

USER

Bun境下，TypeScript、Vite、UnoCSS写一个Todo例程

ASSISTANT

以下是一个使用 Bun、TypeScript、Vite 和 UnoCSS 建的 Todo 用示例：

1. 首先，确保你已安装了 Bun。然后建一个新目：

```
``bash
bun create vite todo-app --template vue-ts
cd todo-app
``
```

2. 安装必要的依：

```
``bash
bun add unocss @unocss/vite
``
```

3. 修改 `vite.config.ts`：

```
``typescript
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'
import UnoCSS from 'unocss/vite'

export default defineConfig({
  plugins: [vue(), UnoCSS()],
})
``
```

4. 建 `uno.config.ts` 文件：

```
``typescript
import { defineConfig } from 'unocss'

export default defineConfig({
  // 可以添加自定或
})
``
```

5. 修改 `src/main.ts`：

```
``typescript
```

```
import { createApp } from 'vue'
import App from './App.vue'
import 'uno.css'
```

```
createApp(App).mount('#app')
...
```

6. 修改 `src/App.vue` :

```
``vue
<template>
  <div class="p-4 max-w-md mx-auto">
    <h1 class="text-2xl font-bold mb-4">Todo App</h1>
    <form @submit.prevent="addTodo" class="mb-4">
      <input v-model="newTodo" type="text" placeholder="Add a new todo" class="border p-2 mr-2">
      <button type="submit" class="bg-blue-500 text-white p-2 rounded">Add</button>
    </form>
    <ul>
      <li v-for="todo in todos" :key="todo.id" class="flex items-center mb-2">
        <input type="checkbox" v-model="todo.completed" class="mr-2">
        <span :class="{ 'line-through': todo.completed }">{{ todo.text }}</span>
        <button @click="removeTodo(todo.id)" class="ml-auto text-red-500">Remove</button>
      </li>
    </ul>
  </div>
</template>
```

```
<script lang="ts">
import { defineComponent, ref } from 'vue'
```

```
interface Todo {
  id: number
  text: string
  completed: boolean
}
```

```
export default defineComponent({
  name: 'App',
  setup() {
    const newTodo = ref('')
    const todos = ref<Todo[]>([])

    const addTodo = () => {
      if (newTodo.value.trim()) {
        todos.value.push({
```

```

      id: Date.now(),
      text: newTodo.value,
      completed: false
    })
    newTodo.value = ""
  }
}

const removeTodo = (id: number) => {
  todos.value = todos.value.filter(todo => todo.id !== id)
}

return {
  newTodo,
  todos,
  addTodo,
  removeTodo
}
}
})
</script>
```

```

## 7. 运行开服务器：

```

```bash
bun run dev
```

```

个的 Todo 应用允许添加新的待事、它已完成，以及删除待事。它使用了 Vue 3 的合式 API 和 TypeScript 来定件，并使用 UnoCSS 来快速置式。

注意，个示例使用了 Vue 3，因它与 TypeScript 和 Vite 配合得很好。如果你想使用其他框架，如 React，你需要相应地整代和配置。

## USER

UnoCSS解

## ASSISTANT

UnoCSS 是一个高性能且极具灵活性的即原子化 CSS 引擎。以下是 UnoCSS 的解析：

### 1. 核心特性：

- 完全可定制：所有都可以定制。
- 属性模式：支持属性模式，可以直接在 HTML 属性中使用。

- CSS：支持 CSS 解决方案。
- 体：支持体，可以更方便地合多个体。
- CSS 作用域：支持 CSS 作用域，可以避免式冲突。
- 代分割：支持按需生成 CSS，减小文件体。

## 2. 基本用法：

在 HTML 中直接使用定的原子类：

```
```html
<div class="text-center font-bold text-2xl color-blue-500">
  Hello UnoCSS
</div>
```
```

## 3. 自定：

在配置文件中自定：

```
```javascript
// uno.config.ts
import { defineConfig } from 'unocss'

export default defineConfig({
  rules: [
    ['custom-rule', { color: 'red' }]
  ]
})
```
```

## 4. ：

UnoCSS 提供了多种，如默、Wind（类似 Tailwind）等：

```
```javascript
import { defineConfig, presetUno, presetAttributify, presetIcons } from 'unocss'

export default defineConfig({
  presets: [
    presetUno(),
    presetAttributify(),
    presetIcons()
  ]
})
```
```

## 5. 主配置：

可以自定义，如色、字体大小等：

```
```javascript
export default defineConfig({
  theme: {
    colors: {
      primary: '#3490dc',
      secondary: '#ffed4a'
    },
    fontSizes: {
      sm: '0.875rem',
      base: '1rem',
      lg: '1.125rem'
    }
  }
})
```
```

## 6. 快捷方式：

定常用的类名：

```
```javascript
export default defineConfig({
  shortcuts: {
    'btn': 'py-2 px-4 font-semibold rounded-lg shadow-md',
    'btn-green': 'text-white bg-green-500 hover:bg-green-700'
  }
})
```
```

## 7. 体：

定条件式：

```
```javascript
export default defineConfig({
  variants: [
    (matcher) => {
      if (!matcher.startsWith('hover:')) return matcher
      return {
        // 理 hover 体
      }
    }
  ]
})
```
```

```
 matcher: matcher.slice(6),
 selector: s => `${s}:hover`
 }
}
]
})
...

```

## 8. 属性模式：

直接在 HTML 属性中使用式：

```
```html
<button
  bg="blue-400 hover:blue-500 dark:blue-500 dark:blue-600"
  text="sm white"
  font="mono light"
  p="y-2 x-4"
  border="2 rounded blue-200"
>
  Button
</button>
```

```

## 9. 支持：

使用 CSS：

```
```html
<div class="i-carbon-airplay text-3xl" />
```

```

## 10. 与构建工具集成：

UnoCSS 可以与 Vite、Webpack、Rollup 等构建工具集成，通插件形式使用。

UnoCSS 的灵活性和高性能使其成代 Web

开中的强大工具。它允开者快速构建自定系，同保持了极高的性能和小的文件体。通其丰富的配置，开者可以根据目需求精确控制生成的 CSS。

## USER

典型的UnoCSS配置案例

## ASSISTANT

以下是一个典型的 UnoCSS 配置案例，包含了多种常用的置和功能：

```

``typescript
// uno.config.ts
import { defineConfig, presetUno, presetAttributify, presetIcons } from 'unocss'
import { presetTypography } from '@unocss/preset-typography'
import { presetWebFonts } from '@unocss/preset-web-fonts'
import transformerDirectives from '@unocss/transformer-directives'
import transformerVariantGroup from '@unocss/transformer-variant-group'

export default defineConfig({
 // 启用
 presets: [
 presetUno(),
 presetAttributify(),
 presetIcons({
 scale: 1.2,
 warn: true,
 }),
 presetTypography(),
 presetWebFonts({
 fonts: {
 sans: 'Roboto',
 mono: ['Fira Code', 'Fira Mono:400,700'],
 },
 }),
],

