



Fabian Deitelhoff

Vue.js

Von Grundlagen
bis Best Practices



dpunkt.verlag



Inhalt

Cover

Titel

Impressum

Widmung

Inhaltsverzeichnis

Vorwort

1 Vue im Überblick

- 1.1 Was ist ein JavaScript-Frontend-Framework?
- 1.2 Historie und das Team rund um Vue
- 1.3 Was ist Vue und was zeichnet es aus?
- 1.4 Warum Vue?
- 1.5 Unterschiede zu anderen Web-Frameworks
- 1.6 Vue 2 oder Vue 3 lernen?
- 1.7 Der Blick in die Zukunft

2 Die Grundlagen: Ein Vue-Schnellstart

- 2.1 Die erste Vue-Anwendung aufsetzen
 - 2.1.1 Die HTML-Struktur
 - 2.1.2 Einbinden von Vue als Skript
 - 2.1.3 Unsere erste Vue-Instanz
 - 2.1.4 Schleifen und Bedingungen
 - 2.1.5 Benutzereingaben verarbeiten
 - 2.1.6 Mit Komponenten kapseln
 - 2.1.7 Das deklarative Rendering und das virtuelle DOM

3 Vue 3 im Überblick

- 3.1 Warum neu schreiben?

- 3.2 Ein mehrschrittiges Release
- 3.3 Interne und externe Anpassungen
- 3.4 Der neue Standard und Framework-Empfehlungen

4 Vue-Projekte aufsetzen

- 4.1 Scaffolding als Basis
- 4.2 Vue-Projekte erstellen
 - 4.2.1 Das Vue CLI
 - 4.2.2 Die Auswahlmöglichkeiten
 - 4.2.3 Ein Projekt erzeugen und den Webserver starten
 - 4.2.4 Rapid (instant) Prototyping
- 4.3 Vite als neuer Standard
- 4.4 Die Entwicklungsumgebung vorbereiten
- 4.5 Server-Side Rendering (SSR)

5 Das (neue) Reactivity-System

- 5.1 Der Ansatz von Vue
- 5.2 Die Implementierung als Proxy in Vue 3
- 5.3 Probleme bei der Reaktivität

6 Komponenten im Detail

- 6.1 Komponentenorientierte Entwicklung
- 6.2 Komponenten und das Separation-of-Concerns-Prinzip
- 6.3 Options API, Class-Style-Syntax und Composition API
 - 6.3.1 Options API
 - 6.3.2 Composition API
- 6.4 Globale und lokale Komponenten
- 6.5 Single-File Components (SFC)
- 6.6 Lifecycle-Hooks
- 6.7 Daten und Props
- 6.8 Non-Prop-Attribute und Refs

- 6.9 Methoden
- 6.10 Benutzerdefinierte Events
- 6.11 Der Datenfluss zwischen Komponenten
- 6.12 Computed Propertys
- 6.13 Watcher
- 6.14 Eigene Komponenten mit v-model
- 6.15 Standard, Named und Scoped Slots
- 6.16 Dynamische Komponenten
- 6.17 Eigene Direktiven und Plug-ins
- 6.18 Fragmente, Portale und Suspense
- 6.19 Asynchrone und funktionale Komponenten
- 6.20 Dependency Injection mit Provide und Inject
- 6.21 Render-Funktionen und JSX
- 6.22 Vue Components und Web Components
- 6.23 Wiederverwendbarkeit und Komponenten-Patterns

7 Data Binding, Formulare und Styles

- 7.1 Einführung ins Data Binding bei Vue
- 7.2 Template-Syntax und das Binding von Attributen
- 7.3 Bedingtes Rendering mit v-if und v-show
- 7.4 Rendern von Listen mit v-for
- 7.5 Die neue Arbeit mit Arrays
- 7.6 Daten filtern, suchen und sortieren
- 7.7 Daten auf Seiten aufteilen (Pagination)
- 7.8 Formulare mit Daten verknüpfen
- 7.9 Events von Anwenderinnen verarbeiten
- 7.10 Formulare validieren
- 7.11 Class- und Style-Binding
- 7.12 Animationen und Transitionen

