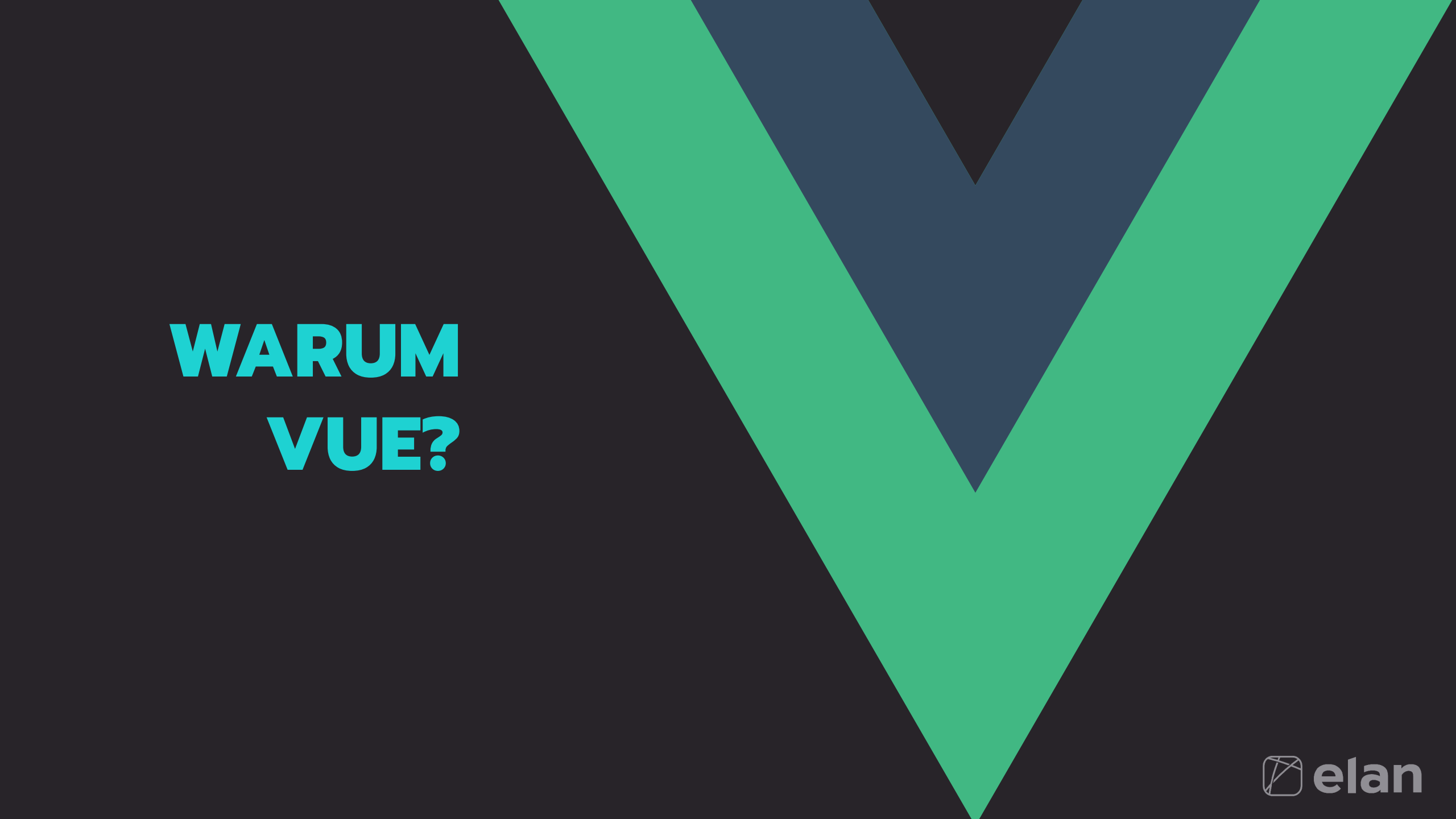
WORKSHOP: VUE 3

Best practices und hands on coding

Ron Lucke – elan e.V.
Till Glöggler – elan e.V.

WARUM VUE?



elan

WARUM VUE?