 // 自定快捷方式
 shortcuts: [
 ['btn', 'px-4 py-1 rounded inline-block bg-teal-600 text-white cursor-pointer hover:bg-teal-700 disabled:cursor-default disabled:bg-gray-600 disabled:opacity-50'],
 ['icon-btn', 'text-[0.9em] inline-block cursor-pointer select-none opacity-75 transition duration-200 ease-in-out hover:opacity-100 hover:text-teal-600 !outline-none'],
],

 // 自定
 rules: [
 ['custom-rule', { color: 'red' }],
 [/^text-(.*)$/, ([, c]) => ({ color: c })],
],

 // 主配置
 theme: {
 colors: {
 'primary': '#3490dc',
 }
 }
})

```

```
'secondary': '#ffed4a',
'danger': '#e3342f',
},
breakpoints: {
 sm: '640px',
 md: '768px',
 lg: '1024px',
 xl: '1280px',
 '2xl': '1536px',
},
},
```

// 体

```
variants: [
 (matcher) => {
 if (!matcher.startsWith('hover:')) return matcher
 return {
 matcher: matcher.slice(6),
 selector: s => `${s}:hover`,
 }
 },
],
```

// 安全列表，确保些类始生成

```
safelist: 'prose prose-sm m-auto text-left'.split(' '),
```

// 器

```
transformers: [
 transformerDirectives(),
 transformerVariantGroup(),
],
```

// 自定注入

```
inject: {
 postprocess: (css) => css.replace(/\/\^[^\s\S]*?\^\/g, ''),
},
```

// IDE 自全支持

```
autocomplete: {
 shorthands: {
 // 自定快捷方式的自全
 'btn': 'btn',
 'icon': 'i-',
 },
},
```

```
}}
'''
```

个配置案例包含了以下特性：

1. 预设：使用了多个，包括基的 `presetUno`、属性模式的 `presetAttributify`、支持的 `presetIcons`、排版的 `presetTypography` 和 Web 字体的 `presetWebFonts`。
2. 快捷方式：定了常用的类合，如按和按。
3. 自定：包含了一个静和一个（使用正表达式）。
4. 主配置：自定了色和断点。
5. 体：添加了一个自定的 hover 体理器。
6. 安全列表：确保某些类始被生成，即使它没有在代中被使用。
7. 器：使用了指令器和体器，增强了 UnoCSS 的功能。
8. 自定注入：在个例子中，移除了所有的 CSS 注。
9. IDE 自全支持：自定的快捷方式添加了自全配置。

个配置涵盖了 UnoCSS

的大多数常用功能，可以根据目需求行整和展。它提供了一个良好的起点，可以满足大多数 Web 目的需求，同保持了高度的可定制性。

## USER

合 HTMX，一下 UnoCSS 如何用到前端页面

## ASSISTANT

合 HTMX 和 UnoCSS 可以建出高效、的前端页面，同保持洁的 HTML 构和灵活的式控制。以下是如何将 UnoCSS 用到使用 HTMX 的前端页面的明：

### 1. 目置

首先，确保你的目中已置好了 HTMX 和 UnoCSS：

```
'''html
<head>
 <!-- HTMX -->
 <script src="https://unpkg.com/htmx.org@1.9.6"></script>
 <!-- UnoCSS -->
 <script src="https://cdn.jsdelivr.net/npm/@unocss/runtime/uno.global.js"></script>
```

```
</head>
```

```
'''
```

## 2. 基本 HTML 构

使用 UnoCSS 的原子类来式化你的 HTML 元素：

```
'''html
```

```
<body class="bg-gray-100 p-4">
```

```
<div id="app" class="max-w-2xl mx-auto bg-white shadow-md rounded-lg p-6">
```

```
<h1 class="text-2xl font-bold mb-4">Task List</h1>
```

```
<div id="task-list" class="space-y-2">
```

```
<!-- Tasks will be loaded here -->
```

```
</div>
```

```
<form id="add-task-form" class="mt-4">
```

```
<input
```

```
 type="text"
```

```
 name="task"
```

```
 placeholder="New task"
```

```
 class="border rounded px-2 py-1 w-full mb-2"
```

```
>
```

```
<button
```

```
 type="submit"
```

```
 class="bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600"
```

```
>
```

```
 Add Task
```

```
</button>
```

```
</form>
```

```
</div>
```

```
</body>
```

```
'''
```

## 3. 使用 HTMX 加和提交数据

添加 HTMX 属性来理内容加和表提交：

```
'''html
```

```
<div
```

```
 id="task-list"
```

```
 class="space-y-2"
```

```
 hx-get="/tasks"
```

```
 hx-trigger="load, newTask from:body"
```

```
>
```

```
<!-- Tasks will be loaded here -->
```

```
</div>
```

```

<form
 id="add-task-form"
 class="mt-4"
 hx-post="/tasks"
 hx-target="#task-list"
 hx-swap="beforeend"
>
 <input
 type="text"
 name="task"
 placeholder="New task"
 class="border rounded px-2 py-1 w-full mb-2"
 >
 <button
 type="submit"
 class="bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600"
 >
 Add Task
 </button>
</form>
'''

```

#### 4. 服务器端返回的 HTML

当 HTMX 请求，服务器返回包含 UnoCSS 类的 HTML 片段：

```

'''html
<!-- 个任 -->
<div class="flex items-center justify-between bg-gray-50 p-2 rounded">
 Task description
 <button
 class="text-red-500 hover:text-red-700"
 hx-delete="/tasks/1"
 hx-target="closest div"
 hx-swap="outerHTML"
 >
 Delete
 </button>
</div>
'''

```

#### 5. 内容的式

对于加的内容，确保使用一致的 UnoCSS 类来保持式的一致性。

## 6. 响应式

利用 UnoCSS 的响应式类来建造不同屏幕尺寸的布局：

```
``html
<div id="app" class="max-w-2xl mx-auto bg-white shadow-md rounded-lg p-4 sm:p-6 lg:p-8">
 <!-- 内容 -->
</div>
``
```

## 7. 状态和交互

使用 UnoCSS 的状态类和 HTMX 的事件处理器来响应交互：

```
``html
<button
 class="bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600 active:bg-blue-700"
 hx-post="/tasks"
 hx-trigger="click"
 hx-target="#task-list"
 hx-swap="beforeend"
>
 Add Task
</button>
``
```

## 8. 动画和过渡

结合 UnoCSS 的过渡类和 HTMX 的触发器来添加动画效果：

```
``html
<div
 class="transition-all duration-300 ease-in-out"
 hx-get="/tasks"
 hx-trigger="load"
 hx-swap="innerHTML transition:fade"
>
 <!-- 任务列表 -->
</div>
``
```

## 9. 自定义

如果需要更复杂的样式，可以在 UnoCSS 配置中定义自定义类或使用 @apply 指令：

