SurveyJS: Architektur, Funktionsumfang, Integrationsmuster und Einordnung im Frontend-Formularbau

8. September 2025

Zusammenfassung

SurveyJS ist ein modulares Open-Source-*Form- und Survey-Framework* (MIT-lizenziertes Form-&-Runtime-Kernmodul), das Umfragen und komplexe Formulare in Webanwendungen ermöglicht. Die Laufzeitbibliothek ist in React, Angular, Vue und anderen Frontend-Stacks einsetzbar und wird über eine deklarative JSON-Definition gesteuert. Der Beitrag systematisiert Ziele und Architektur, beschreibt zentrale Features (Fragetypen, Validierung, Ausdrücke/Regeln, Internationalisierung, Barrierefreiheit, Theming), erläutert Integrationsmuster (Events, Datenspeicherung, Server-Anbindung), diskutiert Erweiterungen (Creator, Dashboard, PDF) und bewertet Stärken, Grenzen und typische Einsatzszenarien. Die Darstellung stützt sich auf offizielle Dokumentation und Produktseiten.

1 Einordnung, Zielsetzung und Ökosystem

SurveyJS adressiert die schnelle und zugleich flexible Erstellung interaktiver Formulare und Umfragen in Single-Page-Anwendungen. Kernidee ist die *Trennung* von *Formdefinition (JSON)* und *Rendering/Logik* in der Laufzeitbibliothek. Dadurch lassen sich Formulare deklarativ konfigurieren und in verschiedenen Frameworks einheitlich ausführen. Die offizielle Produktseite positioniert die Bibliothek mit mehreren Komponenten: *Form Library* (Runtime), *Survey Creator* (visueller Editor), *Dashboard* (Auswertung/Charts) und *PDF Generator* (Server-/Client-seitige Generierung) [1, 2].

Lizenzrechtlich gilt: Die *Form Library* (Runtime, Pakete wie `survey-core`, `survey-react-ui`) ist Open Source (MIT). Kommerzielle Komponenten (*Survey Creator*, *Dashboard*, *PDF*) werden per Entwicklerlizenz vertrieben; das Lizenzmodell und die EULA sind öffentlich dokumentiert [3, 4, 5].

2 Architektur und Datenmodellierung

Formdefinition

Die zentrale Formdefinition erfolgt als JSON-Objekt mit Metadaten (Titel, Beschreibung, Layoutparameter) und einer `pages`/`elements`-Struktur. Jedes Element hat einen `type` (z. B. `text`, `radiogroup`, `checkbox`, `matrix`), eine eindeutige `name` und optionale Validierungs- und Darstellungsattribute. Beispielhaft (gekürzt):

```
{
  "title": "Registration",
  "pages": [{
    "elements": [
      { "type": "text", "name": "firstName", "isRequired": true },
      { "type": "text", "name": "lastName" },
      { "type": "text", "name": "age", "inputType": "number" }
    ]
  }]
}
```

Die Laufzeitbibliothek stellt daraus ein *Model* instanziierbar bereit (z. B. `new Model(json)`), das u. a. *Events*, *Validierung*, *Auswertungen* und *Datenzugriff* kapselt [6].

Laufzeit, Framework-Adapter und Events

Der Kern (`survey-core`) enthält Datenmodelle, Ausdrücke und Validierer; Framework-spezifische UI-Pakete (`survey-react-ui`, `survey-vue-ui`, `survey-angular-ui`) übernehmen das Rendering. Anwendungen binden das *Model* und den *Survey*-Komponent ein und reagieren über Events (*onComplete*, *onValueChanged*, *onValidateQuestion* etc.) [1, 6].

3 Funktionsumfang (Features)

Fragetypen und Layout

SurveyJS unterstützt eine breite Palette an Fragetypen: einfache Eingabefelder (Text, Zahl, Datum), Auswahlfragen (`radiogroup`, `checkbox`, `dropdown`), *Panels* und *Panel Dynamics* (wiederholbare Gruppen), Matrizen/*matrixdynamic* (tabellarische Eingabe), Datei-Upload, Signatur-Pad u. a. [1]. Layout und Seitenlogik (mehrseitig vs. Einzelseite) sind konfigurierbar; dynamische Labels/Platzhalter und HTML-Blöcke (`html`) erlauben reichhaltige Formulargestaltung.