- Starke Trennung von Logik und UI
- Reaktive Datenbindung
- Einfacher Einstieg und erweiterbar

VUE FÜR MODERNE WEBANWENDUNGEN

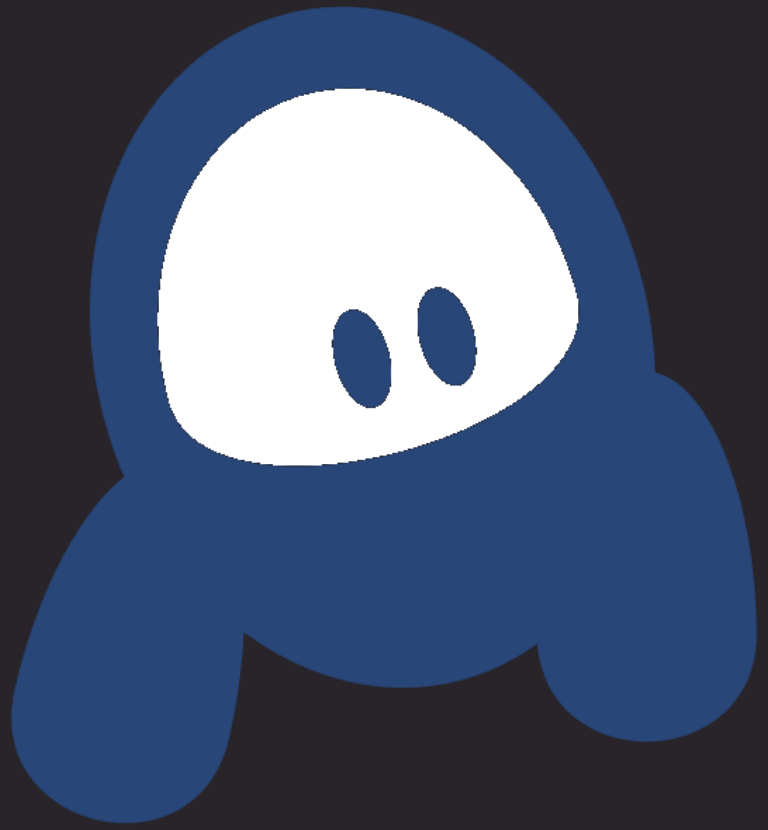
- Möglichkeit, moderne UI-Patterns umzusetzen (z.B. Single-Page-Applications, dynamische UI)
- Mehr Interaktivität und weniger Seitenreloads durch reaktive Datenbindung
- Echtzeit-Updates ohne komplettes Neuladen der Seite

FLEXIBILITÄT IM DESIGN

- Einfaches Umsetzen von interaktiven UI-Elementen, die modern und funktional sind
- Design-Komponenten, die einfach wiederverwendbar und anpassbar sind
- Die Möglichkeit, Design und Logik klar zu trennen, was zu einem konsistenteren UX führt

WAS SIND DIE AKTUELLEN PROBLEME?

- Schwierigkeiten bei der Wartung von PHP-Templates
- Veraltete jQuery-Logik ohne klare Struktur
- Schlechte Skalierbarkeit bei wachsenden Projekten



AKTUELLER STAND IN STUD.IP

elan

WO WIRD VUE GENUTZT?

- Courseware
- Formularbaukasten
- Garuda
- Bilderpool
- Avatar
- Studienbereiche
- etc.

ERFOLGREICHE INTEGRATION VON VUE

- Was verstehen wir darunter?
- Was ist erfolgreich integriert?
- Was ist nicht erfolgreich integriert?

WAS LÄUFT NOCH NICHT OPTIMAL?

- Wir haben immer noch jQuery
- Vue wird nicht einheitlich verwendet
- Trotz vue3 findet man noch jQuery in vue Code

VON ALT NACH NEU

elan

BEST PRACTICES

Einhaltung der Vue-Konventionen

kein Mischung mit jQuery, korrekte Namensgebung, Bindings verwenden

Komponententrennung und Wiederverwendbarkeit

Komponenten sind nicht wie PHP-Templates, allgemeingültiger Aufbau von Komponenten, Nicht wiederverwendbare Komponenten markieren

VUE 2 VS. VUE 3

- Verbesserte Performance und kleinere Bundle-Größe
- Composition API für flexiblere Struktur
- Verbesserte TypeScript-Unterstützung

MIGRATION BESTEHENDER KOMPONENTEN

BRAUCHEN WIR DIE?

