universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Modernisierung einer bestehenden WebGIS-Anwendung zu einer komponentenbasierten Softwarearchitektur – Gezeigt am Beispiel Infrastruktur-Datenbank der Wiener Linien

verfasst von | submitted by

Soyol-Erdene Marksteiner BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2026

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 856
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Kartographie und Geoinformation
Betreut von | Supervisor:	Ass.-Prof. Mag. Dr. Andreas Riedl

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während meines Studiums unterstützt und motiviert haben.

Mein herzlicher Dank gilt Herrn Ass.-Prof. Mag. Dr. Andreas Riedl, der meine Masterarbeit betreut und begutachtet hat. Auch nach Jahren als lediglich angemeldete Studentin hat er mich nicht aufgegeben und mit seinen wertvollen Anregungen sowie konstruktivem Feedback durch die Erstellung dieser Arbeit begleitet.

Ein großes Dankeschön den Wiener Linien und meinem Team für themenspezifische Inhalte, zahlreiche interessante Debatten zur Software-Architektur und -Entwicklung sowie die nicht selbstverständliche Entscheidungsfreiheit. Vor allem meinem Gruppenleiter Daniel Dötzl, MSc, danke ich für das Vertrauen bei technischen Experimenten, die trotz aller Herausforderungen maßgeblich zum Erfolg der ISDB-Modernisierung beigetragen haben.

Meiner Familie – Urna, Fabian und meinem Mann, Andreas – gebührt mein tiefster Dank. Sie sind meine größte Motivation, diesen Weg konsequent weiterzugehen.

Mit dieser Arbeit endet mein jahrelanges Studium – ein Weg, der mich nicht nur fachlich, sondern auch persönlich zur heutigen Version meiner selbst geformt hat.

Soyol Marksteiner

Hörfarth, 08.04.2026

Inhaltsverzeichnis

Danksagung ..... **ii**

Inhaltsverzeichnis ..... **iii**

Abbildungsverzeichnis ..... **vi**

Tabellenverzeichnis ..... **viii**

Kurzfassung ..... **ix**

1 Einleitung **1**

1.1 Ausgangslage und Motivation 1

1.2 Zielsetzung und Aufgabenstellung 2

1.3 Stand der Forschung 3

1.4 Forschungsfragen 4

1.5 Methodische Vorgehensweise 5

1.6 Struktur der Arbeit 6

2 Grundlagen der Geoinformatik und Softwaretechnologien **8**

2.1 Grundlagen der Geoinformatik 8

2.1.1 Grundlagen und Definition von GIS 8

2.1.2 Komponenten und Begriff von GIS 8

2.2 WebGIS vs. Webmapping 10

2.2.1 Grundlegende Begriffe und Definitionen 10

2.2.2 Historische Entwicklung von WebGIS 11

2.2.3 Aktuelle Trends in Geoinformatik und WebGIS 12

2.3 Legacy-Systeme und Modernisierung 12

2.3.1 Begriffe und Einordnung von Legacy-Systemen 13

2.3.2 Entscheidung zur Modernisierung 14

2.4 Software und moderne Frameworks 17

2.4.1 Grundlagen moderner Webtechnologien 18

2.4.2 Kriterien der Interoperabilität 18

2.4.3 Grundlagen und Begriffserklärungen von Framework 19
2.4.4 Serverseitige Java-Frameworks 20
2.4.5 Clientseitige SPA-Frameworks 24
2.4.6 SPA-Framework Zusammenfassung 30
2.4.7 Grundlagen von API 32
3 Einführung und Ausgangssituation 37
3.1 Organisatorische Ist-Zustand 37
3.1.1 Historische Entwicklung der ISDB 38
3.1.2 ISDB in der Systemlandschaft 39
3.2 Infrastrukturdatenbank – ihre Funktionen und Rollen 41
3.2.1 Datenerfassung für Übersichten und Auswertungen 41
3.2.2 Datenmodell – Grundkonzept 43
4 Strategische Planung und Zielarchitektur 52
4.1 Zielsetzung der Modernisierungsstrategie 52
4.1.1 Technischer Ist-Zustand und Problemstellung 52
4.2 Zielarchitektur: ISDB-Webanwendung 53
4.2.1 Kernentscheidung: Entkoppelte Architektur mit SPA und API 53
4.2.2 Festlegung von Vue.js als SPA-Zieltechnologie 54
4.3 Modernisierungsstrategie von Backend und Frontend 56
4.3.1 Backend-Übergangsarchitektur: Parallelbetrieb von Struts und REST-API 56
4.4 Strategie zur Frontend-Modernisierung 57
4.4.1 Strategische Zielkriterien 57
4.5 API-Strategie und Designprinzipien 58
4.6 Transformationsstrategie 59
4.6.1 Steuerung, Entscheidungsfindung und Rückfallmechanismen 60
5 Umsetzung und Modernisierung der ISDB 61
5.1 Methodischer Umsetzungsansatz 61
5.2 Implementierung der Backend-Architektur 62