```

```html
<style>
  @unocss-apply;
  .custom-task-item {
    @apply bg-gray-50 p-2 rounded flex items-center justify-between;
  }
</style>
```

```

## 10. 性能化

UnoCSS 默情况下是按需生成的，与 HTMX 的部分页面更新特性非常契合，可以确保只加必要的 CSS。

通种方式，你可以建出既有 HTMX 交互能力，又具 UnoCSS 灵活式控制的代化前端页面。种合既保持了 HTML 的洁性，又提供了强大的式定制能力，同能享受到两者来的性能。

## USER

我想在才的Todo例程里面，加入Todo目的功能，并且支持Markdown特性

## ASSISTANT

当然,我可以展之前的Todo例程,加入功能和Markdown支持。我将使用一个的Markdown解析器(如marked.js)来Markdown功能。以下是更新后的代:

### 1. 首先,安装必要的依:

```

```bash
bun add marked
```

```

### 2. 更新 `src/App.vue`:

```

```vue
<template>
  <div class="p-4 max-w-2xl mx-auto">
    <h1 class="text-3xl font-bold mb-4">Todo App with Markdown</h1>
    <form @submit.prevent="addTodo" class="mb-4">
      <input v-model="newTodo" type="text" placeholder="Add a new todo" class="border p-2 mr-2 w-full mb-2">
      <button type="submit" class="bg-blue-500 text-white p-2 rounded hover:bg-blue-600">Add</button>
    </form>
    <ul>
      <li v-for="todo in todos" :key="todo.id" class="mb-4 bg-gray-100 p-4 rounded">
        <div v-if="editingId !== todo.id">

```

```

<div class="flex items-center mb-2">
  <input type="checkbox" v-model="todo.completed" class="mr-2">
  <span :class="{ 'line-through': todo.completed }">{{ todo.text }}</span>
  <button @click="startEditing(todo)" class="ml-auto text-blue-500 mr-2">Edit</button>
  <button @click="removeTodo(todo.id)" class="text-red-500">Remove</button>
</div>
<div v-html="renderMarkdown(todo.text)" class="markdown-content text-sm text-gray-600"></div>
</div>
<div v-else>
  <textarea v-model="editingText" class="w-full p-2 border rounded mb-2"></textarea>
  <button @click="saveEdit(todo)" class="bg-green-500 text-white p-2 rounded mr-2">Save</button>
  <button @click="cancelEdit" class="bg-gray-500 text-white p-2 rounded">Cancel</button>
</div>
</li>
</ul>
</div>
</template>

```

```

<script lang="ts">
import { defineComponent, ref } from 'vue'
import { marked } from 'marked'

```

```

interface Todo {
  id: number
  text: string
  completed: boolean
}

```

```

export default defineComponent({
  name: 'App',
  setup() {
    const newTodo = ref("")
    const todos = ref<Todo[]>([])
    const editingId = ref<number | null>(null)
    const editingText = ref("")

    const addTodo = () => {
      if (newTodo.value.trim()) {
        todos.value.push({
          id: Date.now(),
          text: newTodo.value,
          completed: false
        })
        newTodo.value = ""
      }
    }
  }
})

```

```

}

const removeTodo = (id: number) => {
  todos.value = todos.value.filter(todo => todo.id !== id)
}

const startEditing = (todo: Todo) => {
  editingId.value = todo.id
  editingText.value = todo.text
}

const saveEdit = (todo: Todo) => {
  if (editingText.value.trim()) {
    todo.text = editingText.value
    cancelEdit()
  }
}

const cancelEdit = () => {
  editingId.value = null
  editingText.value = ""
}

const renderMarkdown = (text: string) => {
  return marked(text)
}

return {
  newTodo,
  todos,
  editingId,
  editingText,
  addTodo,
  removeTodo,
  startEditing,
  saveEdit,
  cancelEdit,
  renderMarkdown
}
})
</script>

<style>
.markdown-content {

```

```
@apply prose prose-sm;
}
</style>
...
```

3. 更新 `uno.config.ts` 以支持 Markdown 式:

```
``typescript
import { defineConfig, presetUno, presetTypography } from 'unocss'

export default defineConfig({
  presets: [
    presetUno(),
    presetTypography(),
  ],
})
...

```

个更新后的 Todo 用在具有以下新功能:

1. 功能: 用可以点 "Edit" 按来有的 Todo 目。
2. Markdown 支持: Todo 目的文本内容支持 Markdown 格式,并会被渲染 HTML。
3. 改的 UI: 使用了更多的 UnoCSS 类来美化界面。

使用明:

- 添加新的 Todo 目,可以使用 Markdown 法。
- 点 "Edit" 按可以有的 Todo 目。
- ,可以看和修改原始的 Markdown 文本。
- 保存后,Markdown 内容会被重新渲染。

个例子展示了如何将 UnoCSS 与 Vue 3、TypeScript 和 Markdown 功能合使用,建一个功能更丰富的 Todo 用。UnoCSS 的 `prose` 类用于美化渲染后的 Markdown 内容,使其更易。

注意:了安全起见,在用中,你可能需要使用一个 Markdown 解析器,它可以防止 XSS 攻,例如使用 DOMPurify 来清理 HTML 出。

USER

Next.js与Nuxt.js的区别在哪里，在目，我哪个框架？

ASSISTANT

Next.js 和 Nuxt.js 都是秀的服务器端渲染（SSR）框架，分别基于 React 和 Vue.js。它有很多相似之，但也有一些关的区别。以下是它的主要区别和建：

主要区别：

1. 底框架：

- Next.js 基于 React
- Nuxt.js 基于 Vue.js

2. 学曲：

- Next.js 的学曲可能稍陡，因 React 本身就相对复
- Nuxt.js 通常被更容易上手，因 Vue.js 的理念是易学

3. 性能：

- Next.js 在大型用中可能有微的性能
- Nuxt.js 在小到中型目中表异

4. 社区和生系：

- Next.js 有更大的社区和更丰富的生系
- Nuxt.js 的社区也在迅速增长，但模相对小

5. 静站点生成（SSG）：

- Next.js 的静站点生成功能更强大和灵活
- Nuxt.js 也支持静站点生成，但功能相对

6. 配置和定：

- Nuxt.js 更注重定于配置，提供了更多的默置
- Next.js 提供了更多的灵活性，但可能需要更多的手配置

7. 文件系路由：

- 两者都支持基于文件系的路由
- Nuxt.js 的路由系更加直和

8. TypeScript 支持：

- Next.js 对 TypeScript 的支持更加原生和完善
- Nuxt.js 也支持 TypeScript，但集成度不如 Next.js

建：

1. 如果你更熟悉 React 或者目擅长 React， Next.js
2. 如果你喜 Vue.js 的洁性或者更熟悉 Vue， Nuxt.js
3. 对于大型、复的企用，Next.js 可能是更好的
4. 对于中小型目或快速原型开，Nuxt.js 可能更合适
5. 如果目需要高度的定制化和灵活性，Next.js 可能更适合
6. 如果你希望有更多的默配置和定，Nuxt.js 可能是更好的
7. 如果目heavily依TypeScript，Next.js可能更合适

