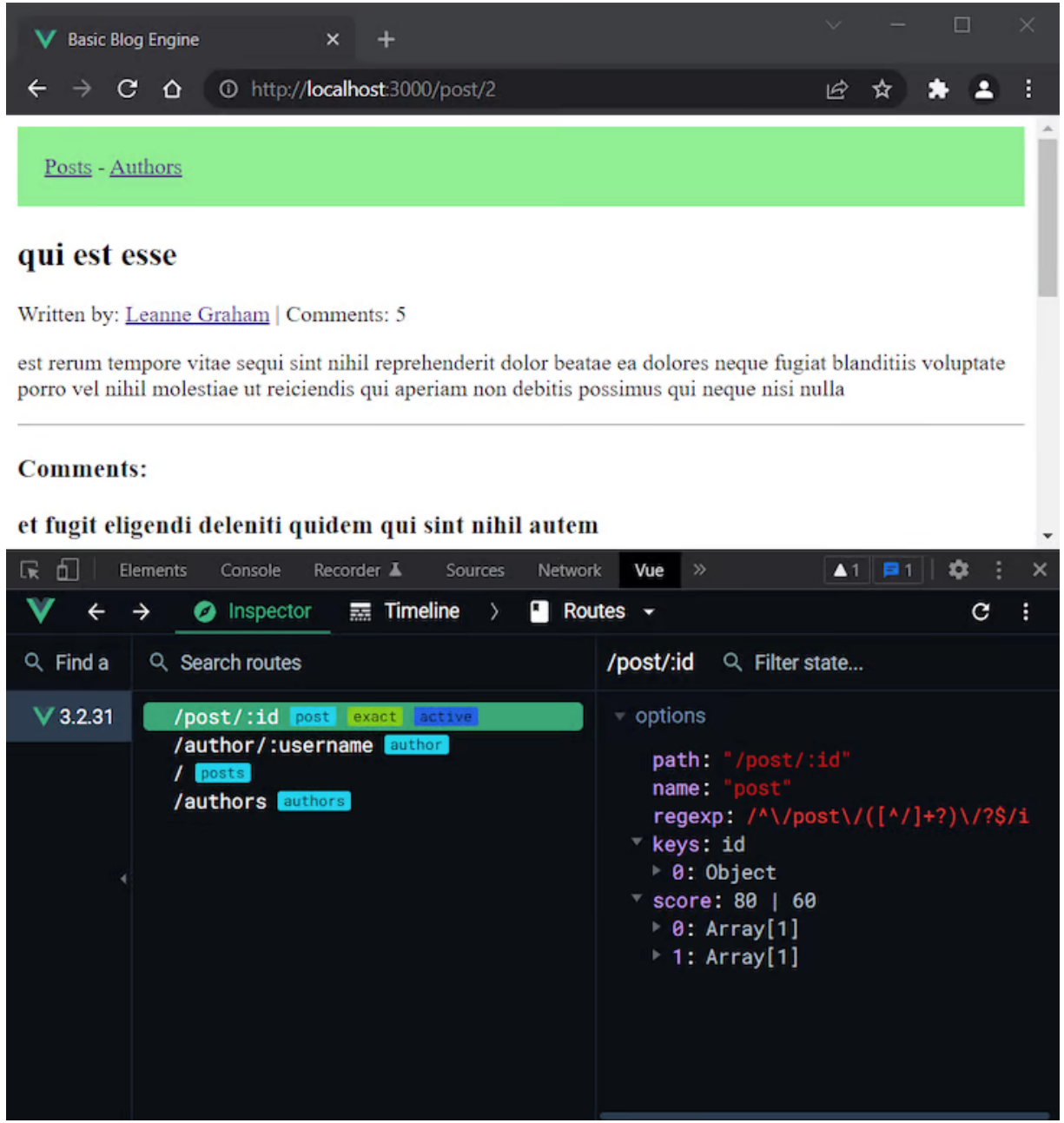
Adapt the router

```
import { createRouter, createWebHistory } from 'vue-router'
import PostsView from '../views/PostsView.vue'
import PostView from '../views/PostView.vue'
import AuthorView from '../views/AuthorView.vue'

const router = createRouter({
  history: createWebHistory(),
  routes: [
    {
      path: '/',
      name: 'posts',
      component: PostsView
    },
    {
      path: '/authors',
      name: 'authors',
      // route level code-splitting
      // this generates a separate chunk (About.[hash].js) for this route
      // which is lazy-loaded when the route is visited.
      component: () => import('../views/AuthorsView.vue')
    },
    { path: '/post/:id', name: 'post', component: PostView },
    { path: '/author/:username', name: 'author', component: AuthorView },
  ]
})

export default router
```