5.2.1 ISDB Backend Strukturumstellung 62
5.2.2 REST-API Implementierung 64
5.2.3 Swagger Dokumentation 69
5.3 Frontend Implementierung mit Vue.js 70
5.4 Architekturstruktur des Vue.js – Frontends 70
5.5 GIS-Komponenten und OpenLayers-Integration 72
5.5.1 Einbindung der Karte als wiederverwendbarer UI-Baustein 73
5.5.2 Karteneditierung in Feature-Detailansichten 74
5.5.3 Layer-Management und Bindung externer Kartendienste 75
5.5.4 Trennung von Datenzugriff und GIS-Funktionslogik 77
5.5.5 Architekturimplikation und Austauschbarkeit 77
5.6 Herausforderungen und Lösungsansätze 78
5.6.1 Paralleler Betrieb von Legacy- und Neusystem 78
5.6.2 Abstimmung und Validierung im Fachbereich 78
5.6.3 API-Design und Schnittstellenqualität 79
5.6.4 ISDB-Projektmanagement als Erfolgsfaktor 79
5.7 Bewertung der Umsetzung und Zielarchitektur 81
5.7.1 Erreichte architektonische Zielsetzungen 81
5.7.2 Quantitative Bewertung anhand KPIs 83
6 Zusammenfassung und Ausblick 87
6.1 Fragestellung und Zielsetzung der Arbeit 87
6.2 Fazit der Forschungsfragen 87
6.2.1 Empfehlungen für vergleichbare WebGIS-Systeme 91
6.2.2 Ausblick und weiterführender Forschungsbedarf 91
7 Anhang 92
8 Literaturverzeichnis 95
9 Hilfsmittelverzeichnis – Einsatz von KI-Tools 98

Abbildungsverzeichnis

Abbildung 1: Der Xerox PARC Map Viewer aus dem Jahr 1993 11

Abbildung 2: Lebenszyklen in Kontext (Annett, R. 2019. S.3) 13

Abbildung 3: Strategy Decision Flowchart (Annett, R. 2019. S.22).....16

Abbildung 4: Model View Controller Muster. Quelle: https://developer.mozilla.org/en-US/docs/Glossary/MVC 21

Abbildung 5: ISDB in die Systemlandschaft der Wiener Linien 40

Abbildung 6: Aktuelle Startseite der ISDB.....41

Abbildung 7: Überblickseite eines Datensatzes (Gleisarbeit) 42

Abbildung 8: Fachliches Komponentenmodell.....43

Abbildung 9: Gleisgraph GIS-Layers als Referenzdaten dargestellt auf einer Basemap.....44

Abbildung 10: Weboberfläche zum Konfigurationsmanagement für Komponentenparameter.....45

Abbildung 11: Detaillierte Nutzungsinformation einer Weichenanlage 46

Abbildung 12: Zustandsdaten einer Weichenanlage inkl. Bewertungsnoten sowie Datum.....47

Abbildung 13: GIS-Verortung von Infrastrukturkomponenten anhand Gleisgraph-Referenzdaten mit metergenauer Präzision (ISDB).....47

Abbildung 14: Dokumentenarchivierung und -auflistung im DMS für Konfigurationsmanagement.....48

Abbildung 15: Wartungskalender-Tabelle: Tätigkeitsbezeichnung, letztes/nächstes Datum, Status.....49

Abbildung 16: Aktivitätenmanagement-Tabelle: Tätigkeitslebenszyklus, Planung, Zuordnung, Status.....49

Abbildung 17: Kalenderansicht des Aktivitätenmanagements der ISDB mit Darstellung geplanter und laufender Infrastrukturmaßnahmen.....50

Abbildung 18: Lebenszyklus-Management – Aktuelle Status, Gültigkeitsdaten und Deaktivierungsfunktion 51

