immoGenio

No. 02
Mai 2026

GENIOFLOW

GenioFlow Engine: Guard-Sichtbarkeit, Payload-Schemas mit Zod und automatische Letter-Drafts pro Transition

GenioFlow Engine im Detail: Guard-Sichtbarkeit als UX-Prinzip, Zod-Payload-Schemas, Letter-Drafts pro Transition und actor_type-Audit für Hausverwaltungen.

AUTOR
ImmoGenio

VERÖFFENTLICHT
12. Mai 2026

ONLINE
www.immogenio.de/blog

— AUF EINEN BLICK

Inhalt

Wenn ein Button fehlt — und niemand weiß, warum

Es ist Dienstagvormittag, der Verwalter öffnet das Mietverhältnis Walter-Rathenau-Straße 12, Wohnung 03 links. Der Mietspiegel zeigt für die Lage und Baualtersklasse eine ortsübliche Vergleichsmiete, die deutlich über der vereinbarten Miete liegt. Der Verwalter klickt im Cockpit auf den Tab „Aktionen" und sucht den Button „Mieterhöhung beantragen". Der Button ist nicht da. Stattdessen sieht er die Aktionen „Kaution verzinsen", „Nebenkosten abrechnen", „Mietverhältnis kündigen" — aber Mieterhöhung fehlt. Im klassischen Verwaltungstool-Design ist genau hier das Problem zu Ende: Der Verwalter weiß nicht, warum der Button fehlt. Hat er ein Recht zu wenig? Ist das Mietverhältnis falsch konfiguriert? Ist die Funktion noch nicht freigeschaltet? Vielleicht ist die Software defekt.

In GenioFlow ist der Button da. Er ist nur ausgegraut, mit einem Hinweissymbol daneben. Der Verwalter fährt mit der Maus darüber, und ein Tooltip erscheint: „Sperrfrist nach § 558 Abs. 1 BGB noch 4 Monate aktiv. Erhöhung frühestens am 15.09.2026 zulässig." Damit ist die Frage beantwortet, bevor sie zum Support-Ticket wird. Die scheinbar kleine Designentscheidung — alle Aktionen anzeigen, blockierte mit Begründung markieren — ist eines der zentralen Prinzipien der GenioFlow Engine. In diesem Beitrag beschreiben wir die Infrastruktur dahinter: Zustandsmaschinen auf XState v5, Guard-Sichtbarkeit als UX-Prinzip, Payload-Schemas auf Basis von Zod, automatisch erzeugte Brief-Entwürfe pro Transition und ein Audit-Modell mit drei Akteurstypen.

XState v5 als Wahl

Hinter jedem Workflow in GenioFlow steht eine deklarative Zustandsmaschine, modelliert mit XState v5. Diese Entscheidung ist bewusst getroffen und nicht selbstverständlich. Eine einfachere Implementierung wäre, jede Transition als if-else-Kette im Service zu führen: „Wenn Status gleich `entwurf` und Datum gleich heute, dann setze Status auf `versandt`." Solche Implementierungen funktionieren auf den ersten Blick — und werden ab dem fünften Zustand zur Spaghetti-Logik, die niemand mehr nachvollziehen kann.

Zustandsmaschinen lösen das, indem sie die erlaubten Übergänge formal beschreiben. Eine `mahnstufe_2` kann nur dann in den Zustand `mahnstufe_3` übergehen, wenn das Event `eskalieren` empfangen wird und der Guard `kuendigungsandrohungFristAbgelaufen` `true` zurückgibt. Was nicht im Maschinen-Graphen steht, kann technisch nicht passieren. Der Code wird damit verifizierbar, statisch typisiert und im Stately Inspector visualisierbar.

XState v5 bringt drei Eigenschaften, die für die Hausverwaltungs-Domäne entscheidend sind. Erstens: TypeScript-First. Die Event-Typen und Kontext-Strukturen werden vom Compiler geprüft, eine vergessene Property fliegt vor dem Deploy auf. Zweitens: Saubere Trennung von Guards, Actions und Side-Effects. Guards entscheiden, ob ein Übergang erlaubt ist; Actions verändern den Kontext der Maschine; Side-Effects (im XState-Jargon Actors genannt) erledigen externe Arbeit wie das Anlegen eines Briefes oder den Versand einer E-Mail. Drittens: Die Maschinen sind serialisierbar. Der Zustand kann in der Datenbank persistiert und nach Server-Neustart fortgesetzt werden — eine Voraussetzung für Workflows, die über Tage oder Monate laufen.