最，哪个框架很大程度上取决于你的具体需求、技能和个人偏好。两者都是excellent的框架，能够帮助你构建高量的web用。建你可以两者的小型demo目，看看哪个更符合你的开风格和目需求。

USER

那么，将才的Todo目，用Bun、Elysia、Next.js重写一下

ASSISTANT

好的,我可以使⤵用Bun、Elysia和Next.js重写之前的Todo目。个合将提供一个高性能的全解决方案。以下是步和代:

1. 首先,建一个新的Next.js目:

```
```bash
bun create next-app todo-elysia
cd todo-elysia
```
```

2. 安装必要的依:

```
```bash
bun add elysia @elysiajs/html marked
```
```

3. 建Elysia服器文件 `server.ts`:

```
```typescript
import { Elysia, t } from 'elysia';
import { html } from '@elysiajs/html';
import { marked } from 'marked';

let todos = [
 { id: 1, text: "Learn **Elysia**", completed: false },
 { id: 2, text: "Build a _Todo_ app", completed: false },
];

const app = new Elysia()
 .use(html())
 .get('/', () => 'Hello Elysia')
 .get('/api/todos', () => todos)
 .post('/api/todos', ({ body }) => {
 const newTodo = {
 id: todos.length + 1,
 text: body.text,
 completed: false
 };
 todos.push(newTodo);
 return newTodo;
 });
```
```

```

}, {
  body: t.Object({
    text: t.String()
  })
})
.put('/api/todos/:id', ({ params, body }) => {
  const todo = todos.find(t => t.id === parseInt(params.id));
  if (todo) {
    todo.text = body.text;
    todo.completed = body.completed;
  }
  return todo;
}, {
  params: t.Object({
    id: t.String()
  }),
  body: t.Object({
    text: t.String(),
    completed: t.Boolean()
  })
})
.delete('/api/todos/:id', ({ params }) => {
  todos = todos.filter(t => t.id !== parseInt(params.id));
  return { success: true };
}, {
  params: t.Object({
    id: t.String()
  })
})
.get('/api/todos/render/:id', ({ params }) => {
  const todo = todos.find(t => t.id === parseInt(params.id));
  if (todo) {
    return html(`<div>${marked(todo.text)}</div>`);
  }
  return 'Todo not found';
}, {
  params: t.Object({
    id: t.String()
  })
})
.listen(3000);

```

```

console.log(` Elysia is running at ${app.server?.hostname}:${app.server?.port}`);
```

```

4. 更新 `package.json` 的 scripts:

```
``json
"scripts": {
 "dev": "bun run server.ts & next dev",
 "build": "next build",
 "start": "next start",
 "lint": "next lint"
}
``
```

5. 建 `pages/index.tsx`:

```
``tsx
import { useState, useEffect } from 'react';
import { marked } from 'marked';

interface Todo {
 id: number;
 text: string;
 completed: boolean;
}

export default function Home() {
 const [todos, setTodos] = useState<Todo[]>([]);
 const [newTodo, setNewTodo] = useState("");
 const [editingId, setEditingId] = useState<number | null>(null);
 const [editingText, setEditingText] = useState("");

 useEffect(() => {
 fetchTodos();
 }, []);

 const fetchTodos = async () => {
 const response = await fetch('/api/todos');
 const data = await response.json();
 setTodos(data);
 };

 const addTodo = async (e: React.FormEvent) => {
 e.preventDefault();
 if (newTodo.trim()) {
 const response = await fetch('/api/todos', {
 method: 'POST',
 headers: { 'Content-Type': 'application/json' },

```

```

 body: JSON.stringify({ text: newTodo })
 });
 const data = await response.json();
 setTodos([...todos, data]);
 setNewTodo("");
}
};

const updateTodo = async (id: number, text: string, completed: boolean) => {
 const response = await fetch(`/api/todos/${id}`, {
 method: 'PUT',
 headers: { 'Content-Type': 'application/json' },
 body: JSON.stringify({ text, completed })
 });
 const updatedTodo = await response.json();
 setTodos(todos.map(todo => todo.id === id ? updatedTodo : todo));
};

const deleteTodo = async (id: number) => {
 await fetch(`/api/todos/${id}`, { method: 'DELETE' });
 setTodos(todos.filter(todo => todo.id !== id));
};

const startEditing = (todo: Todo) => {
 setEditingId(todo.id);
 setEditingText(todo.text);
};

const saveEdit = async () => {
 if (editingId !== null) {
 await updateTodo(editingId, editingText, todos.find(t => t.id === editingId)?.completed || false);
 setEditingId(null);
 setEditingText("");
 }
};

const renderTodoText = async (id: number) => {
 const response = await fetch(`/api/todos/render/${id}`);
 const html = await response.text();
 return { __html: html };
};

return (
 <div className="p-4 max-w-2xl mx-auto">
 <h1 className="text-3xl font-bold mb-4">Todo App with Markdown</h1>

```

```

<form onSubmit={addTodo} className="mb-4">
 <input
 value={newTodo}
 onChange={(e) => setNewTodo(e.target.value)}
 placeholder="Add a new todo"
 className="border p-2 mr-2 w-full mb-2"
 />
 <button type="submit" className="bg-blue-500 text-white p-2 rounded
hover:bg-blue-600">Add</button>
</form>

 {todos.map(todo => (
 <li key={todo.id} className="mb-4 bg-gray-100 p-4 rounded">
 {editingId === todo.id ? (
 <div>
 <textarea
 value={editingText}
 onChange={(e) => setEditingText(e.target.value)}
 className="w-full p-2 border rounded mb-2"
 />
 <button onClick={saveEdit} className="bg-green-500 text-white p-2 rounded
mr-2">Save</button>
 <button onClick={() => setEditingId(null)} className="bg-gray-500 text-white p-2
rounded">Cancel</button>
 </div>
) : (
 <div>
 <div className="flex items-center mb-2">
 <input
 type="checkbox"
 checked={todo.completed}
 onChange={(e) => updateTodo(todo.id, todo.text, e.target.checked)}
 className="mr-2"
 />
 {todo.text}
 <button onClick={() => startEditing(todo)} className="ml-auto text-blue-500 mr-2">Edit</button>
 <button onClick={() => deleteTodo(todo.id)} className="text-red-500">Remove</button>
 </div>
 <div dangerouslySetInnerHTML={renderTodoText(todo.id)} className="markdown-content
text-sm text-gray-600" />
 </div>
)}

)}


```

```
 </div>
);
}
...

```

6. 添加一些基本式,建 `styles/globals.css`:

```
``css
@import 'tailwindcss/base';
@import 'tailwindcss/components';
@import 'tailwindcss/utilities';

.markdown-content {
 @apply prose prose-sm;
}
...

```

7. 更新 `pages/\_app.tsx`:

