Plugin Vue

Introduced in Tiki21, this wiki plugin allows you to add a Vue.js
Use this plugin to integrate powerful Vue.js features seamlessly into your Tiki pages.

Parameters

Add a Vue.js component
Introduced in Tiki 21.
Go to the source code
Preferences required: wikiplugin_vue

Parameters	Accepted Values	Description	Default	Since
(body of plugin)		HTML		
`name`		Identifier of the component.		21.0
`app`	(blank) y n	Make the base App object and initialise Vue.js		21.0

Notes

- **API Support**: You can now write Vue.js components using either the Composition API (recommended for modern development) or the Options API (ideal for those migrating older components or new to Vue.js).
- **Scoped Styles**: Styles defined in components will not leak into the global CSS scope, making it easier to maintain and debug your projects.
- **Reactive State Management**: Leverage Vue's reactivity system to build highly interactive and dynamic interfaces.

'*Prerequisites*':
The rest of this documentation assumes that you have basic familiarity with Vue.js. If you are completely new to Vue.js, it's recommended to first learn the framework's fundamentals, such as component structure, reactivity, and lifecycle methods, before using this plugin. Prior experience with other frameworks is not required, but having a good understanding of Vue.js concepts will help you make the most of the PluginVue.

Once you've grasped the basics, you'll find it easier to build and integrate Vue.js components into your Tiki site.

Examples

This section provides a walkthrough for building and integrating Vue.js components using the PluginVue.

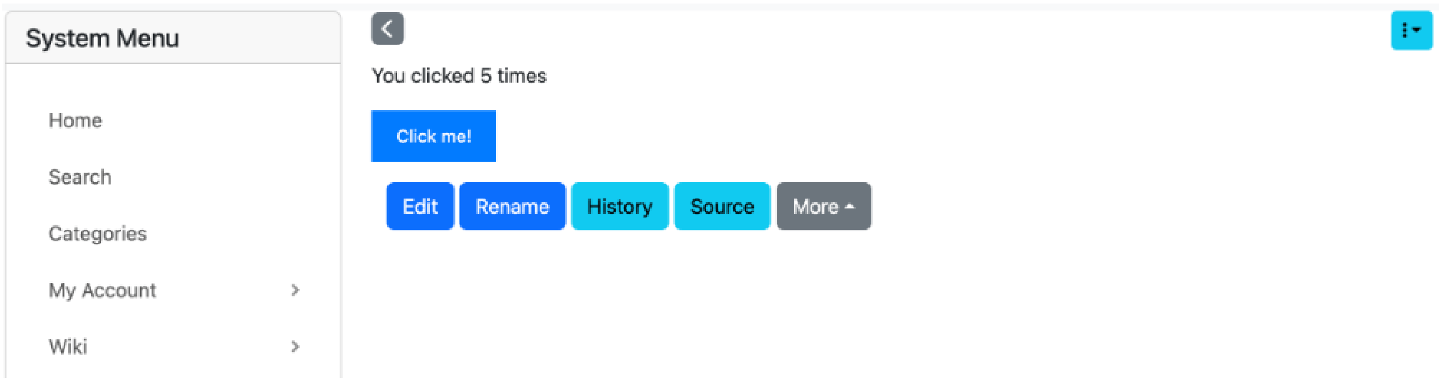
Example 1: Simple Counter Component

A simple example to help you get started with Vue.js 3. This counter demonstrates reactivity by incrementing a number when a button is clicked.

Using the Composition API

```
{VUE(app="y" name="CounterComposition")} <script setup> import { ref } from "vue"; const count = ref(0); const increment = () => { count.value++; }; </script> <template> <div> <p>You clicked {{ count }} times</p> <button @click="increment">Click me!</button> </div> </template> <style scoped> button { background-color: #007bff; color: white; border: none; padding: 10px 20px; font-size: 14px; cursor: pointer; } button:hover { background-color: #0056b3; } </style> {VUE}
```

Would produce:



You clicked 5 times

Click me!

Using the Options API

{VUE(app="y" name="CounterOptions")} <script> export default { data() { return { count: 0 }; }, methods: { increment() { this.count++; } } }; </script> <template> <div> <p>You clicked {{ count }} times</p> <button @click="increment">Click me!</button> </div> </template> <style scoped> button { background-color: #007bff; color: white; border: none; padding: 10px 20px; font-size: 14px; cursor: pointer; } button:hover { background-color: #0056b3; } </style> {VUE}