Die Faustregel für GenioFlow: Sobald ein Prozess mehr als fünf Zustände und mehr als drei verzweigte Übergänge hat, lohnt sich der Aufwand einer expliziten Maschine. Unterhalb dieser Schwelle ist ein einfaches Statusfeld pragmatischer. Über dieser Schwelle wird ein freies Statusfeld zur Haftungsfalle.

Guard-Sichtbarkeit als UX-Prinzip

Die zentrale UX-Entscheidung der GenioFlow Engine ist die Sichtbarkeit von Guards. Klassische Workflow-Implementierungen blenden Aktionen aus, deren Guards nicht erfüllt sind. Aus Software-Sicht wirkt das aufgeräumt: Was nicht ausgeführt werden kann, wird nicht angezeigt. Aus Anwender-Sicht ist es das Gegenteil von aufgeräumt — es ist verwirrend. Der Verwalter weiß, dass es eine Mieterhöhung gibt, er erwartet den Button, und wenn er ihn nicht findet, beginnt die Fehlersuche an der falschen Stelle.

GenioFlow dreht die Logik um. Das Backend liefert auf jeder Detailseite die vollständige Liste der für den aktuellen Zustand definierten Transitions zurück, nicht nur die ausführbaren. Jede Transition kommt mit drei Feldern aus dem Engine-Layer:

- `event`: Der technische Event-Name (`mieterhoehung_beantragen`).
- `label`: Die menschenlesbare Beschriftung („Mieterhöhung beantragen").
- `disabled`: Ein boolean, das angibt, ob der Guard aktuell blockiert.
- `disabledReason`: Bei `disabled = true` eine konkrete, dem Fachjargon entsprechende Begründung.

Die Begründung ist die entscheidende Information. Statt „Aktion nicht verfügbar" zeigt die UI „Sperrfrist nach § 558 Abs. 1 BGB noch 4 Monate aktiv". Statt „Bedingung nicht erfüllt" steht dort „Kappungsgrenze nach § 558 Abs. 3 BGB würde überschritten — maximal zulässig: 612,00 €". Diese Texte werden zentral in `guard-reasons.ts` gepflegt, damit eine Änderung der Formulierung sich überall konsistent niederschlägt. Die Begründungs-Texte sind eigene Strings im Repository, keine vom Compiler generierten Fragmente, und werden gemeinsam mit den Übersetzungs-Texten gereviewt.

Die UI rendert blockierte Buttons als `disabled` mit einem dezenten Hinweissymbol. Hover oder Long-Press auf Touch-Geräten öffnet einen Tooltip mit der Begründung. Auf Mobilgeräten erscheint stattdessen ein leicht antippbares Popover mit derselben Information. Der Verwalter sieht damit nicht nur, was er gerade nicht tun kann — er sieht auch, was er ändern müsste, damit es ginge.

Die Konsequenz für die Backend-Sicherheit darf nicht unterschätzt werden. Würde ein Angreifer im Frontend den `disabled`-Status manipulieren und den Button auf „enabled" patchen, käme die ausgelöste Transition trotzdem nicht durch. Die Engine prüft beim Aufruf von `transition()` zunächst per `snapshot.can(event)`, ob der Guard erfüllt ist. Schlägt diese Prüfung fehl, wird ein typisierter `ConflictError` mit HTTP-Status 409 zurückgegeben — bevor irgendein Side-Effect läuft. Frontend und Backend reden also über dieselbe Wahrheit, und die Wahrheit liegt in der Maschine. Die Verteidigungslinie ist doppelt: Die UI zeigt den korrekten Zustand zur besten User-Experience, und das Backend lehnt manipulierte Requests konsistent ab.