```
``tsx
import '../styles/globals.css'
import type { AppProps } from 'next/app'

function MyApp({ Component, pageProps }: AppProps) {
 return <Component {...pageProps} />
}

export default MyApp
...

```

8. 配置 Tailwind CSS,建 `tailwind.config.js`:

```
``javascript
module.exports = {
 content: [
 './pages/**/*.js,ts,jsx,tsx',
 './components/**/*.js,ts,jsx,tsx',
],
 theme: {
 extend: {},
 },
 plugins: [
 require('@tailwindcss/typography'),
],
}

```

```

9. 安装 Tailwind CSS 和插件:

```
```bash
bun add -d tailwindcss postcss autoprefixer @tailwindcss/typography
```
```

个使用了 Bun 作运行,Elysia 作后端 API 服务器,Next.js 作前端框架。它保留了之前的 Todo 功能,包括 Markdown 支持和功能。

要运行个项目,使用以下命令:

```
```bash
bun run dev
```
```

将同时启动 Elysia 服务器和 Next.js 服务器。你可以在浏览器中 `http://localhost:3000` 来看和使用个 Todo 应用。

个项目提供了一个强大的全解决方案,结合了 Bun 的高性能,Elysia 的轻量级 API 服务,以及 Next.js 的强大前端功能。它适合构建现代的、高性能的 web 应用。

USER

我想使用UnoCSS，不想使用Tailwindcss，改写

ASSISTANT

当然可以使用 UnoCSS 替代 Tailwind CSS。我需要做一些调整来集成 UnoCSS 到我的 Next.js 项目中。以下是修改后的步骤和代码：

1. 首先，移除 Tailwind CSS 相关的依赖，并安装 UnoCSS：

```
```bash
bun remove tailwindcss postcss autoprefixer @tailwindcss/typography
bun add -d unocss @unocss/webpack
```
```

2. 在根目录创建 `uno.config.ts` 文件：

```
```typescript
import { defineConfig, presetUno, presetTypography } from 'unocss'

export default defineConfig({
 presets: [
 presetUno(),
],
})
```
```

```
    presetTypography(),  
  ],  
})  
...
```

3. 更新 `next.config.js` :

```
````javascript  
const UnoCSS = require('@unocss/webpack').default

/** @type {import('next').NextConfig} */
const nextConfig = {
 reactStrictMode: true,
 webpack: (config) => {
 config.plugins.push(UnoCSS())
 return config
 },
}

module.exports = nextConfig
...
```

### 4. 更新 `pages/\_app.tsx` :

```
````tsx  
import 'uno.css'  
import type { AppProps } from 'next/app'  
  
function MyApp({ Component, pageProps }: AppProps) {  
  return <Component {...pageProps} />  
}  
  
export default MyApp  
...
```

5. 除 `tailwind.config.js` 文件。

6. 更新 `styles/globals.css` :

```
````css  
/* 你可以保留一些全局式，但不再需要入 Tailwind */
...
```

### 7. 更新 `pages/index.tsx`，将 Tailwind 类替 UnoCSS 类：

```
``tsx
```

```
import { useState, useEffect } from 'react';
```

```
import { marked } from 'marked';
```

```
interface Todo {
```

```
 id: number;
```

```
 text: string;
```

```
 completed: boolean;
```

```
}
```

```
export default function Home() {
```

```
 const [todos, setTodos] = useState<Todo[]>([]);
```

```
 const [newTodo, setNewTodo] = useState("");
```

```
 const [editingId, setEditingId] = useState<number | null>(null);
```

```
 const [editingText, setEditingText] = useState("");
```

```
 useEffect(() => {
```

```
 fetchTodos();
```

```
 }, []);
```

```
 const fetchTodos = async () => {
```

```
 const response = await fetch('/api/todos');
```

```
 const data = await response.json();
```

```
 setTodos(data);
```

```
 };
```

```
 const addTodo = async (e: React.FormEvent) => {
```

```
 e.preventDefault();
```

```
 if (newTodo.trim()) {
```

```
 const response = await fetch('/api/todos', {
```

```
 method: 'POST',
```

```
 headers: { 'Content-Type': 'application/json' },
```

```
 body: JSON.stringify({ text: newTodo })
```

```
 });
```

```
 const data = await response.json();
```

```
 setTodos([...todos, data]);
```

```
 setNewTodo("");
```

```
 }
```

```
 };
```

```
 const updateTodo = async (id: number, text: string, completed: boolean) => {
```

```
 const response = await fetch(`/api/todos/${id}`, {
```

```
 method: 'PUT',
```

```
 headers: { 'Content-Type': 'application/json' },
```

```
 body: JSON.stringify({ text, completed })
```

```

});
const updatedTodo = await response.json();
setTodos(todos.map(todo => todo.id === id ? updatedTodo : todo));
};

const deleteTodo = async (id: number) => {
 await fetch(`/api/todos/${id}`, { method: 'DELETE' });
 setTodos(todos.filter(todo => todo.id !== id));
};

const startEditing = (todo: Todo) => {
 setEditingId(todo.id);
 setEditingText(todo.text);
};

const saveEdit = async () => {
 if (editingId !== null) {
 await updateTodo(editingId, editingText, todos.find(t => t.id === editingId)?.completed || false);
 setEditingId(null);
 setEditingText("");
 }
};

const renderTodoText = async (id: number) => {
 const response = await fetch(`/api/todos/render/${id}`);
 const html = await response.text();
 return { __html: html };
};

return (
 <div className="p-4 max-w-2xl mx-auto">
 <h1 className="text-3xl font-bold mb-4">Todo App with Markdown</h1>
 <form onSubmit={addTodo} className="mb-4">
 <input
 value={newTodo}
 onChange={(e) => setNewTodo(e.target.value)}
 placeholder="Add a new todo"
 className="border p-2 mr-2 w-full mb-2"
 />
 <button type="submit" className="bg-blue-500 text-white p-2 rounded
hover:bg-blue-600">Add</button>
 </form>

 {todos.map(todo => (
 <li key={todo.id} className="mb-4 bg-gray-100 p-4 rounded">

```

```

 {editingId === todo.id ? (
 <div>
 <textarea
 value={editingText}
 onChange={(e) => setEditingText(e.target.value)}
 className="w-full p-2 border rounded mb-2"
 />
 <button onClick={saveEdit} className="bg-green-500 text-white p-2 rounded
mr-2">Save</button>
 <button onClick={() => setEditingId(null)} className="bg-gray-500 text-white p-2
rounded">Cancel</button>
 </div>
) : (
 <div>
 <div className="flex items-center mb-2">
 <input
 type="checkbox"
 checked={todo.completed}
 onChange={(e) => updateTodo(todo.id, todo.text, e.target.checked)}
 className="mr-2"
 />
 {todo.text}
 <button onClick={() => startEditing(todo)} className="ml-auto text-blue-500 mr-2">Edit</button>
 <button onClick={() => deleteTodo(todo.id)} className="text-red-500">Remove</button>
 </div>
 <div dangerouslySetInnerHTML={renderTodoText(todo.id)} className="prose prose-sm text-sm
text-gray-600" />
 </div>
)}

)}

</div>
);
}
...

```

注意：UnoCSS 和 Tailwind CSS 的类名在大多数情况下是相同的，所以我不需要对 HTML 结构做太多更改。主要的区别在于配置和集成方式。

8. 如果你想使用 UnoCSS 的 attributify 模式，你可以在 `uno.config.ts` 中添加相的：

```

``typescript
import { defineConfig, presetUno, presetAttributify, presetTypography } from 'unocss'

```

```
export default defineConfig({
 presets: [
 presetUno(),
 presetAttributify(),
 presetTypography(),
],
})
...

```

然后你可以在 JSX 中使用属性模式，例如：

```
```jsx
<div bg="blue-500" text="white" p="2" rounded>
  This is a button
</div>
...