Abbildung 19: Protokoll- und Lebenszyklusübersicht – Historische Verarbeitungsdaten mit Audit-Tracking 51

Abbildung 20: Zielarchitektur WebGIS ISDB – Entkoppelte SPA mit REST-API nach OpenAPI Standard (vgl. Kapitel 2.4.7.3) 54

Abbildung 21: Übergangsarchitektur - Backend Parallelbetrieb 57

Abbildung 22: Swagger Dokumentation der ISDB 69

Abbildung 23: Projektstruktur und Abhängigkeitsdefinition des Vue.js Frontends 71

Abbildung 24: ReadOnlyMap-Komponente – Wiederverwendbarer UI-Baustein zur Visualisierung von Element-Features 73

Abbildung 25: FeatureMap – Komponente zur interaktiven Bearbeitung von GIS-Feature-Geometrien 74

Abbildung 26: Frontend Layer-Konfiguration – Wiederverwendbare XYZ-Tile, WMS und REST-GeoJSON-Integration 76

Abbildung 27: Projektmanagement-Prozess der ISDB-Modernisierung 80

Abbildung 28: ISDB-Zielarchitektur – API-First-Design mit entkopplter Spring Boot / Vue.js Architektur und paralleler Struts-Migration 82

Abbildung 29: Struts Startseite - Performance 83

Abbildung 30: Vue.js Startseite – Performance 83

Abbildung 31: Struts Kartenladezeit 83

Abbildung 32: Vue.js Kartenladezeit inkl. FeatureCollections 84

Abbildung 33: Struts API-Response von Stationszugang 84

Abbildung 34: Vue.js API-Response von Stationszugang 84

Abbildung 35: Product Increment (PI) Planning – Digital Infra Release Train 86

Tabellenverzeichnis

Tabelle 1: Top 10 Java Webframeworks Vergleich (Quelle: GeeksforGeeks 2025) 22

Tabelle 2: Überblick ausgewählter Single-Page-Application-Frameworks 24

Tabelle 3: Vergleich React 27

Tabelle 4: Vergleich Angular 28

Tabelle 5: Vergleich Vue.js 29

Tabelle 6: Vergleich Svelte 30

Tabelle 7: Vergleichstabelle - Fokus Data Bindung & Datenbank nahe Nutzung 31

Tabelle 8: Kernmerkmale der API-Spezifikation 33

Tabelle 9: API-Entwicklungsansätze im Vergleich (Wagner J. 2021) 33

Tabelle 10: API-Spezifikationssprachen im Vergleich 34

Tabelle 11: Strategische Vorteile im Vergleich 58

Tabelle 12: Vergleichende Analyse der API-Designprinzipien 59

Tabelle 13: Struktur des src-Verzeichnisses der Vue-Frontend-Anwendung 71

Tabelle 14: KPI-Entwicklung der ISDB-Modernisierung 85

Kurzfassung

Diese Masterarbeit befasst sich mit der Modernisierung einer bestehenden WebGIS-Anwendung im Kontext öffentlicher Infrastruktursysteme. Gezeigt am Beispiel der Infrastruktur-Datenbank der Wiener Linien (ISDB) wird untersucht, wie ein monolithisches, in die Jahre gekommenes Frontend durch eine moderne, modulare Benutzeroberfläche (UI) ersetzt werden kann.

Die ISDB ist das zentrale System zur Erfassung, Verwaltung und Visualisierung von Geodaten gleisbezogener Infrastrukturobjekte und dient als wesentliche Entscheidungsgrundlage für Inspektion, Planung und Dokumentation infrastruktureller Maßnahmen.

Ziel der Arbeit ist es, die Usability, Wartbarkeit und Skalierbarkeit der Webanwendung signifikant zu verbessern, ohne funktionale oder performancebezogene Einschränkungen zu riskieren. Methodisch basiert die Arbeit auf einem iterativen Prototyping-Ansatz, bei dem zentrale Komponenten des bestehendes sukzessive in eine neue Architektur überführt werden. Dabei wird besonderes Augenmerk auf eine intuitive Benutzerführung, die Integration moderner Webtechnologien sowie die Anbindung an bestehende REST-APIs gelegt.

Die Arbeit liefert praxisnahe Erkenntnisse zur Migration von Legacy-Systemen im Rahmen eines großbetrieblichen Infrastrukturdatenmanagements und zeigt auf, welche Herausforderungen und Chancen mit der Digitalisierung öffentlicher Geoinformationssysteme einhergehen. Die Ergebnisse bieten eine übertragbare Grundlage für vergleichbare Modernisierungsprojekte im Bereich technischer Infrastrukturen.