8 Das Composition API

- 8.1 Der Hintergrund
- 8.2 Setup, Daten und Props
- 8.3 Script Setup
- 8.4 Methoden und Lifecycle-Hooks
- 8.5 Computed Propertys und Watcher
- 8.6 Composables (Composition Function)
- 8.7 Vorgefertigte Kompositionen in Bibliotheken
- 8.8 ref vs. reactive und Reactivity Transform
- 8.9 Provide und Inject
- 8.10 Render Functions und Templates Refs
- 8.11 Eine komplette Komponente mit dem Composition API
- 8.12 Das Reactivity API
- 8.13 Composition API vs. Options API vs. Class API
- 8.14 Das Composition API und Vue 2

9 Routing mit dem Vue Router

- 9.1 Was ist Routing und was der Vue Router?
- 9.2 Installation und Bestandteile des Vue Router
- 9.3 Routen konfigurieren
- 9.4 Routen nutzen
- 9.5 Dynamische Routen mit Parametern und Props
- 9.6 Das Pattern-Matching
- 9.7 Die verschiedenen History-Modi
- 9.8 Verschachtelte Routen, Redirects und Lazy Loading
- 9.9 Route-Guards und Metadaten
- 9.10 CSS-Klassen, Transitionen und Composition API
- 9.11 Fehler bei der Navigation erkennen
- 9.12 Ein DIY-Router für Minimalistinnen und Minimalisten

10 State Management mit Vuex und Pinia

10.1 Was ist State Management und das Flux-Pattern

10.2 Andere State-Management-Patterns

10.3 Das neue Vuex und die Installation

10.4 Die drei Kernprinzipien

10.5 Die 4 + 1 Kernkonzepte

10.5.1 Der Zustand (State)

10.5.2 Zugriff auf den Zustand (Getter)

10.5.3 Veränderungen des Zustands (Mutations)

10.5.4 Aktionen durchführen (Actions)

10.5.5 Einen Store aufteilen (Module)

10.6 Einen Store in Komponenten nutzen

10.7 Vuex und das Composition API

10.8 Die Struktur einer Vuex-Anwendung und Plug-ins

10.9 Einen Store debuggen

10.10 State Management mit Pinia

10.11 Vuex bzw. Pinia nutzen oder nicht?

11 Mit Vue auf Ressourcen zugreifen

11.1 Das Beispiel-API

11.2 Zugriff auf ein API über das Fetch API

11.3 Axios vs. Fetch

11.4 Die Bibliothek Axios als Quasi-Standard

11.5 Axios in Vue integrieren

11.6 Daten abfragen und anzeigen

11.7 Daten hinzufügen und löschen mit POST und DELETE

11.8 Axios konfigurieren

11.9 Pagination mit API-Aufrufen

11.10 Mit Fehlern umgehen

- 11.11 API-Zugriffe bündeln
- 11.12 Axios mit Vue Router und Vuex
- 11.13 Ein kurzer Blick auf Alternativen

12 Anwendungsentwicklung mit Vue

- 12.1 Ein Blick über den Tellerrand
- 12.2 Das Tooling als Grundvoraussetzung
- 12.3 Eine Vue-Projektstruktur
- 12.4 Vue-Apps debuggen und Fehlerbehandlung
- 12.5 Web-Apps auf Basis von UI-Frameworks
- 12.6 Web-Apps auf Basis von Application Frameworks
- 12.7 Mobile App-Entwicklung und statische Seiten
- 12.8 Eine Web-App am Beispiel von Quasar

13 Internationalisierung und Barrierefreiheit

- 13.1 Installation und Konfiguration von Vue I18n
- 13.2 Das aktive Gebietsschema abrufen und verändern
- 13.3 Übersetzungen nutzen
- 13.4 Vue I18n und das Composition API
- 13.5 Asynchrones Laden von Übersetzungsdateien
- 13.6 Routen und Router-Links übersetzen
- 13.7 Vuex und Vue I18n
- 13.8 Anwendungsentwicklung und das Ökosystem
- 13.9 Etwas zur Barrierefreiheit

14 Testen von Vue-Anwendungen

- 14.1 Das Testen und Vue
- 14.2 Vorbereitungen im Projekt
- 14.3 Tests organisieren und strukturieren
- 14.4 Unit-Tests für JavaScript und TypeScript
- 14.5 Unit-Tests für Vue-Komponenten