Validierung, Regeln und Ausdrücke

Neben einfachen Regeln (`isRequired`, Längen/Typen) stellt SurveyJS eine Ausdruckssprache für *Sichtbarkeitslogik*, *Aktivierungsbedingungen*, *berechnete Werte* (`calculatedValues`) und *Validierung* bereit. Typische Operatoren sind „`notempty`“, Vergleichsoperatoren, String-/Array-Funktionen; komplexe Bedingungen lassen sich verschachteln. Dokumentiert sind u. a. *Conditional Visibility* und *Expressions/Calculated Values* [7, 8, 14].

Internationalisierung und Barrierefreiheit

Die Form Library bringt Übersetzungen und Mechanismen zur Lokalisierung (`locale`, `title_loc`, `choicesByUrl` mit lokalisierbaren Texten). Zur Barrierefreiheit zählen Tastaturnavigation, ARIA-Attribute und Screenreader-Support; die Produktseite nennt *Section 508* und *WCAG*-Konformität als Zielvorgaben [12, 2]. Für barrierearme Gestaltung sind inhaltliche und visuelle Aspekte (Kontrast, Fokusreihenfolge) in den Themes berücksichtigt.

Theming und Styling

Das Styling ist über *Themes* steuerbar (Farben, Typografie, Abstände, Komponentenstates). Die Dokumentation beschreibt Default-Themes, CSS-Klassen und die Anwendung eigener Themes („Manage default themes and styles“) [11]. Zusätzlich steht ein *Theme Builder* auf der Website zur Verfügung (interaktive Anpassung mit Export als JSON/CSS).

Leistungsaspekte

Für große, „content-heavy“ Formulare existieren Muster wie *Lazy Loading*, um Renderkosten zu reduzieren (späte Initialisierung, Seitenweise-Laden) [13]. Allgemein profitieren Performance und Responsiveness von der Trennung zwischen Model und UI-Adapter; das Rendering ist an das jeweilige Frontend-Framework gekoppelt (React/Vue/Angular) und folgt dessen Optimierungsmechanismen.

4 Integration und Speicherung

Datentransfer und Backend-Anbindung

SurveyJS bringt *keine* eigene Persistenz- oder Authentifizierungslösung mit. Die Speicherung von Antworten ist explizit als Integrationsaufgabe dokumentiert („Store Survey Results“). Beispiele zeigen REST-Anbindungen, File- und Datenbank-Backends; offizielle *Server-Beispiele* existieren für Node.js, ASP.NET, PHP [9, 10].

Reaktives Muster (React). Typisch ist die Verarbeitung über das `onComplete`-Event des *Model*. Ein Minimalbeispiel (POST an eine REST-API):

```
import { Model } from "survey-core";
import { Survey } from "survey-react-ui";

const model = new Model(json);
model.onComplete.add(async (sender) => {
  const payload = sender.data; // Antworten als JSON
  await fetch("http://localhost:5050/persons", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(payload)
  });
});

return <Survey model={model} />;
```

Für bidirektionale Szenarien (Vorbelegung/Editing) werden *Daten* über `model.data = ...` gesetzt, Teilantworten über `onValueChanged` synchronisiert. Bei Listen-/Tabellenmustern (`paneldynamic`/`matri`
empfiehlt sich *diff-basiertes* Speichern (neue/aktualisierte/gelöschte Einträge auseinanderhalten), wie man es auch bei `json-server` oder nahen Mock-APIs verwendet.

Sicherheit und Datenschutz

SurveyJS rendert ausschließlich Client-seitig. Die Verantwortung für sichere Übertragung (HTTPS), Speicherung (z. B. Verschlüsselung, Zugriffskontrolle), Löschung/Archivierung und DSGVO-konforme Prozesse liegt beim integrierenden Backend. Für personenbezogene Daten gilt: Minimierung, Verschlüsselung *in transit/at rest* und konsistente Einwilligungs-/Informationsprozesse sind umzusetzen (Best Practices; kein Bestandteil der Form Library).