Payload-Schemas mit Zod

Nicht jede Transition kommt ohne Eingabe aus. Eine Mieterhöhung braucht den neuen Mietbetrag und das Gültigkeitsdatum. Eine Mahnstufen-Eskalation braucht den optionalen Kommentar des Verwalters. Eine Kündigung braucht den Kündigungsgrund nach § 573 BGB und das gewünschte Endedatum. In freier Workflow-Software wird das jedes Mal als handgeschriebenes Modal implementiert — mit allen Risiken der inkonsistenten Validierung, fehlender Pflichtfelder und veralteter Formulare.

GenioFlow geht einen anderen Weg. Jede Transition kann ein Zod-Schema als `payloadSchema` deklarieren. Das Schema beschreibt deklarativ die erwarteten Felder, ihre Typen, optionale und Pflicht-Eigenschaften, Validierungs-Regeln und sogar voneinander abhängige Bedingungen wie „Endedatum muss nach Kündigungsdatum liegen". Die Schemas sind in der Lib `@assetfux/workflow-payload-schemas` zentral abgelegt und werden gleichzeitig vom Backend für die Validierung und vom Frontend für die Formular-Generierung verwendet.

Die UI rendert daraus dynamisch ein FormModal. Eine `string`-Property wird zum Text-Input, ein `number` zum Numeric-Input mit Min/Max-Constraints, ein `date` zum Date-Picker mit dem konfigurierten Frist-Offset, ein `enum` zum Select. Validierungs-Fehler werden inline angezeigt, weil das identische Schema im Browser läuft. Pflichtfelder werden visuell markiert. Optional-Felder können aufgeklappt werden. Das FormModal liegt z-Index-technisch über dem Workflow-Drawer (`!z-[220]` über dem Sheet-Overlay `!z-[215]` über dem Sheet selbst auf `z-[210]`) — eine Detailfrage des CSS-Stacking, die ohne Konvention zu unsichtbaren Bugs führt.

Die zentrale Validierungsregel lautet: doppelt validieren. Das Frontend validiert für die UX — sofort, fehlerarm und mit Inline-Feedback. Das Backend validiert für die Sicherheit, bevor eine Transition den State ändert. Dieselbe Schema-Datei wird beide Male importiert; eine Drift zwischen Frontend- und Backend-Regeln ist damit ausgeschlossen. Bei Schema-Änderungen genügt ein Commit, und beide Seiten ziehen automatisch nach.

Sub-Workflows werden in der Schema-Hierarchie als nested Objekte modelliert. Wenn der Master-Workflow „Vermietung" einen Sub-Workflow „Wohnungsübergabe" auslöst, erbt der Sub-Workflow strukturierte Felder vom Master (Mietvertrags-ID, Beteiligte), kann aber eigene Pflichtfelder ergänzen (Übergabedatum, Zählerstände nach DIN 18017 für Lüftungsanlagen). Die Vererbungs-Logik wird über `z.intersection()` ausgedrückt, was die Typen-Kompatibilität zwischen Master- und Sub-Workflow zur Compile-Zeit erzwingt.

Automatische Letter-Drafts

Viele Workflow-Schritte in der Hausverwaltung münden in einen Brief. Die zweite Mahnung geht raus, das Mieterhöhungsverlangen wird zugestellt, die Kündigung wird postalisch versandt. In klassischer Software ist das ein manueller Schritt: Verwalter klickt „Brief erstellen", wählt eine Vorlage, befüllt die Platzhalter und versendet das Ergebnis. Vier Klicks, drei Wartezeiten und potenziell zwei Tippfehler später ist der Brief raus.

GenioFlow automatisiert diesen Schritt über Workflow-Template-Bindings. Jeder Workflow-State, der einen brieflichen Ausgang hat, ist mit einer Mustervorlage verknüpft. Beim Eintritt in den State `mieterhoehung_brief_versand` feuert ein `entry`-Action der Maschine eine Datenbank-Operation, die einen Brief-Draft mit der bindenden Vorlage erzeugt. Der Draft enthält bereits alle Platzhalter ausgefüllt — Mieter-Anschrift aus dem Mietvertrag, Mietbetrag aus dem Payload, Gültigkeitsdatum aus dem Workflow-Kontext, Verwalter-Signatur aus den Tenant-Einstellungen.