14.6 Unit-Tests für das Composition API

14.7 Events testen

14.8 Tests für Formulareingaben

14.9 Tests für Vue Router

14.10 Tests für Vuex

14.11 Ende-zu-Ende-Tests

14.12 Snapshot-Tests

14.13 Tests der Internationalisierung

15 Build und Deployment von Vue-Anwendungen

15.1 Development- vs. Production-Build

15.2 Den Build im Auge behalten

15.3 Deployment auf Netlify und Heroku

15.4 CI-/CD-Pipelines mit GitHub Actions und Azure DevOps

16 Performante Vue-Apps entwickeln

16.1 Vue 3 als allgemeiner Vorteil

16.2 Performance von Komponenten prüfen

16.3 Events und Reactivity

17 Migration von Vue 2 auf Vue 3

17.1 Die Wege einer Migration und der Migration-Build

17.2 Breaking Changes

17.2.1 Globales API

17.2.2 Template-Direktiven

17.2.3 Das Key-Attribut

17.2.4 Komponenten

17.2.5 Render-Funktionen

17.2.6 Viele kleinere Änderungen

17.3 Vue Router und Vuex

18 Vue und TypeScript

- 18.1 Vue (3) und TypeScript
- 18.2 Ein neues Vue-Projekt mit TypeScript
- 18.3 TypeScript im Vue-Projekt
- 18.4 Die verschiedenen API-Modelle
 - 18.4.1 Options API
 - 18.4.2 Class-Style Vue Components
 - 18.4.3 Composition API
- 18.5 Vue Router, Vuex und Internationalisierung
- 18.6 Vor- und Nachteile bei Vue mit TypeScript
- 18.7 Migration auf TypeScript

19 Vue und das Backend

- 19.1 Express (Node.js, JavaScript und TypeScript)
- 19.2 Django (Python)
- 19.3 Laravel (PHP)
- 19.4 Spring Boot (Java)
- 19.5 ASP.NET Core (.NET, C#, F# und Visual Basic)
- 19.6 Firebase (Low Code)
- 19.7 Serverless als Motto
- 19.8 Low- und No-Code als Wachstumsmotor

20 Etwas zu Bibliotheken

- 20.1 Composition API
- 20.2 Awesome Vue als umfassender Überblick
- 20.3 Eine Bibliothek für Vue 2 und Vue 3 erstellen

Literaturverzeichnis

Index

3 Vue 3 im Überblick

In Vue 3 sind zahlreiche Änderungen eingeflossen, die in einem länger andauernden Prozess definiert, der Community im Voraus vorgestellt und mit dieser diskutiert worden sind. Diese Diskussionen hatten nicht immer einen Fokus auf einen sachlichen Diskurs. Die Ankündigungen, erste Request-for-Comments-Dokumente (RFCs) und Beispiele zu den geplanten Änderungen zu Vue 3 haben durchaus kontroverse und heftige Diskussionen ausgelöst. Das betrifft insbesondere die Spezifikation zum Composition API (vorgestellt im Juni 2019). Viele Diskussionen wurden durch Missverständnisse befördert. Zum Beispiel war nicht allen von Anfang an klar, dass das neue Composition API optional ist und das vorhandene Options API nicht ersetzen sollte. Letztendlich zeigen diese Reaktionen deutlich, dass Änderungen an Frameworks aller Art durchaus stressig sind und sich eine Community missverstanden oder angegriffen fühlen kann.

Zahlreiche Neuerungen haben es in die finale Version geschafft, bei anderen wurden verschiedene Implementierungen getestet und wiederum andere Ideen dagegen komplett verworfen. Laut Release-Post (Team V., Release-Post zu v3.0.0 One Piece, 18) auf GitHub hat sich die Performance deutlich verbessert (Team V., Vue 3 vs. Vue 2 Perf comparison, 2022) sowie die Größe der Bibliothek und der Speicherverbrauch verringert.

3.1 Warum neu schreiben?

Eine primäre Frage bei der neuen Vue-Version ist, warum diese zu einem Großteil neu implementiert wurde. Diese Tendenz gibt es bei Softwareprojekten immer wieder, weil sich die Rahmenbedingungen, Anforderungen und technischen Voraussetzungen ändern und die Verlockung groß ist, bei der nächsten Version noch mal von vorne anzufangen.