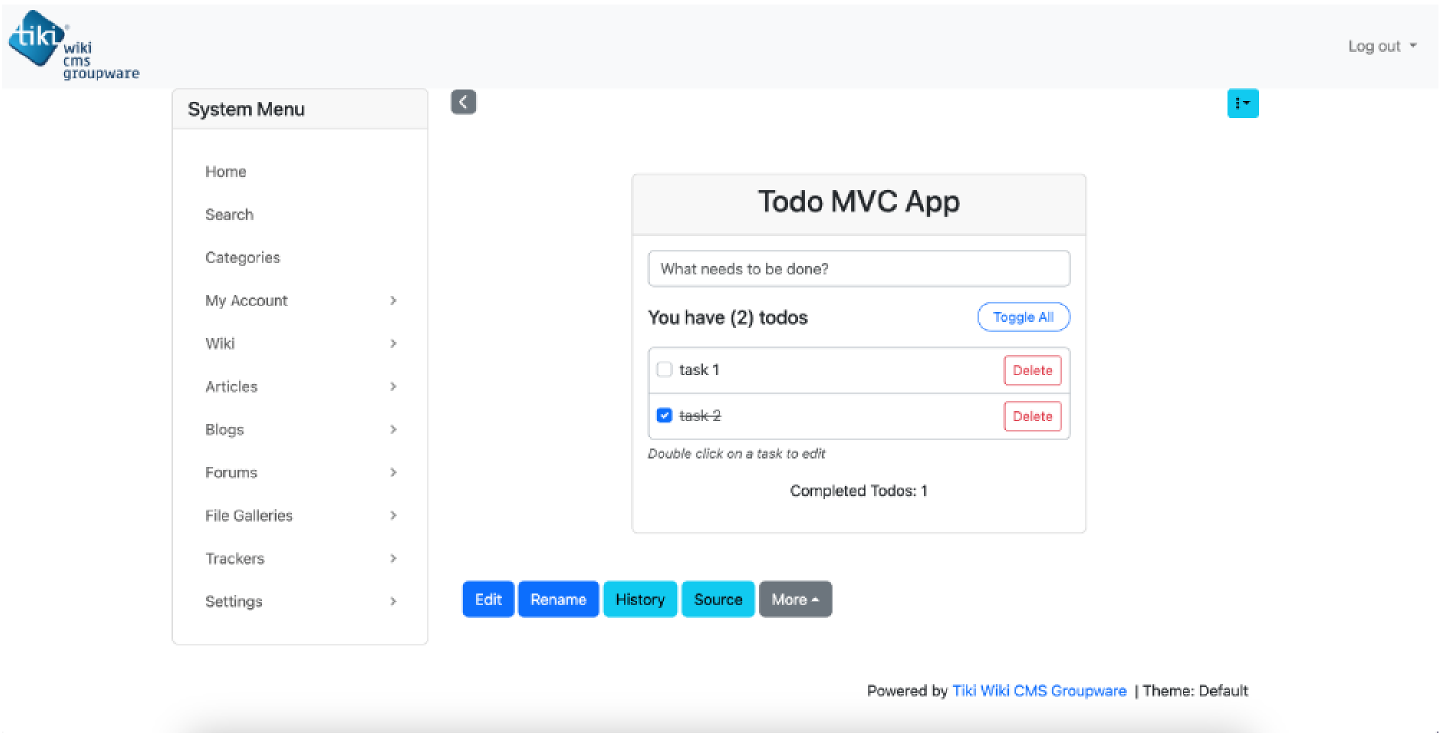
Example 2: TODO App (bootstrap themed)

A more advanced example that demonstrates state management, event handling, and dynamic rendering with Vue.js. This application allows users to add, complete, and delete tasks.

Using the Composition API

{VUE(app="y" name="TodoApp")} <script setup> import { ref, computed } from "vue"; const newTodo = ref(""); const todos = ref([]); const todoCount = computed(() => todos.value.length); const completedCount = computed(() => todos.value.filter((todo) => todo.completed).length); const addTodo = () => { if (newTodo.value.trim() !== "") { todos.value.push({ text: newTodo.value, completed: false }); newTodo.value = ""; } }; const removeTodo = (index) => { todos.value.splice(index, 1); }; const toggleComplete = (todo) => { todo.completed = !todo.completed; }; const toggleAll = () => { const allCompleted = todos.value.every((todo) => todo.completed); todos.value.forEach((todo) => { todo.completed = !allCompleted; }); }; const editTodo = (todo) => { todo.editing = true; }; const finishEdit = (todo) => { if (todo.text.trim() === "") { removeTodo(todos.value.indexOf(todo)); } todo.editing = false; }; const cancelEdit = (todo) => { todo.editing = false; }; </script> <template> <div class="container mt-5 todo-app"> <div class="card"> <div class="card-header text-center"> <h2>Todo MVC App</h2> </div> <div class="card-body"> <div class="mb-3"> <input type="text" v-model="newTodo" @keyup.enter="addTodo" class="form-control" placeholder="What needs to be done?" /> </div> <div v-if="todoCount" class="d-flex justify-content-between align-items-center mb-3"> <h5 class="mb-0">You have ({{ todoCount }}) todos</h5> <button @click="toggleAll" class="btn btn-sm btn-outline-primary rounded-pill px-3">Toggle All</button> </div> <ul class="list-group"> <li v-for="(todo, index) in todos" :key="index" class="list-group-item d-flex justify-content-between align-items-center px-2"> <div v-if="!todo.editing" class="form-check" @dblclick="editTodo(todo)"> <input type="checkbox" class="form-check-input" v-model="todo.completed" /> <label class="form-check-label" :class="{ 'text-decoration-line-through text-muted': todo.completed }"> {{ todo.text }} </label> </div> <input v-else type="text" class="form-control edit-input" v-model="todo.text" @blur="finishEdit(todo)" @keyup.enter="finishEdit(todo)" @keyup.esc="cancelEdit(todo)" /> <button @click="removeTodo(index)" class="btn btn-outline-danger btn-sm">Delete</button> </li> </ul> <div v-if="todoCount > 0" class="form-text fst-italic text-start">Double click on a task to edit</div> <div v-if="todos.length === 0" class="alert alert-success mt-3" role="alert"> No todos left. You're all caught up! </div> <p v-if="completedCount > 0" class="mt-3"> Completed Todos: {{ completedCount }} </p> </div> </div> </div> </template> <style scoped> .todo-app { max-width: 500px; margin: 50px auto; text-align: center; } .completed { text-decoration: line-through; color: grey; } .edit-input { width: 100%; } </style> {VUE}

Would produce:



Resources

- https://vuejs.org/guide/introduction.html