5 Erweiterungen im SurveyJS-Portfolio

Survey Creator (Form Builder)

Der *Survey Creator* ist ein visueller Editor, der die JSON-Definition generiert. Er richtet sich an Entwicklerteams, Produktmanager oder „Power User“, die Formulare modellieren, ohne JSON

manuell zu schreiben. Das Produkt ist *kommerziell* lizenziert („not available for free commercial usage“); Details sind im Licensing und der EULA festgehalten [3, 4, 5]. Integrierbar ist der Creator als Komponente ins eigene Backoffice.

SurveyJS Dashboard (Analytics)

Das *Dashboard* liefert Diagramme, Filter und Auswertungs-Widgets über erhobene Antworten und ist ebenfalls kommerziell lizenziert [2]. Die Lösung kann in Portale integriert werden, um Fachbereichen Self-Service-Analysen zu ermöglichen.

PDF Generator

Für Compliance-/Reporting-Fälle (Druck, Archiv) steht ein PDF-Generator bereit (Client/Server-Variante). Auch dies ist ein kommerzielles Modul [2, 3].

6 Vergleich und Abgrenzung

Gegenüber Formular-Frameworks mit JSON Schema. SurveyJS verwendet ein *eigenes* Konfigurationsschema (kein JSON Schema Draft-Standard). Im Gegensatz zu bibliotheken wie *JSON Forms* oder *react-jsonschema-form*, die stark um JSON Schema kreisen, fokussiert SurveyJS auf *Survey-/Form-Logik*, dynamische Panels/Matrizen und eine integrierte Ausdruckssprache. Das ist für Survey-Szenarien vorteilhaft; in streng schemagetriebenen Enterprise-Domänen kann die fehlende JSON-Schema-Konformität jedoch ein Nachteil sein (z. B. Validierungswiederverwendung aus Backendschemata).

Gegenüber SaaS-Tools. Gegenüber SaaS-Survey-Plattformen (SurveyMonkey, Typeform etc.) bietet SurveyJS volle *On-Prem/Embedded*-Kontrolle, Versionierbarkeit und keinerlei Vendor-Lock-in auf Datenebene. Dafür müssen Hosting, Sicherheit und Auswertungen (sofern nicht das kommerzielle Dashboard genutzt wird) selbst bereitgestellt werden.

7 Stärken und Grenzen

Stärken

- **Deklarativer Ansatz & breite Fragetypen:** Schlanke JSON-Definition, reichhaltige Controls (inkl. dynamischer Panels/Matrizen) [1].
- **Mächtige Ausdruckssprache:** Sichtbarkeit, Validierung, berechnete Werte, verzahnte Regeln [8, 7].
- **Multi-Framework & Theming:** Adapter für React/Vue/Angular; Themes/CSS anpassbar [1, 11].
- **Barrierefreiheit:** Fokus auf 508/WCAG-Ziele; produktseitige Hinweise [12, 2].
- **Erweiterbares Portfolio:** Creator (visuelles Modeling), Dashboard (Analytics), PDF (Export) [2, 3].

Grenzen / Trade-offs

- **Kein integriertes Backend:** Speicherung/Authentisierung sind Sache der Integration („Store Survey Results“) [9, 10].

- **Eigenes Konfigurationsschema:** Keine direkte Wiederverwendung vorhandener JSON-Schemata; ggf. Transformationsaufwand.
- **Kommerzielle Module:** Creator/Dashboard/PDF sind kostenpflichtig; Lizenz pro Entwickler, separate EULA [3, 4].
- **Komplexität in Großformularen:** Viele Regeln/Abhängigkeiten erfordern saubere Modellierung; Performance-Muster (Lazy Loading) beachten [13].