Umstellung von Options API auf Composition API

Composition API vs. Options API

Feature	Options API 🏛️	Composition API 🚀
Struktur	Getrennte Optionen (`data` , `methods` , `computed`)	Alles in `setup()` gebündelt
Lesbarkeit	Klare Trennung, aber bei komplexen Komponenten zerstreut	Bessere Gruppierung verwandter Logik
Wiederverwendbarkeit	**Mixins** (können Namenskollisionen verursachen)	**Composables** (klarere Struktur, keine Konflikte)
TypeScript-Support	Möglich, aber eingeschränkter	Besserer TypeScript-Support durch direkte Nutzung von JS-Funktionen
Reaktivität	Automatische Reaktivität mit `data()`	Explizite Reaktivität mit `ref()` und `reactive()`
Performance	Etwas Overhead durch Proxy-Generierung	Effizienter, da direkter Zugriff auf Reaktivitäts-APIs

Mixins vs. Composables

Feature	Mixins 🏛️	Composables 🚀
Verwendung	In Komponenten importiert	Normale JS-Funktionen, die reaktive Werte zurückgeben
Kapselung	Können globale Namenskollisionen verursachen	Kein globaler Kontext, sauber gekapselt
Flexibilität	Eingeschränkte Wiederverwendbarkeit	Sehr flexibel und unabhängig nutzbar
Lesbarkeit	Code ist über verschiedene Optionen verteilt	Klar strukturiert, Logik in einer Datei

Wann welche API?

✅ Options API & Mixins

- Gut für kleinere Projekte
- Einfacher für Anfänger
- Vertraut für Vue 2-Nutzer

🚀 Composition API & Composables

- Besser für größere, skalierbare Projekte
- Mehr Wiederverwendbarkeit
- Bessere Struktur
- Beste Wahl für TypeScript

Vuex vs Pinia - Einführung

Vuex (Vue 2.x / Vue 3)

- State Management für Vue: Verwendet zentralen Store für Anwendungszustand
- Mutations: State-Änderungen nur durch Mutations möglich
- Actions: Asynchrone Operationen, die Mutations auslösen
- Komplexer API: Mehr Boilerplate und Struktur erforderlich
- Vue 2.x: Standard in Vue 2 und oft verwendet

Pinia (Vue 3)

- State Management für Vue 3: Neue offizielle Bibliothek für Vue 3
- Direct State Access: Direkter Zugriff auf den State ohne Mutations
- Stores: Einfachere API durch Stores (reaktive Objekte)
- Bessere Integration mit Composition API: Entwickelt für Vue 3 und Composition API
- Klein und leichtgewichtig: Weniger Boilerplate, einfach zu verwenden


SREATEGIE FÜR DIE ZUKUNFT
VUE ALS CODE-RICHTLINIE
elan

ZUKÜNFTIGE ENTWICKLUNGEN

Vue als zentrale Technologie?

neue Entwicklungen nur mit Vue umsetzen, notwendige Technologien ergänzen oder ersetzen (vuex -> pinia)

Förderung von Vue-Expertise

Entwickler-Teams sollten sich gemeinsam die Fähigkeiten für die Entwicklung mit Vue aneignen

DAS UI-KIT

- Vue als primäre Technologie für UI-Komponenten
- HTML-Struktur-Vorlagen als Fallback, Architektur-Patterns können in PHP-Templates verwendet werden

VUE ALS STANDARD?

- Einheitlicher Code-Stil und Wartbarkeit
- Klarheit über Technologieentscheidungen für neue Entwicklungen

LET'S TALK ABOUT VUE


WAS HINDERT DICH AM EINSATZ VON VUE?
elan

WAS HINDERT DICH AM EINSATZ VON VUE?

- Mangelnde Erfahrung im Team
- Technische Schulden in bestehenden Komponenten
- Unklarheit über Migration und Integration


WAS BRAUCHEN WIR FÜR DEN ERFOLG?
elan

WAS BRAUCHEN WIR FÜR DEN ERFOLG?

- Workshop: Hands on Vue
- Schulungen und Dokumentation für den Einsatz von Vue in Stud.IP
- Klare Entscheidung, Vue als Standard zu etablieren

VIELEN DANK FÜR DIE AUFMERKSAMKEIT

elan