Abstract

This master's thesis addresses the modernization of an existing WebGIS application within the context of public infrastructure systems. Using the Infrastructure Database of Wiener Linien (ISDB) as a case study, it explores how a monolithic and outdated frontend can be replaced by a modern, modular user interface (UI).

The ISDB serves as the central platform for collecting, managing and visualizing geospatial data related to rail infrastructure assets, providing a critical foundation for inspection, planning, and documentation activities.

The primary objective is to significantly enhance the usability, maintainability, and scalability of the web application while ensuring that no functional or performance-related constraints are introduced.

Methodologically, the thesis adopts an iterative prototyping approach, gradually migrating core components of the legacy system to a new architectural framework. Particular emphasis is placed on intuitive UI design, the use of state-of-the-art web technologies, and the efficient integration of existing REST APIs.

In addition, the thesis offers practical insights into legacy system migration in large-scale infrastructure data management and analyzes the challenges and opportunities associated with the digital transformation of public geoinformation systems. The resulting framework provides a transferable foundation for similar modernization initiatives in the field of technical infrastructure.

1 Einleitung

In der heutigen digitalen Transformation sind die Erfassung, Analyse und Visualisierung raumbezogener Daten aus dem sowohl öffentlichen als auch betrieblichen Umfeld nicht mehr wegzudenken. Geoinformationssystem (GIS) hat sich dabei längst als unverzichtbares Werkzeug etabliert, das komplexe Infrastruktur- und Geodaten nicht nur strukturiert und analysiert, sondern auch intuitiv und verständlich auf digitalen Karten visualisiert – und das auch für nicht fachkundige AnwenderInnen (de Lange, 2020).

Besonders im technischen Infrastrukturmanagement bildet GIS die Grundlage für eine präzise Planung, effiziente Instandhaltung und nachhaltige Optimieren von Anlagen, Netzen sowie Betriebsabläufen. Die Kombination von räumlichen und Fachdaten sowie deren visuelle Aufbereitung schafft Transparenz, macht räumliche Lage und Abhängigkeiten sichtbar und unterstützt fundierte Entscheidungen.

Weit verbreitete Form solcher Darstellungsart ist WebGIS, das GIS-Funktionalitäten mit moderner Webtechnologie verbindet und so eine plattformunabhängige, browserbasierte Nutzung ermöglicht (Li & Li, 2022).

Obwohl WebGIS-Systeme in vielen Organisationen – etwa bei den Wiener Linien – seit Jahren zum Alltag gehören, zeigt sich zunehmend, dass viele solcher Anwendungen technologisch überholt sind. Veraltete Benutzeroberfläche, eingeschränkte Erweiterbarkeit sowie Schnittstellen machen eine grundlegende Modernisierung notwendig, um heutigen Anforderungen an Interaktivität, Systemintegration und Benutzerfreundlichkeit gerecht zu werden (de Lange, 2022).

1.1 Ausgangslage und Motivation

Die Wiener Linien als zentrales Verkehrsunternehmen der österreichischen Hauptstadt spielen eine wesentliche Rolle in der täglichen Mobilität tausender Menschen. Im Laufe der Jahre wurde eine umfassende Datenbasis aufgebaut, deren zentrale Grundlage die seit 2004 im Einsatz befindliche WebGIS-Anwendung ISDB (Infrastruktur-Datenbank der Wiener Linien) bildet. Diese enthält geographische Informationen zu zahlreichen Elementen des Verkehrsnetzes – von U-Bahn-Linien über Straßenbahnen und Busse bis hin zu Haltestellen, Gleisen und Signalanlagen. Die ISDB verknüpft eine relationale Datenbank zur Speicherung infrastruktureller Informationen mit einem Geoinformationssystem (GIS), über das eine räumliche Verortung ermöglicht wird (Ott, 2012). Die Anwendung basiert auf dem inzwischen veralteten Java-Framework Struts, das heute als schwer wartbar gilt und die digitale Weiterentwicklung erheblich erschwert (Wolfart et al., 2021).