8 Best Practices

- **Modularisieren:** Große Formulare in Seiten/Panels gliedern; *paneldynamic*/*matrixdynamic* gezielt für wiederholbare Strukturen nutzen.
- **Ausdrücke testen:** Regeln (Sichtbarkeit/Validierung) früh mit realistischen Daten prüfen; *calculatedValues* für Wiederverwendung.
- **Theming zentralisieren:** Ein gemeinsames Theme (CSS/*Theme JSON*) pflegen; Kontraste und Fokusreihenfolgen barrierefrei gestalten.
- **Persistenz klar trennen:** `onComplete`/`onValueChanged` für Datentransfer nutzen; Fehlertoleranz (Retry/Timeout), Validierungen serverseitig spiegeln.
- **Performance bewusst:** Seitenweise Navigation, Lazy Loading, bedarfsgerechte Initialisierung aktiv nutzen [13].

9 Fazit

SurveyJS bietet eine reife, modulare Laufzeit für komplexe, dynamische Webformulare und Umfragen. Die Kombination aus umfangreichen Fragetypen, deklarativem JSON, Ausdruckssprache und Theming ermöglicht produktive Entwicklung in React/Vue/Angular. Die klare Abgrenzung – keine eingebaute Persistenz, dafür offen dokumentierte Integrationspfade – passt gut in moderne Frontend-First-Architekturen. Wer *visuelles Modeling*, *Analytics* oder *PDF* benötigt, kann kommerzielle Module des gleichen Herstellers beziehen. Einschränkungen ergeben sich aus dem proprietären Konfigurationsschema (ohne JSON-Schema-Kompatibilität out-of-the-box) und der Notwendigkeit, Backend-Aspekte sauber zu lösen. Insgesamt ist SurveyJS für *Embedded-Surveys*, *Self-Service-Formulare* und *Headless-Formularbau* eine technisch überzeugende Option.

Literatur

[1] SurveyJS: *Form Library* (Produktseite und Einstieg). `https://surveyjs.io/form-library`.

[2] SurveyJS: *Products Overview* (Form Library, Survey Creator, Dashboard, PDF Generator). `https://surveyjs.io/`.

[3] SurveyJS: *Licensing* (Open-Source- und kommerzielle Komponenten, Lizenzmodelle). `https://surveyjs.io/licensing`.

[4] Devsoft Baltic OÜ: *Commercial developer license for SurveyJS Creator, SurveyJS PDF Generator and SurveyJS Dashboard* (EULA, PDF). `https://surveyjs.io/Developer-License-Agreement.pdf`.

[5] *survey-creator* (npm-Paketseite, Lizenzhinweis: nicht frei für kommerzielle Nutzung). https://www.npmjs.com/package/survey-creator.

[6] SurveyJS Docs: *Documentation* (Model, Events, Integration, API-Referenz). https://surveyjs.io/Documentation/Library?id=overview.

[7] SurveyJS Docs: *Conditional Visibility* (Sichtbarkeitslogik). https://surveyjs.io/Documentation/Library?id=Conditional-Visibility.

[8] SurveyJS Docs: *Expressions and Calculated Values*. https://surveyjs.io/Documentation/Library?id=Expressions-and-calculated-values.

[9] SurveyJS Docs: *Store Survey Results* (Speicherung, Integrationshinweise). https://surveyjs.io/Documentation/Library?id=LibraryOverview#store-survey-results.

[10] SurveyJS Docs: *Servers* (Beispiele/Guides für Node.js, ASP.NET, PHP). https://surveyjs.io/Documentation (Menüpunkt „Servers“).

[11] SurveyJS Docs: *Manage default themes and styles*. https://surveyjs.io/Documentation/Library?id=Manage-default-themes-and-styles.

[12] SurveyJS: *Accessibility Statement* (WCAG/508-Zielsetzung). https://surveyjs.io/AccessibilityStatement.pdf.

[13] SurveyJS Docs: *Lazy loading for content-heavy forms*. https://surveyjs.io/Documentation/Library?id=Examples-Lazy-loading-for-content-heavy-fo

[14] SurveyJS Blog: *What you should know about SurveyJS expressions*. https://surveyjs.medium.com/what-you-should-know-about-surveyjs-expressions-3a680b807ae