```

些更改将使你的目从 Tailwind CSS 迁移到 UnoCSS。UnoCSS 提供了类似的功能，但具有更高的性能和更大的灵活性。住要根据 UnoCSS 的文档整任何自定配置或使用方式。

USER

对Todo的目加上分类的概念，同支持数据存，数据Turso，ORMDrizzle

ASSISTANT

好的,我可以对Todo目添加分类功能,并使用Turso数据和Drizzle ORM来数据持久化。以下是步:

1. 首先,安装必要的依:

```
```bash
bun add @libsql/client drizzle-orm @types/node
bun add -d drizzle-kit
...

```

2. 建数据schema文件 `db/schema.ts`:

```
```typescript
import { sqliteTable, text, integer } from 'drizzle-orm/sqlite-core';

export const categories = sqliteTable('categories', {
  id: integer('id').primaryKey(),
  name: text('name').notNull(),
});

export const todos = sqliteTable('todos', {

```

```

id: integer('id').primaryKey(),
text: text('text').notNull(),
completed: integer('completed', { mode: 'boolean' }).notNull().default(false),
categoryId: integer('category_id').references(() => categories.id),
});
```

```

### 3. 建数据配置文件 `db/index.ts`:

```

``typescript
import { createClient } from '@libsql/client';
import { drizzle } from 'drizzle-orm/libsql';
import * as schema from './schema';

const client = createClient({
 url: process.env.DATABASE_URL as string,
 authToken: process.env.DATABASE_AUTH_TOKEN,
});

export const db = drizzle(client, { schema });
```

```

4. 更新 `server.ts` 文件以使用数据:

```

``typescript
import { Elysia, t } from 'elysia';
import { html } from '@elysiajs/html';
import { marked } from 'marked';
import { db } from './db';
import { todos, categories } from './db/schema';
import { eq } from 'drizzle-orm';

const app = new Elysia()
  .use(html())
  .get('/', () => 'Hello Elysia')

// Categories API
.app.get('/api/categories', async () => {
  return await db.select().from(categories);
})
.app.post('/api/categories', async ({ body }) => {
  const [newCategory] = await db.insert(categories).values(body).returning();
  return newCategory;
}, {
  body: t.Object({

```

```

    name: t.String()
  })
})

// Todos API
.get('/api/todos', async () => {
  return await db.select().from(todos).leftJoin(categories, eq(todos.categoryId, categories.id));
})
.post('/api/todos', async ({ body }) => {
  const [newTodo] = await db.insert(todos).values(body).returning();
  return newTodo;
}, {
  body: t.Object({
    text: t.String(),
    categoryId: t.Optional(t.Number())
  })
})
.put('/api/todos/:id', async ({ params, body }) => {
  const [updatedTodo] = await db.update(todos)
    .set(body)
    .where(eq(todos.id, parseInt(params.id)))
    .returning();
  return updatedTodo;
}, {
  params: t.Object({
    id: t.String()
  }),
  body: t.Object({
    text: t.Optional(t.String()),
    completed: t.Optional(t.Boolean()),
    categoryId: t.Optional(t.Number())
  })
})
.delete('/api/todos/:id', async ({ params }) => {
  await db.delete(todos).where(eq(todos.id, parseInt(params.id)));
  return { success: true };
}, {
  params: t.Object({
    id: t.String()
  })
})
.get('/api/todos/render/:id', async ({ params }) => {
  const [todo] = await db.select().from(todos).where(eq(todos.id, parseInt(params.id)));
  if (todo) {
    return html(`<div>${marked(todo.text)}</div>`);
  }
});

```

```

    }
    return 'Todo not found';
  }, {
    params: t.Object({
      id: t.String()
    })
  })
})
.listen(3000);

console.log(` Elysia is running at ${app.server?.hostname}:${app.server?.port}`);
...

```

5. 更新 `pages/index.tsx` 以支持分类:

```

``tsx
import { useState, useEffect } from 'react';
import { marked } from 'marked';

interface Category {
  id: number;
  name: string;
}

interface Todo {
  id: number;
  text: string;
  completed: boolean;
  categoryId: number | null;
  category?: Category;
}

export default function Home() {
  const [todos, setTodos] = useState<Todo[]>([]);
  const [categories, setCategories] = useState<Category[]>([]);
  const [newTodo, setNewTodo] = useState('');
  const [newCategory, setNewCategory] = useState('');
  const [selectedCategory, setSelectedCategory] = useState<number | null>(null);
  const [editingId, setEditingId] = useState<number | null>(null);
  const [editingText, setEditingText] = useState('');

  useEffect(() => {
    fetchTodos();
    fetchCategories();
  }, []);

```

```
const fetchTodos = async () => {  
  const response = await fetch('/api/todos');  
  const data = await response.json();  
  setTodos(data);  
};
```

```
const fetchCategories = async () => {  
  const response = await fetch('/api/categories');  
  const data = await response.json();  
  setCategories(data);  
};
```

```
const addTodo = async (e: React.FormEvent) => {  
  e.preventDefault();  
  if (newTodo.trim()) {  
    const response = await fetch('/api/todos', {  
      method: 'POST',  
      headers: { 'Content-Type': 'application/json' },  
      body: JSON.stringify({ text: newTodo, categoryId: selectedCategory })  
    });  
    const data = await response.json();  
    setTodos([...todos, data]);  
    setNewTodo("");  
  }  
};
```

```
const addCategory = async (e: React.FormEvent) => {  
  e.preventDefault();  
  if (newCategory.trim()) {  
    const response = await fetch('/api/categories', {  
      method: 'POST',  
      headers: { 'Content-Type': 'application/json' },  
      body: JSON.stringify({ name: newCategory })  
    });  
    const data = await response.json();  
    setCategories([...categories, data]);  
    setNewCategory("");  
  }  
};
```

```
const updateTodo = async (id: number, text: string, completed: boolean, categoryId: number | null) => {  
  const response = await fetch(`/api/todos/${id}`, {  
    method: 'PUT',  
    headers: { 'Content-Type': 'application/json' },  
    body: JSON.stringify({ text, completed, categoryId })  
  });
```

```

});
const updatedTodo = await response.json();
setTodos(todos.map(todo => todo.id === id ? updatedTodo : todo));
};

const deleteTodo = async (id: number) => {
  await fetch(`/api/todos/${id}`, { method: 'DELETE' });
  setTodos(todos.filter(todo => todo.id !== id));
};

const startEditing = (todo: Todo) => {
  setEditingId(todo.id);
  setEditingText(todo.text);
  setSelectedCategory(todo.categoryId);
};

const saveEdit = async () => {
  if (editingId !== null) {
    await updateTodo(editingId, editingText, todos.find(t => t.id === editingId)?.completed || false,
selectedCategory);
    setEditingId(null);
    setEditingText("");
    setSelectedCategory(null);
  }
};

const renderTodoText = async (id: number) => {
  const response = await fetch(`/api/todos/render/${id}`);
  const html = await response.text();
  return { __html: html };
};

return (
  <div className="p-4 max-w-2xl mx-auto">
    <h1 className="text-3xl font-bold mb-4">Todo App with Categories</h1>

    {/* Add Category Form */}
    <form onSubmit={addCategory} className="mb-4">
      <input
        value={newCategory}
        onChange={(e) => setNewCategory(e.target.value)}
        placeholder="Add a new category"
        className="border p-2 mr-2"
      />
      <button type="submit" className="bg-green-500 text-white p-2 rounded hover:bg-green-600">Add