Die zunehmende Digitalisierung von Produkten und Prozessen stellt Organisationen auch in den kommenden Jahren vor große Herausforderungen. Digitalprojekte – egal in welcher Branche und welcher Typ – sind Vorhaben mit einer Losgröße 1 und damit höheren Risiken unterworfen als eine Kleinserien- oder gar eine Massenproduktion (Woehe, 2021). Die stetige Einführung neuer Technologien, der permanent steigende Innovations-druck und die zunehmende Komplexität führen dazu, dass Krisen in Digitalprojekten häufig keine Ausnahmen, sondern systemimmanente Begleiterscheinungen sind. Ein wichtiges Ziel der Digitalen Transformation muss es unbedingt sein, so wenig wie möglich Krisenprojekte zu haben und so früh als möglich Krisensituationen zu erkennen und Gegenmaßnahmen durchzuführen (Woehe, 2021).

Diese Erkenntnisse lassen sich auch auf die aktuelle Situation bei der Wiener Linien übertragen: Die ISDB ist heute noch eine zentrale WebGIS-Anwendung zur Erfassung, Visualisierung und Verwaltung gleisbezogener, georeferenzierter Infrastrukturobjekte. Obwohl die Anwendung seit ihrer Einführung kontinuierlich In-House weiterentwickelt wurde, basiert jedoch nach wie vor auf einem monolithischen Legacy-Frontend, das heutigen Anforderungen an Benutzerfreundlichkeit, Wartbarkeit, Modularität und Performance nicht mehr gerecht wird.

Die bestehende Architektur erschwert insbesondere die flexible Einbindung unter anderem neuer GIS-Funktionalitäten, wie dynamische Layer, Filterfunktionen oder Geoservices, sowie die Integration moderner Webstandards und responsiver UI-Komponenten. Vor dem Hintergrund steigender Nutzeranforderungen, zunehmender Datenkomplexität und des Bedarfs an aktuellen Sicherheitsstandards ergibt sich ein klarer Handlungsbedarf. Die technische und gestalterische Erneuerung der Benutzeroberfläche der ISDB ist somit notwendig, um den langfristigen Betrieb sicherzustellen und zukunftsorientierte Weiterentwicklung zu ermöglichen.

1.2 Zielsetzung und Aufgabenstellung

Ziel dieser Masterarbeit ist die Entwicklung eines methodisch fundierten Konzepts sowie die prototypische Umsetzung einer modernen WebGIS-Oberfläche mit REST-API Anbindung (Fielding, 2000). Diese soll den heutigen Anforderungen an Benutzerfreundlichkeit, Skalierbarkeit und Wartbarkeit gerecht werden. Im Fokus steht die Migration von einer monolithischen, veralteten Benutzeroberfläche hin zu einer modularen, komponentenbasierten Frontend-Architektur.

Neben den technischen Migrationsstrategien werden auch zentrale Begrifflichkeiten sowie deren Bedeutung im praktischen Einsatzkontext von GIS-Systemen beleuchtet. Auf diese Weise soll ein umfassendes Verständnis für die Herausforderungen und Gestaltungsmöglichkeiten moderner WebGIS-Anwendungen entwickelt werden.

Folgende Schwerpunkte stehen dabei im Zentrum:

- *Systemanalyse und Anforderungen:*

  Analyse der Legacy-Anwendung sowie technische und funktionale Zieldefinition

- *Technologieauswahl und API-Konzept:*

  Auswahl des Frontend-Frameworks und Entwurf eines REST-API Design

- *Proof of Concept:*

  Prototypische Umsetzung mit Fokus auf Usability und GIS-Funktionalität

1.3 Stand der Forschung

Geoinformationssysteme (GIS) haben sich in den vergangenen Jahrzehnten zu zentralen Instrumenten für die Planung, Analyse und Entscheidungsgrundlage in verschiedenen Fachbereichen entwickelt. Besonders durch die Integration moderner Webtechnologien konnten WebGIS-Anwendungen eine niederschwellige, plattformunabhängige und intuitive Nutzung raumbezogener Daten ermöglichen, auch für NutzerInnen ohne vertiefte GIS-Kenntnisse (Rahn, 2015).

Die Relevanz von WebGIS-Systemen zeigt sich vor allem im Bereich öffentlicher Infrastrukturen. Studien und Fachliteratur belegen, dass Institutionen wie Verkehrsunternehmen zunehmend auf digitale Systeme zur Erfassung, Visualisierung und Analyse komplexer Infrastrukturdaten angewiesen sind. Gerade im urbanen Raum, wo hohe Komplexität und Dynamik herrschen, sind leistungsfähige und benutzerfreundliche WebGIS-Oberflächen eine unverzichtbare Grundlage für Betrieb und Weiterentwicklung (Ott et al., 2012).