Der Draft landet in der `WorkflowDraftsInbox`, einer dedizierten Ansicht im Portal. Der Verwalter prüft das Ergebnis, kann es im BriefEditor mit Markdown-WYSIWYG-Oberfläche und DIN-5008-konformer PDF-Vorschau anpassen und löst den Versand aus — entweder per Pingen für die digitale Zustellung oder per LetterXpress für den physischen Druck-und-Frankier-Service. Beide Versandwege liefern eine Zustellungsbestätigung zurück. Diese Bestätigung wird als Event in den Workflow zurückgespielt: Sobald sie eintrifft, wechselt der Workflow vom State `mieterhoehung_brief_versand` in den State `mieterhoehung_zustellung_dokumentiert` und startet damit die nächste Phase — etwa die Drei-Monats-Frist nach § 558b Abs. 2 BGB bis zur Wirksamkeit der Erhöhung.

Der gesamte Brief-Lebenszyklus — Erzeugung, Bearbeitung, Versand, Zustellung — wird damit Teil des Workflow-Audits. Wer den Draft erstellt hat, wer ihn bearbeitet hat, wann er versendet wurde und wann die Zustellung bestätigt wurde, lässt sich aus einer einzigen

Quelle rekonstruieren. Das ist insbesondere bei Mieterhöhungen relevant, deren Wirksamkeit nach § 558b BGB an den Zugang gebunden ist — die Engine kann den Zugang aus der Zustellungs-Bestätigung deterministisch errechnen, ohne dass der Verwalter das Datum nachträglich aus der gesendeten E-Mail oder dem LetterXpress-Report extrahieren muss.

Audit mit actor_type

Jedes Workflow-Event in GenioFlow wird in der Tabelle `workflow_events` protokolliert — append-only, wie im Beitrag zum revisionssicheren Audit-Trail nach GoBD, HGB § 257, AO § 147 und DSGVO Art. 5 Abs. 2 ausführlich beschrieben. Eine Eigenschaft, die für die forensische Auswertbarkeit entscheidend ist, ist das Feld `actor_type`. Es kennt drei Werte:

- `user`: Eine konkrete natürliche Person hat das Event ausgelöst, identifiziert über `actor_user_id`. Beispiele: Der Verwalter klickt „Mahnstufe 2 senden", die Buchhaltungsfachkraft setzt einen Sondervorgang.
- `system`: Ein synchroner Side-Effect der Engine selbst, ausgelöst durch eine andere Transition. Beispiel: Wenn das Versenden eines Briefes erfolgreich war, schreibt der Engine-Worker ein `system`-Event zurück, das den Workflow weiterführt.
- `cron`: Ein zeitgesteuerter Hintergrund-Job hat das Event ausgelöst. Beispiel: Der stündliche Banking-Sync erkennt einen Zahlungseingang und schließt damit eine offene Mahnung.

Zusätzlich trägt jedes Event ein `label`-Feld mit der menschenlesbaren Beschriftung des Übergangs („Mahnstufe 2 versendet", „Workflow durch Zahlungseingang abgeschlossen"). Damit ist die Audit-Timeline im Portal nicht mehr eine Liste technischer Event-Namen, sondern eine lesbare Chronologie. Die UI markiert die drei `actor_type`-Werte visuell unterschiedlich: `user`-Events bekommen einen blauen Punkt, `system`-Events einen grauen, `cron`-Events einen orangen. Die forensische Frage „Wer hat das ausgelöst?" wird damit in Sekunden beantwortbar — kein Tool-Wechsel, kein Logfile-Greppen.

Compliance-Anker

Die Engine-Infrastruktur ist nicht nur architektonisch motiviert, sondern verankert konkrete regulatorische Anforderungen.

- § 257 HGB und § 147 AO verlangen die zehnjährige Aufbewahrung aufzeichnungspflichtiger Daten. Der Workflow-Audit ist Teil der Buchführung, wenn er buchungsrelevante Vorgänge dokumentiert (Mahnungen, Mieterhöhungen). Die Append-only-Logik der `workflow_events`-Tabelle stellt sicher, dass keine Mutation den Trail manipulieren kann.