```

Category</button>

</form>

{/* Add Todo Form */}

<form onSubmit={addTodo} className="mb-4">

<input

value={newTodo}

onChange={(e) => setNewTodo(e.target.value)}

placeholder="Add a new todo"

className="border p-2 mr-2 w-full mb-2"

/>

<select

value={selectedCategory || ""}

onChange={(e) => setSelectedCategory(Number(e.target.value) || null)}

className="border p-2 mr-2"

>

<option value="">No Category</option>

{categories.map(category => (

<option key={category.id} value={category.id}>{category.name}</option>

)))

</select>

<button type="submit" className="bg-blue-500 text-white p-2 rounded hover:bg-blue-600">Add

Todo</button>

</form>

{/* Todo List */}

{todos.map(todo => (

<li key={todo.id} className="mb-4 bg-gray-100 p-4 rounded">

{editingId === todo.id ? (

<div>

<textarea

value={editingText}

onChange={(e) => setEditingText(e.target.value)}

className="w-full p-2 border rounded mb-2"

/>

<select

value={selectedCategory || ""}

onChange={(e) => setSelectedCategory(Number(e.target.value) || null)}

className="border p-2 mr-2"

>

<option value="">No Category</option>

{categories.map(category => (

<option key={category.id} value={category.id}>{category.name}</option>

)))

```

        </select>
        <button onClick={saveEdit} className="bg-green-500 text-white p-2 rounded
mr-2">Save</button>
        <button onClick={() => setEditingId(null)} className="bg-gray-500 text-white p-2
rounded">Cancel</button>
    </div>
  ): (
    <div>
      <div className="flex items-center mb-2">
        <input
          type="checkbox"
          checked={todo.completed}
          onChange={(e) => updateTodo(todo.id, todo.text, e.target.checked, todo.categoryId)}
          className="mr-2"
        />
        <span className={todo.completed ? 'line-through' : ''}>{todo.text}</span>
        <span className="ml-2 text-sm text-gray-500">{todo.category?.name}</span>
        <button onClick={() => startEditing(todo)} className="ml-auto text-blue-500 mr-2">Edit</button>
        <button onClick={() => deleteTodo(todo.id)} className="text-red-500">Remove</button>
      </div>
      <div dangerouslySetInnerHTML={renderTodoText(todo.id)} className="prose prose-sm text-sm
text-gray-600" />
    </div>
  )}
</li>
)}}
</ul>
</div>
);
}
'''

```

6. 建一个`.env`文件来存数据接信息:

```

'''
DATABASE_URL=libsql://your-database-url
DATABASE_AUTH_TOKEN=your-auth-token
'''

```

确保将些替你的 Turso 数据 URL 和令牌。

7. 更新`package.json`中的 scripts 以包含数据迁移:

```

```json
"scripts": {

```

```
"dev": "bun run server.ts & next dev",
"build": "next build",
"start": "next start",
"lint": "next lint",
"db:generate": "drizzle-kit generate:sqlite",
"db:push": "drizzle-kit push:sqlite"
}
```
```

8. 运行数据迁移:

```
```bash
bun run db:generate
bun run db:push
```
```

个更新后的 Todo 用在支持分类功能,并使用 Turso 数据行数据存。用可以建新的分类,将 Todo 分配到特定分类,以及按分类 Todo 。

要运行个目,使用以下命令:

```
```bash
bun run dev
```
```

将后 Elysia 服务器和 Next.js 开服务器。你可以在器中 `http://localhost:3000` 来使用个增强版的 Todo 用。

注意,个假如你已置了 Turso 数据。如果没有,你需要建一个 Turso ,建一个新的数据,并取接 URL 和令牌。

USER

我想加入用功能,支持用Google、Github直接登入功能,并保存登用的信息

ASSISTANT

了添加用功能,支持 Google 和 GitHub 登,并保存登用的信息,我可以使用 NextAuth.js。是一个 Next.js 的解决方案,可以轻松集成多种提供者。以下是步:

1. 安装必要的依:

```
```bash
bun add next-auth @auth/drizzle-adapter
```
```

2. 更新数据 schema (`db/schema.ts`) 以包含用相关表:

```
``typescript
```

```
import { sqliteTable, text, integer } from 'drizzle-orm/sqlite-core';
import { relations } from 'drizzle-orm';
```

```
export const users = sqliteTable('users', {
  id: text('id').primaryKey(),
  name: text('name'),
  email: text('email').notNull(),
  emailVerified: integer('emailVerified', { mode: 'timestamp_ms' }),
  image: text('image'),
});
```

```
export const accounts = sqliteTable('accounts', {
  id: text('id').primaryKey(),
  userId: text('userId').notNull().references(() => users.id, { onDelete: 'cascade' }),
  type: text('type').notNull(),
  provider: text('provider').notNull(),
  providerAccountId: text('providerAccountId').notNull(),
  refresh_token: text('refresh_token'),
  access_token: text('access_token'),
  expires_at: integer('expires_at'),
  token_type: text('token_type'),
  scope: text('scope'),
  id_token: text('id_token'),
  session_state: text('session_state'),
});
```

```
export const sessions = sqliteTable('sessions', {
  id: text('id').primaryKey(),
  sessionToken: text('sessionToken').notNull(),
  userId: text('userId').notNull().references(() => users.id, { onDelete: 'cascade' }),
  expires: integer('expires', { mode: 'timestamp_ms' }).notNull(),
});
```

```
export const verificationTokens = sqliteTable('verificationToken', {
  identifier: text('identifier').notNull(),
  token: text('token').notNull(),
  expires: integer('expires', { mode: 'timestamp_ms' }).notNull(),
},
(vt) => ({
  compoundKey: [vt.identifier, vt.token],
}));
```

```
export const categories = sqliteTable('categories', {
```

```

    id: integer('id').primaryKey(),
    name: text('name').notNull(),
    userId: text('userId').notNull().references(() => users.id),
  });

export const todos = sqliteTable('todos', {
  id: integer('id').primaryKey(),
  text: text('text').notNull(),
  completed: integer('completed', { mode: 'boolean' }).notNull().default(false),
  categoryId: integer('category_id').references(() => categories.id),
  userId: text('userId').notNull().references(() => users.id),
});

export const usersRelations = relations(users, ({ many }) => ({
  todos: many(todos),
  categories: many(categories),
}));

export const categoriesRelations = relations(categories, ({ one }) => ({
  user: one(users, { fields: [categories.userId], references: [users.id] }),
}));

export const todosRelations = relations(todos, ({ one }) => ({
  category: one(categories, { fields: [todos.categoryId], references: [categories.id] }),
  user: one(users, { fields: [todos.userId], references: [users.id] }),
}));