In Bezug auf bestehende WebGIS-Systeme zeigt sich, dass viele Anwendungen, die vor Jahren auf der Basis damaliger Technologien entwickelt wurden, heute technologisch überholt sind. Fachartikel wie Wolfart et al. (2021) verweisen darauf, dass monolithische Architekturen zunehmende durch komponentenbasierte Systeme ersetzt werden müssen, um Flexibilität, Wartbarkeit und Skalierbarkeit zu gewährleisten. Veraltete Technologien wie klassische Java-Frameworks (z.B. Struts) schränken dabei nicht nur die Benutzerfreundlichkeit ein, sondern erschweren auch die Integration neuer Funktionalitäten wie dynamischer Kartenlayer, moderner UI-Komponenten und Sicherheitsstandards (Wolfart et al., 2021).

Darüber hinaus weisen Legacy-Systeme häufig strukturelle Defizite hinsichtlich Skalierbarkeit, Sicherheit und Integrationsfähigkeit auf, wodurch sie den Anforderungen moderner Anwendungen nur eingeschränkt gerecht werden (Ogunwole et al., 2023). Diese Defizite verstärken sich insbesondere in komplexen Systemlandschaften, in denen gewachsene Abhängigkeiten und fehlende Standardisierung die Weiterentwicklung zusätzlich erschweren.

Aktuelle Ansätze zur Modernisierung von WebGIS-Anwendungen beschäftigen sich vor allem mit der Migration von Legacy-Systemen hin zu modularen, API-getriebenen Webarchitekturen. Die Forschungsliteratur betont, dass neben technischen Aspekten auch Usability, Performance und eine hohe Skalierbarkeit zentrale Anforderungen an moderne WebGIS-Systeme darstellen (Woehe & Kurz, 2021). Während bestehende Arbeiten primär allgemeine Modernisierungsstrategien und architekturelle Konzepte adressieren (Ogunwole et al., 2023; Assunção et al., 2025), bleibt die konkrete Umsetzung dieser Ansätze in realen, produktiven Systemumgebungen häufig unzureichend betrachtet.

Praxisorientierte und empirische Studien zeigen zudem, dass die Modernisierung von Legacy-Systemen nicht ausschließlich ein technisches Problem darstellt, sondern maßgeblich durch organisatorische, strategische und infrastrukturelle Rahmenbedingungen beeinflusst wird (Irani et al., 2023). Insbesondere im öffentlichen Sektor verdeutlichen Fallstudien, dass Legacy-Systeme durch ihre Komplexität, mangelnde Interoperabilität sowie starke Abhängigkeiten erhebliche Hindernisse für digitale Transformationsprojekte darstellen. Die Migration wird dabei nicht nur durch technische Herausforderungen geprägt, sondern auch durch bestehende Geschäftsprozesse, institutionelle Strukturen und organisatorische Entscheidungswege erschwert (Irani et al., 2023).

Trotz dieser Erkenntnisse gibt es bisher nur eine begrenzte Anzahl an Studien, die sich spezifisch mit der Migration veralteter WebGIS-Frontends unter Beibehaltung komplexer Infrastrukturdaten und Integration moderner REST-API-Architekturen beschäftigen. Der Schwerpunkt vieler Arbeiten wie – etwa in den Bereichen technologischer Modernisierungsprojekte, wissenschaftlicher Studien zur IT-Systemerneuerung oder öffentlicher IT-Vergaben – liegt entweder auf allgemeinen Modernisierungsstrategien oder auf der Entwicklung neuer WebGIS-Systeme, ohne den komplexen Kontext bestehender Systeminfrastrukturen vollständig zu berücksichtigen.
Eine integrierte Betrachtung technologischer, organisatorischer und systemarchitektonischer Aspekte im Kontext konkreter Anwendungssysteme wie WebGIS bleibt bislang weitgehend unzureichend.

Die vorliegende Masterarbeit setzt hier an und schließt eine bestehende Forschungslücke, indem sie sich mit der Modernisierung eines produktiv eingesetzten WebGIS-Systems – konkret der Infrastruktur-Datenbank (ISDB) der Wiener Linien – befasst. Ziel ist es, ein methodisch fundiertes Konzept zu entwickeln, das den Übergang von einem monolithischen zu einem modularen, modernen WebGIS ermöglicht und somit einen Beitrag zur nachhaltigen Weiterentwicklung digitaler Infrastruktursysteme leistet.