Das Vue-Team hat sich dennoch dazu entschlossen, die Codebasis von Vue 2 grundlegend zu verändern. Zwei primäre Abwägungen haben zu dieser

Entscheidung geführt: Zunächst die allgemeine Verfügbarkeit vieler neuer JavaScript-Sprach-Features in Browsern, die mittlerweile in der Community umfassend genutzt werden. Darüber hinaus gibt es viele Verbesserungen bei den Design- und Architekturentscheidungen von Vue, die sich über die 2,5 Jahre Entwicklungszeit der Vue-2-Codebasis herauskristallisiert haben und jetzt umgesetzt werden.

Die primären Gründe, Vue 3 neu zu implementieren, sind daher die Nutzung neuer JavaScript-Funktionen, zum Beispiel aus ECMAScript 2015 (ES2015). Am nennenswertesten sind die neuen Proxys, die Vue für das verbesserte Reactivity-System nutzt (siehe Kapitel 5). Zudem wurde die Codebasis von JavaScript auf TypeScript umgestellt, um das Typsystem nutzen zu können. Auf der Architekturseite wurden interne Pakete vom Vue-Kern entkoppelt, um explizite Schnittstellen zu definieren. Das verbessert die Wartbarkeit und ermöglicht es Entwicklern aus der Community, einfacher Änderungen beizusteuern. Umfangreiche Anpassungen wurden bei der Art und Weise vorgenommen, wie das virtuelle DOM erzeugt wird. Der neue Compiler ist in der Lage, Optimierungen durchzuführen, um so viele unnötige Render-Durchläufe wie möglich einzusparen.

Mit dem Rewrite von Vue wurden daher zahlreiche neue Möglichkeiten geschaffen und alte Probleme ausgemerzt, sodass die neue Version von Vue zahlreiche Vorteile gegenüber der Version 2 besitzt.

3.2 Ein mehrschrittiges Release

Vue hat sich in Version 3 deutlich stärker bestehenden Web-Frameworks angenähert, was unter anderem am Composition API liegt, mit dem sich lose gekoppelte Komponenten implementieren lassen. Da sich dieses Buch auf Vue 3 fokussiert, sind alle Beispiele und Erklärungen darauf bezogen. Der gesamte Veröffentlichungsprozess zu Vue 3 war mehrschrittig. Ein Vorabcheck der im Projekt eingesetzten Bibliotheken und Tools ist dennoch Pflicht, damit nicht erst bei der Migration ein Show-Stopper auffällt. Wichtige Punkte einer Migration von Version 2 auf Version 3 behandelt das Buch in Kapitel 17 ebenfalls.

Mit *petite-vue* wird zudem ein spezielles Vue-Release angeboten, das einige besondere Eigenschaften vereint. Diese alternative Distribution von Vue ist für das sogenannte *Progressive Enhancement* optimiert. Dahinter verbirgt sich eine Designphilosophie, die möglichst vielen Nutzern einen Grundstock an wesentlichen Inhalten und Funktionen zur Verfügung stellt, während das bestmögliche Erlebnis nur den Nutzern der modernsten Browser geboten wird,

die den gesamten erforderlichen Code ausführen können. Das Wort *progressiv* in *Progressive Enhancement* bedeutet, dass ein Design geschaffen wird, das für Benutzer älterer Browser und Geräte mit eingeschränkten Möglichkeiten ein einfacheres, aber dennoch brauchbares Erlebnis bietet, während es gleichzeitig ein Design ist, das das Benutzererlebnis für Benutzer neuerer Browser und Geräte mit umfangreicheren Möglichkeiten zu einem überzeugenderen Erlebnis mit vollem Funktionsumfang weiterentwickelt.

Das Projekt bietet die gleiche Template-Syntax und das gleiche Reaktivitäts-Modell wie das Standard-Vue an. Es ist jedoch speziell dafür optimiert, eine kleine Menge an Interaktionen auf eine bestehende HTML-Seite, die von einem Server-Framework gerendert wird, zu streuen. *petite-vue* ist nur sechs Kilobyte groß und arbeitet direkt auf dem DOM. Es wird eine Untermenge der Funktionalität von Vue angeboten.