```

3. 建 NextAuth 配置文件 `pages/api/auth/[...nextauth].ts` :

```

``typescript
import NextAuth from "next-auth"
import GoogleProvider from "next-auth/providers/google"
import GithubProvider from "next-auth/providers/github"
import { DrizzleAdapter } from "@auth/drizzle-adapter"
import { db } from "../../db"

export default NextAuth({
  adapter: DrizzleAdapter(db),
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID!,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET!,
    }),
    GithubProvider({

```

```

    clientId: process.env.GITHUB_ID!,
    clientSecret: process.env.GITHUB_SECRET!,
  )),
],
callbacks: {
  session: async ({ session, user }) => {
    if (session?.user) {
      session.user.id = user.id;
    }
    return session;
  },
},
pages: {
  signIn: '/auth/signin',
},
})
...

```

4. 建一个自定义登录页面 `pages/auth/signin.tsx` :

```

````tsx
import { getProviders, signIn } from "next-auth/react"
import { GetServerSideProps } from "next"

export default function SignIn({ providers }: { providers: any }) {
 return (
 <div className="flex flex-col items-center justify-center min-h-screen py-2">
 <h1 className="text-4xl font-bold mb-6">Sign In</h1>
 {Object.values(providers).map((provider: any) => (
 <div key={provider.name} className="mb-4">
 <button
 onClick={() => signIn(provider.id)}
 className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded"
 >
 Sign in with {provider.name}
 </button>
 </div>
))}
 </div>
)
}

export const getServerSideProps: GetServerSideProps = async (context) => {
 const providers = await getProviders()
 return {

```

```
 props: { providers },
 }
}
...

```

5. 更新 `pages/\_app.tsx` 以包含 SessionProvider :

```
``tsx
import { SessionProvider } from "next-auth/react"
import type { AppProps } from 'next/app'
import 'uno.css'

function MyApp({ Component, pageProps: { session, ...pageProps } }: AppProps) {
 return (
 <SessionProvider session={session}>
 <Component {...pageProps} />
 </SessionProvider>
)
}

export default MyApp
...

```

6. 更新 `pages/index.tsx` 以使用用 :

```
``tsx
import { useState, useEffect } from 'react';
import { useSession, signIn, signOut } from "next-auth/react"
import { marked } from 'marked';

// ... (保留之前的 interface 定)

export default function Home() {
 const { data: session } = useSession()
 // ... (保留之前的 state 定)

 useEffect(() => {
 if (session) {
 fetchTodos();
 fetchCategories();
 }
 }, [session]);

 // ... (保留之前的函数定，但需要修改 API 用以包含用 ID)

```

```

const addTodo = async (e: React.FormEvent) => {
 e.preventDefault();
 if (newTodo.trim() && session?.user?.id) {
 const response = await fetch('/api/todos', {
 method: 'POST',
 headers: { 'Content-Type': 'application/json' },
 body: JSON.stringify({ text: newTodo, categoryId: selectedCategory, userId: session.user.id })
 });
 const data = await response.json();
 setTodos([...todos, data]);
 setNewTodo("");
 }
};

// ... (类似地修改其他 API 用函数)

if (!session) {
 return (
 <div className="flex flex-col items-center justify-center min-h-screen py-2">
 <h1 className="text-4xl font-bold mb-6">Welcome to Todo App</h1>
 <button onClick={() => signIn()} className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2
px-4 rounded">
 Sign in
 </button>
 </div>
)
}

return (
 <div className="p-4 max-w-2xl mx-auto">
 <div className="flex justify-between items-center mb-4">
 <h1 className="text-3xl font-bold">Todo App with Categories</h1>
 <button onClick={() => signOut()} className="bg-red-500 text-white p-2 rounded hover:bg-red-600">
 Sign out
 </button>
 </div>

 {/* ... (保留之前的 JSX, 包括表和列表) */}
 </div>
);
}
...

```

7. 更新 `server.ts` 文件中的 API 路由以包含用 ID :

```

``typescript
// ... (previous imports)
import { getSession } from "next-auth/react"

const app = new Elysia()
 .use(html())
 // ... (previous routes)

 .post('/api/todos', async ({ body, request }) => {
 const session = await getSession({ req: request });
 if (!session?.user?.id) {
 return new Response('Unauthorized', { status: 401 });
 }
 const [newTodo] = await db.insert(todos).values({ ...body, userId: session.user.id }).returning();
 return newTodo;
 }, {
 body: t.Object({
 text: t.String(),
 categoryId: t.Optional(t.Number())
 })
 })

 // ... (similarly update other routes to check for authentication and include userId)

 .listen(3000);
...

```

8. 更新 `.env` 文件以包含 OAuth 凭：

```

...
DATABASE_URL=libsql://your-database-url
DATABASE_AUTH_TOKEN=your-auth-token
GOOGLE_CLIENT_ID=your-google-client-id
GOOGLE_CLIENT_SECRET=your-google-client-secret
GITHUB_ID=your-github-client-id
GITHUB_SECRET=your-github-client-secret
NEXTAUTH_URL=http://localhost:3000
NEXTAUTH_SECRET=your-nextauth-secret
...

```

确保在 Google Cloud Console 和 GitHub Developer Settings 中建 OAuth 用并取相的客端 ID 和密。

9. 运行数据迁移以建新的用相关表：

```

``bash

```

```
bun run db:generate
bun run db:push
```
```

个更新后的 Todo 用支持用，允用使用 Google 或 GitHub 登。它将用信息与 Todo 目和分类关起来，确保每个用只能看到和管理自己的数据。

要运行个目，使用以下命令：

```
```bash
bun run dev
```
```

将启 Elysia 服务器和 Next.js 开服务器。你可以在器中 `http://localhost:3000` 来使用个有用的 Todo 用。

注意，你需要正确配置 OAuth 提供者（Google 和 GitHub）并在 `.env` 文件中置相的境量。此外，确保你的 Turso 数据已更新以包含新的用相关表。