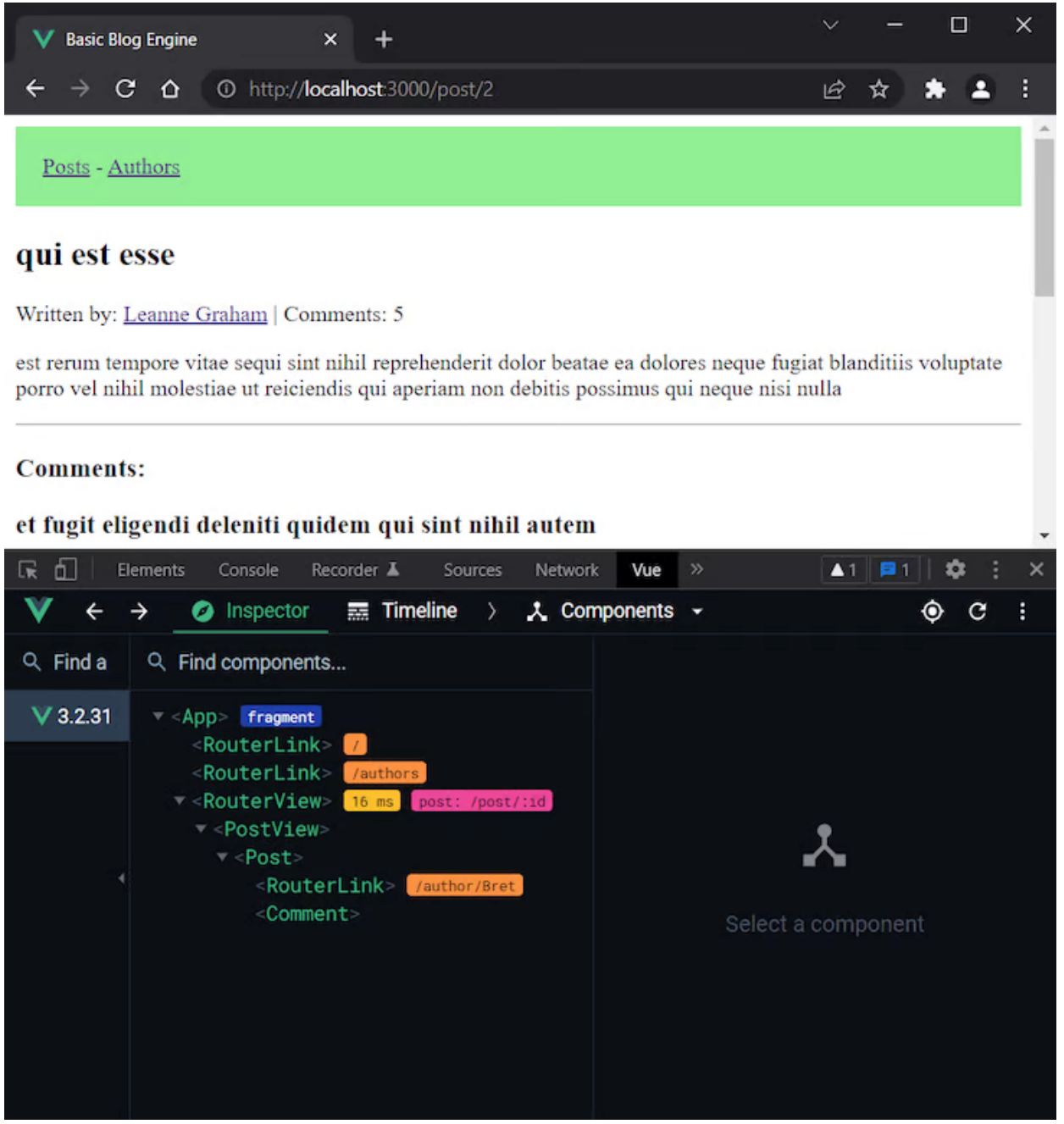
Pinia Exercise

The next few parts are skipped for shortening the process.