3.3 Interne und externe Anpassungen

Ganz grundlegend gab es in Vue interne und externe Änderungen. Die externen fallen bei der täglichen Arbeit mit Vue am stärksten auf. Das sind zum Beispiel Anpassungen am API oder am Verhalten von Tools beziehungsweise Bibliotheken. Oft haben sich Schnittstellen geändert oder sind komplett weggefallen. Interne Anpassungen, die in Vue 3 ebenfalls zuhauf vorhanden sind, bleiben dagegen eher im Verborgenen. Dazu gehört prominent die Modularisierung des Web-Frameworks, in offiziellen Dokumenten häufig mit *Layered Internal Modules* beschrieben. Die Kernfunktionen befinden sich jetzt nicht mehr in einem einzelnen und damit umfangreichen Core, sondern in einzelnen Paketen. Das ermöglicht eine Entkopplung, da alle Module eigenständig sind. Vue besitzt nun *drei* Kernmodule: das *Reactivity*-Modul, das *Compiler*-Modul und das *Renderer*-Modul. Ersteres erlaubt es, reaktive JavaScript-Objekte zu erstellen, die auf Änderungen überwacht werden können. Das Compiler-Modul besitzt die Implementierungen, die notwendig sind, um HTML-Templates zu Render-Funktionen zu kompilieren. Meistens läuft dieser Prozess, wenn unser Vue-Projekt erzeugt wird. Das Renderer-Modul enthält Implementierungen für *drei* verschiedene Phasen, um eine Vue-Komponente in einer Webseite zu rendern: die *Render*-Phase, *Mount*-Phase und *Patch*-Phase.

Die folgende Liste fasst die wichtigsten Änderungen zusammen, die in Vue 3 Einzug gehalten haben. Zusätzlich zeigt die Liste auch wichtige Änderungen im Ökosystem. Ein Deep Dive zu diesen Features ist in den nachfolgenden Abschnitten und Kapiteln enthalten.

- Änderungen beim Mounten (Global Mounting) (Abschnitt 2.1.3)
- Verbesserte Reaktivität (Kapitel 5)
- Fragmente und native Portale durch Teleports (Kapitel 6)
- Anpassungen bei CSS-Variablen (Kapitel 7)
- Composition API (Kapitel 8)

Beim Ökosystem sieht es mit der Unterstützung für Vue 3 ebenfalls sehr gut aus. Die primären Tools und Bibliotheken wie das Vue CLI (Kapitel 4), der Vue Router (Kapitel 9) und Vuex (Kapitel 10) haben sehr früh in Major-Versionen Anpassungen für Vue 3 bekommen. Das gilt ebenso für zahlreiche Plug-ins, zum Beispiel Vetur für die Integration von Vue 3 in Entwicklungsumgebungen (Abschnitt 4.3). Dem Einsatz von Vue 3 steht damit bereits seit dem Frühjahr 2021 nichts mehr im Wege. Im produktiven Kontext ist der Einsatz seit circa Sommer 2021 empfehlenswert.

3.4 Der neue Standard und Framework-Empfehlungen

Wie bereits kurz angesprochen, ist Vue 3 der neue Standard. Das Ökosystem hat sich angepasst und neue Projekte, zum Beispiel über das CLI, werden mit Vue 3 als Standard erzeugt, und die offizielle Dokumentation nimmt auch Vue 3 als Standard an. Das bedeutet auch, dass Repositorys und Links von CDNs nicht mehr auf den *next*-Tag verweisen. Hier sind eventuell Anpassungen im eigenen Projekt notwendig, um dieser Änderung Rechnung zu tragen.

Zusätzlich gibt es eine Reihe von neuen *Framework-Empfehlungen*. Einige davon sind sicherlich überraschend, zeigen allerdings auch, wie viel sich in den letzten Jahren und Monaten getan hat. Die neuen Standard-Empfehlung zu Vue 3 sind die folgenden:

- Neue Versionen von Vue Router, den Devtools und den Test Utils mit Vue-3-Support.
- Beim IDE-Support gibt es einen Wechsel von Vetur auf Volar.
- Die Build-Toolchain wechselt von Vue CLI auf Vite.
- Beim State Management gibt es einen Wechsel von Vuex auf Pinia, das besser mit dem Composition API zusammenarbeitet.
- Den Support auf der Kommandozeile für TypeScript übernimmt jetzt vue-tsc.
- VitePress als Ersatz für VuePress als Generator für statische Seiten.

Das sind zahlreiche Änderungen, die für sich genommen schon einiges Feedback in der Community ausgelöst haben. Zum Beispiel der Wechsel auf Vite an praktisch allen Fronten und der Wechsel auf Pinia für das State Management. Die Änderungen sind aber durchweg positiv und versprechen auch in Zukunft viele Verbesserungen.