- Die GoBD (BMF-Schreiben vom 28.11.2019) fordert Nachvollziehbarkeit, Unveränderbarkeit und Vollständigkeit. Die Kombination aus expliziter State Machine, append-only Audit und doppelter Validierung erfüllt diese Anforderungen technisch.
- DSGVO Art. 5 Abs. 1 lit. c (Datenminimierung) spiegelt sich darin, dass der Audit ausschließlich Workflow-relevante Daten speichert. Personenbezogene Daten, die nicht für die Rekonstruktion des Vorgangs nötig sind, werden vor dem Schreiben verworfen.
- DSGVO Art. 32 (Technische und organisatorische Maßnahmen) wird durch die Backend-Guards als TOM dokumentierbar. Eine Mieterhöhung lässt sich vor Fristablauf nicht versenden — das ist nicht Organisations-, sondern Technik-Logik und damit auditierbar.
- § 558 Abs. 1 BGB (15-Monats-Sperrfrist) und § 573c BGB (Kündigungsfristen) sind in den Guards der jeweiligen Workflows verankert. Die Fristen werden nicht durch Benutzer-Disziplin gewahrt, sondern durch die Engine.
- § 543 Abs. 3 BGB (Kündigungsandrohung bei Zahlungsverzug) ist als Guard zwischen Mahnstufe 2 und Mahnstufe 3 implementiert. Der Übergang ist erst möglich, wenn die im Workflow erfasste Androhungsfrist abgelaufen ist.

Grenzen der ersten Engine-Generation

GenioFlow ist eine Engine, kein Workflow-Designer. Die Maschinen werden im Source-Code definiert, nicht im Portal-UI. Wer einen eigenen Workflow gestalten möchte, braucht einen Entwickler — die grafische Workflow-Bearbeitung für Endkunden ist nicht Teil dieser Version. Diese Entscheidung ist bewusst: Eine Workflow-Engine, deren Regeln laienhaft veränderbar sind, ist schwer zu auditieren.

Workflow-übergreifende Cross-Transaktionen sind nicht vorgesehen. Jede Transition ist atomar in einer Datenbank-Transaktion gekapselt. Wenn ein Workflow einen anderen anstößt, geschieht das über Events, nicht über gemeinsame Transaktionen. Das vereinfacht das Locking-Modell, verhindert aber Szenarien, in denen zwei Workflows „atomar zusammen abgeschlossen" werden müssten.

Letter-Drafts sind aktuell an einen Brief-Provider gebunden, der pro Tenant ausgewählt wird. Multi-Versand — derselbe Brief an Pingen *und* an LetterXpress gleichzeitig — wird nicht unterstützt. Das `actor_type`-Vokabular ist auf drei Werte begrenzt; weitere Quellen (API-Clients, Mieter-Portal) werden vorerst unter `user` oder `system` subsumiert.

Häufige Fragen

Wie sieht ein konkretes Payload-Schema aus?

Das Schema für die Transition „Mieterhöhung beantragen" enthält drei Pflichtfelder: `neue_miete` als positiver Geldbetrag im Bereich der Kappungsgrenze, `gueltig_ab` als Date mit dem Constraint, dass es mindestens drei Monate in der Zukunft liegt (entsprechend § 558b Abs. 2 BGB), und `begruendung_typ` als Enum mit den Werten `mietspiegel`, `gutachten` oder `vergleichswohnungen` nach § 558a Abs. 1 BGB. Optional kann ein `freitext_zusatz` ergänzt werden. Das Schema validiert sowohl beim Klick auf „Absenden" als auch erneut serverseitig in `transition()`.

Warum doppelte Validierung Frontend und Backend?

Die Frontend-Validierung dient der UX: sofortiges Feedback, keine Round-Trip-Wartezeit, klare Fehlerkennzeichnung am Eingabefeld. Die Backend-Validierung dient der Sicherheit: Selbst wenn das Frontend manipuliert oder umgangen wird, kann kein invalides Payload den Workflow erreichen. Dass beide dasselbe Zod-Schema verwenden, ist der entscheidende Punkt — Drift ist ausgeschlossen.

Was passiert, wenn ein Letter-Draft nicht abgesendet wird?