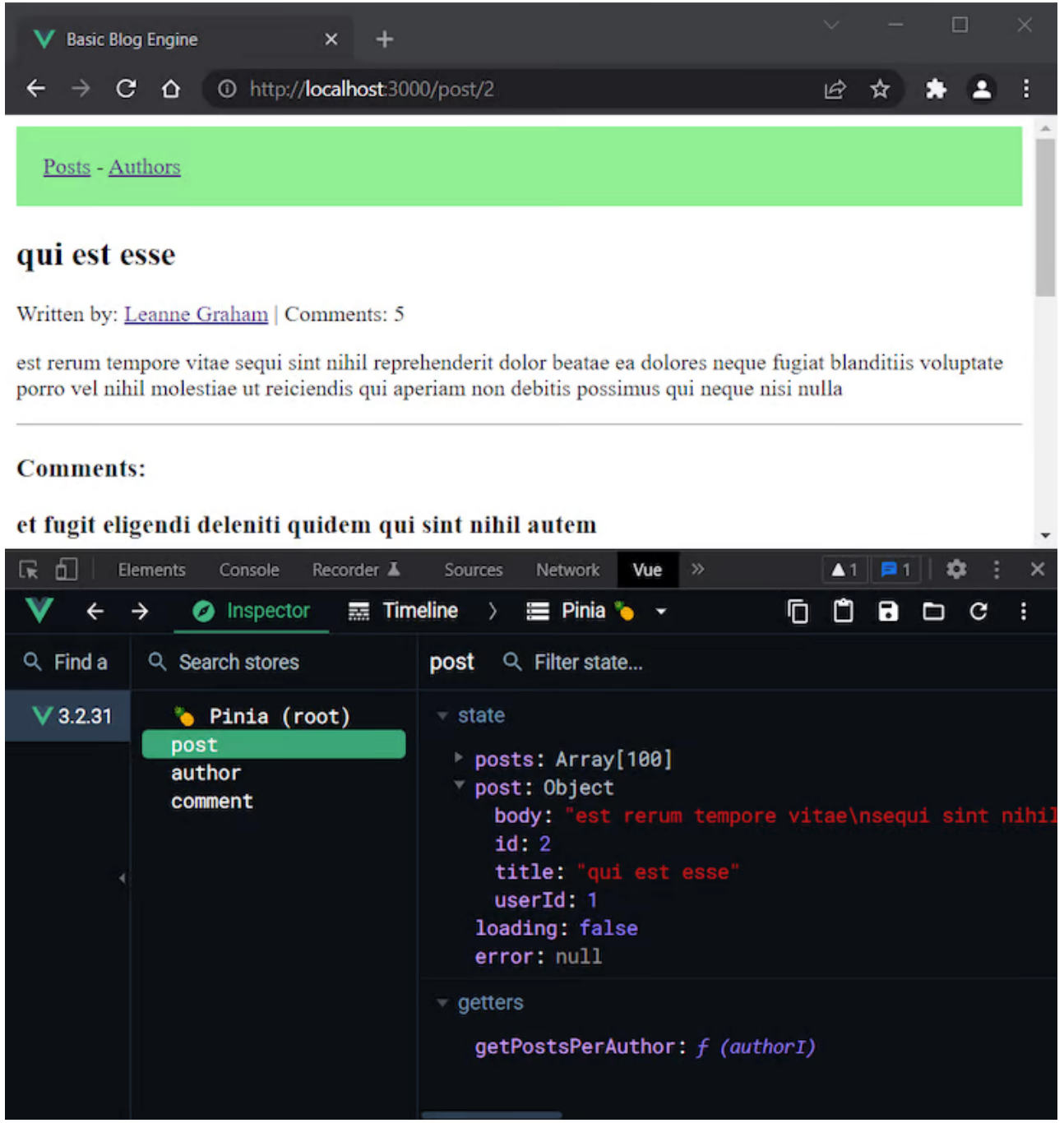
Pinia Exercise



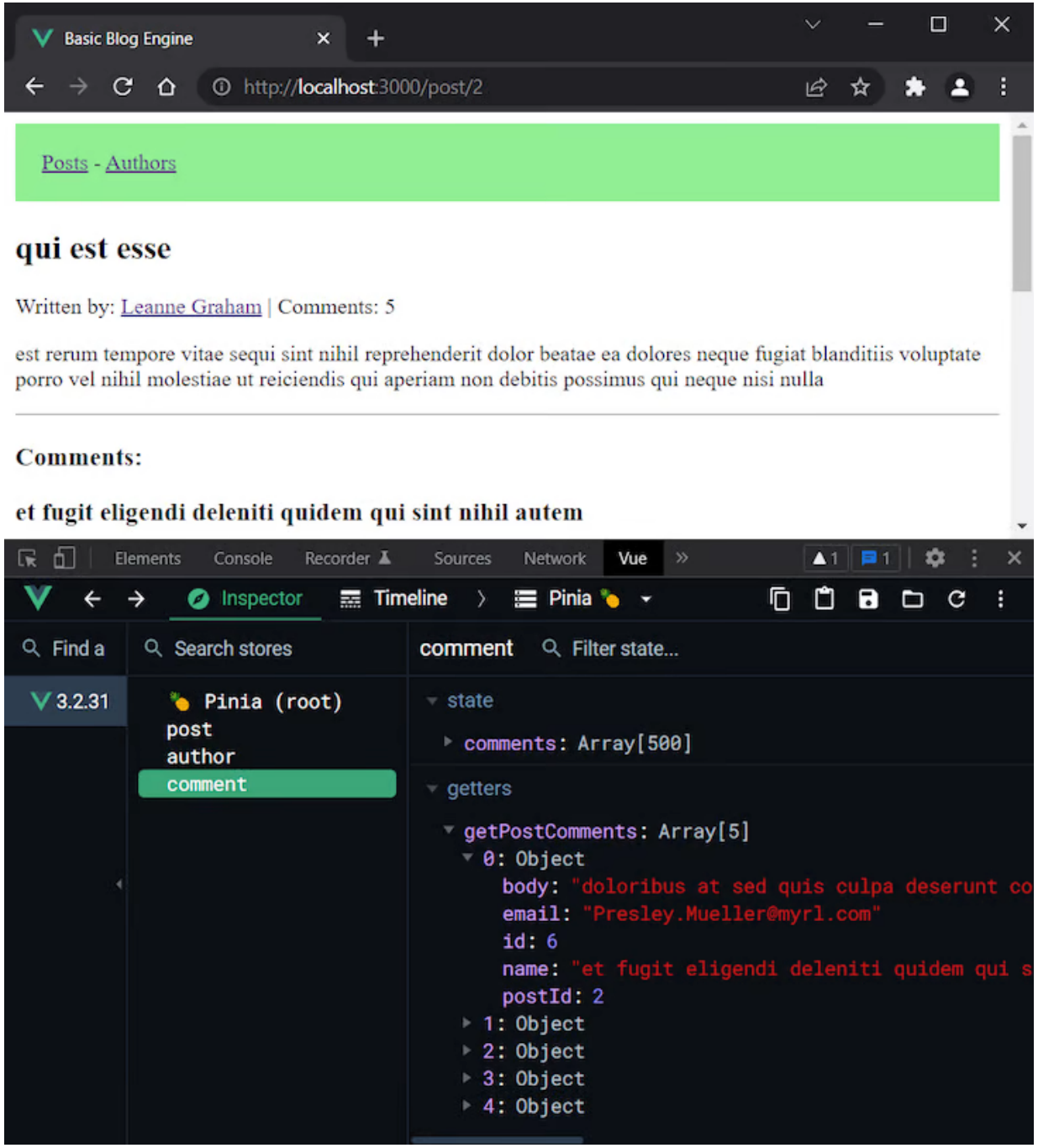
Pinia Exercise



Pinia Exercise



Pinia Exercise



End

That was all for this chapter