Index

4 + 1 Kernkonzepte 172

A

Accessibility, a11y 215

Animation 108

Application Instance 10

Array 94

Array Change Detection 94

Array-Syntax 107

Axios 193

Azure DevOps 257

B

Backend 283

Barrierefreiheit 215

Base Component Pattern 84

Bedingtes Rendering 89

Build 253

Bundle Size 259

C

Class-Binding 106

Component-based Software Engineering (CBSE) 47

Composable (Composition Function) 122

Composable Components 3

Composition API 52, 113

 Unit-Tests 238

Composition Functions 122

Computed Property 40, 67, 121

Configuration Props 83

Content Delivery Network 10

Custom Event 65

D

Data Binding 85–86

datebasierte Projektstruktur 208

deep reactive 45

Dependency Injection 80

Deployment 253

Document Object Model (DOM) 12, 19

DOM Event 65

E

E2E-Test 230

Eigene Direktive 76

Event 102

- Benutzerdefiniertes Event 64

- Event testen 239

F

Fehlerbehandlung 208

Fetch 193

Fetch API 192

Fluent API 184

Flux-Pattern 165

Formulare 85, 99, 104

- Formulare testen 241

Fragment 77

Function-based Component API 113

G

Geltungsbereichs 52

Gesamtgröße 259

GitHub Actions 257

Global Mounting 13

Globales API 266

H

History-Modus 151

Hot Reloading 28

I

inject 129

Interface Layer 5

Internationalisierung 215

Internationalisierung testen 251

Internationalization, i18n 215

J

JSX 81

K

Key-Attribut 267

Komponente

asynchrone Komponente 79

dynamische Komponente 75

funktionale Komponente 79

globale Komponente 52

kombinierbarer Komponente 3

Komponente testen 235

lokale Komponente 52

Komponentenmodell 47

Komponentenorientierte Entwicklung 48

Komponenten-Pattern 83

L

Lazy Loading 153

Lifecycle-Hooks 57, 120

Listen 91

Localization, l10n 215

M

Metadaten 156

Migration 263, 282

Migration-Build 263
Model-View-ViewModel 3, 86
modulbasierte Projektstruktur 208
Multi-Page Application 205
Mustache-Syntax 12

N

Nested Route 153
Node.js 283
Non-Prop-Attribut 61

O

One-Way-down-Binding 66
Options API 50

P

Pagination 97
Pattern-Matching 150
Performance 259

- Events und Reactivity 261
- Performance prüfen 260

Pinia 165, 185
Plug-ins 76
Portal 77
Production-Build 253
Property 40

- Global Property 62

Property Shorthand 53
Props 58
Prototyping 31
provide 129
Proxy 41
Proxy-Objekt 41

Q

Quasar 213

R

- reactive 127
- Reactive Data Binding 3
- Reactivity Transform 7, 127
- Reactivity-System 37, 39
- reaktive Datenbindung 3
- Reconciliation 20
- Redirect 153
- Ref 61
- ref 127
- Render Function 130
- Render-Funktion 81
- Route 141
 - dynamische Route 146
 - Routen übersetzen 224
 - verschachtelte Route 153
- Route-Guard 156

S

- Scaffolding 27
- Schnittstellenschicht 5
- Scope 52
- Separation of Concerns 2, 49
- Server-Side Rendering (SSR) 35
- Setup 116
- Single Page Application 3
- Single Point of Truth 48
- Single-File Components (SFC) 49, 54
- Single-Page Application 205
- Slots 72
 - Named Slots 72
 - Scoped Slots 72
 - Standard Slots 72
- Snapshot-Tests 250
- State 172
- State Management 38, 165
- statische Seiten 212

Store 177–178, 184

Style 85

Style-Binding 106

Suspense 77

T

Teleport 78

Template-Direktive 267

Templates Ref 130

Testen 229

Text-Interpolation 88

Transition 108

TypeScript 273

U

Unit-Test 230, 234

unwrapped 121

V

Vendor-Präfix 108

v-for 91

v-if 89

Vite 32

v-model 70

v-show 89

Vue CLI 29

Vue l18n 216

Vue Router 139

Tests für Vue Router 241

Vuex 165, 169

Tests für Vuex 243

W

Watcher 68, 121

Web Components 82

Z

Zombie Notification 166

Zustand 172

Zustandsmanagement 38

Zwei-Wege-Data-Binding 16