Der Draft bleibt im Inbox-Status liegen. Der Workflow bleibt im State `mieterhoehung_brief_versand` und blockiert damit weitere Transitions, die einen versandten Brief voraussetzen. Im Verwalter-Cockpit erscheint eine Erinnerung. Nach einer konfigurierbaren Frist (Standard: sieben Tage) wird ein Cron-Event ausgelöst, das den verantwortlichen Mitarbeiter erneut informiert. Ein automatisches Versenden ohne Verwalter-Freigabe findet nicht statt.

Wie lässt sich ein Guard für einen Sonderfall überschreiben?

Gar nicht direkt. Die Guards sind die Sicherheits-Schicht der Engine; ein Override widerspricht dem Konzept. Was möglich ist: ein expliziter „Sondervorgang"-Workflow, der Voraussetzungen abweichend modelliert und seine eigene Begründungspflicht hat. Damit wird die Ausnahme zur dokumentierten Entscheidung, nicht zur stillen Umgehung.

Was bedeutet die Guard-Sichtbarkeits-Entscheidung für die UX?

Sie reduziert Support-Tickets messbar. In Tenants, die GenioFlow seit mindestens drei Monaten produktiv nutzen, ist die Frequenz von Anfragen der Art „warum kann ich X nicht?" um über 70 Prozent gesunken. Der Tooltip beantwortet die Frage am Ort des Klicks, statt sie in den Support zu eskalieren. Der eigentliche Effekt liegt aber tiefer: Verwalter lernen die Regeln des Systems schneller, weil sie die Regeln im Kontext sehen, nicht in einem Handbuch.

Wie wird der `actor_type` bei API-Calls bestimmt?

Aus dem Auth-Kontext. Anfragen mit JWT eines natürlichen Users werden als `actor_type = user` mit der jeweiligen `actor_user_id` markiert. Anfragen von Cron-Workern laufen unter einem Service-Account und werden als `actor_type = cron` markiert. Synchron innerhalb der Engine erzeugte Folge-Events tragen `actor_type = system`. Der Auth-Layer setzt das Feld vor der Trigger-Ausführung; der Trigger selbst kann den Wert nicht überschreiben.

Verwandte Beiträge

- GenioFlow — Die Evolution der Immobilienverwaltung: Geführte Workflows statt freier Statusfelder
- GenioFlow Master-Sub-Workflow: Vermietung-Lifecycle
- Audit-Trail in der Hausverwaltungs-Software: Append-only, Trigger und Revisionssicherheit
- BriefEditor: Markdown-WYSIWYG-Oberfläche und DIN-5008-konforme PDF-Vorschau

Wo wir stehen

Guard-Sichtbarkeit, Payload-Schemas mit Zod und automatische Letter-Drafts sind in allen 22 produktiv aktiven GenioFlow-Workflows umgesetzt. Jede Transition liefert die `disabled`- und `disabledReason`-Felder zurück, jeder Workflow mit Brief-Versand hat sein Template-Binding, jedes Workflow-Event trägt seinen `actor_type`. Die zentrale Begründungs-Bibliothek `guard-reasons.ts` enthält aktuell über 80 Einträge — vom 15-Monats-Sperrfrist-Hinweis bis zur Kappungsgrenze. Die Drafts-Inbox läuft als eigenständige Portal-Sektion und ist die meistgenutzte Seite im Verwalter-Alltag nach dem Cockpit.

Die Engine-Infrastruktur bleibt der wachsende Teil des Systems. Neue Workflows kommen hinzu, wenn die fachliche Domäne reift. Neue Guards entstehen, wenn der Gesetzgeber neue Fristen einführt. Neue Brief-Templates werden ergänzt, wenn die Verwaltungspraxis neue Anforderungen formuliert. Das Fundament — XState v5, Zod-Schemas, append-only Audit mit `actor_type` — bleibt unverändert. Damit ist GenioFlow eine Plattform, nicht ein Feature.

Fragen, Rückmeldungen oder eigene Erfahrungen

Wenn Sie ähnliche Workflow-Engines betreiben, eigene Erfahrungen mit Zod-basierten Payload-Schemas gemacht haben oder konkrete Detailfragen zur Guard-Sichtbarkeits-Implementierung haben, freuen wir uns über einen Austausch unter kontakt@immogenio.-

de. Die Engine ist ein lebendes System, und die wertvollsten Impulse kommen aus dem Vergleich mit anderen Architektur-Entscheidungen.