1.4 Forschungsfragen

Die zentrale Forschungsfrage, die im Rahmen dieser Masterarbeit untersucht wird, lautet:

„*Wie kann eine Legacy-Anwendung erfolgreich in eine moderne, wartbare und performante Webanwendung überführt werden?“*

Um die Forschungsfrage gezielt bearbeiten zu können, wurden ergänzend folgende Arbeitsfragen entwickelt:

1. Welche Kriterien sind bei der Auswahl eines geeigneten Frontend-Frameworks im Kontext von WebGIS-Anwendungen entscheidend?
2. Wie lässt sich eine performante und flexible REST-API-Struktur für GIS-Daten gestalten?
3. Welche Herausforderungen ergeben sich bei der Umstellung im Hinblick auf Legacy-WebGIS-Anwendung, Datenstruktur und Systemintegration?
4. Welche Lösungsansätze sowie schrittweisen Implementierungsstrategien ermöglichen die erfolgreiche Migration zu einer entkoppelten WebGIS-Architektur?

1.5 Methodische Vorgehensweise

Die Methodik dieser Masterarbeit basiert auf dem iterativen Prototyping Ansatz (Hermans, 2023). Dabei wird der Übergang von einem monolithischen Legacy-Frontend hin zu einer modernen, komponentenbasierten WebGIS-Oberfläche Schritt für Schritt vollzogen. Besonders relevant ist dabei das *Evolutionary Prototyping*, das darauf abzielt, ein bereits entwickeltes Softwaresystem schrittweise an sich ändernde Anforderungen anzupassen (Studer, 2013). Diese Form der inkrementellen Systementwicklung eignet sich besonders gut für die Modernisierung komplexer Anwendung wieder ISDB. Zudem entfällt damit auch die Wartungsphase im klassischen Software-Life-Cycle-Modell (SLCM) – (Studer, 2013), was den iterativen Entwicklungsprozess zusätzlich beschleunigt.

Zu Beginn erfolgt eine umfassende Systemanalyse der bestehenden Infrastruktur-Datenbank der Wiener Linien (ISDB). In diesem Schritt werden sowohl die technische Struktur als auch die fachlichen Anforderungen analysiert und dokumentiert. Dabei wird besonderes Augenmerk der bestehenden Schwachstellen sowie Verbesserungspotenziale gelegt. Darauf aufbauend erfolgt die Erarbeitung einer Zieldefinition, die sowohl technische als auch funktionale Anforderungen an das neue System beschreibt. Diese Anforderungen bilden anschließend die Grundlage für Technologieauswahl, insbesondere hinsichtlich des Frontend-Frameworks sowie des Designs der REST-API.

In weiterer Folge wird auf Basis der ausgewählten Technologien ein erster Prototyp in Form von *Proof of Concept (PoC)* entwickelt, die zentralen Funktionalitäten und Designelemente des geplanten Systems abbildet. Dieser Prototyp dient dazu, frühzeitig Usability-Aspekte sowie technische Machbarkeit zu evaluieren. Feedback aus dieser ersten Phase fließt direkt in den weiteren Entwicklungsprozess.

In einem iterativen Prozess werden weitere Funktionen schrittweise implementiert und regelmäßig auf ihre technische und funktionale Erfüllung der Anforderungen geprüft (Hermans, 2023). Dabei spielen kontinuierliche Validierungsschritte und Testphasen eine

zentrale Rolle, um sicherzustellen, dass alle Entwicklungsschritte zielführend sind und die definierten Anforderungen erfolgreich zu erfüllen.

Abschließend erfolgt eine umfassende Evaluierung des fertigen PoC. Hierbei stehen insbesondere die Usability, Performance, Skalierbarkeit und die Möglichkeit der Integration in bestehende Systeme im Fokus. Die Ergebnisse werden empirisch analysiert und bewertet, um abschließend fundierte Empfehlung für die weitere Umsetzung und Übertragbarkeit des Konzepts auf ähnliche Systeme geben zu können.

Dieser iterative Ansatz erlaubt somit eine zielgerichtete, flexible Entwicklung und reduziert Risiken durch laufende Validierung.

1.6 Struktur der Arbeit

Die vorliegende Masterarbeit ist in sechs Hauptkapitel gegliedert, die systematisch von den theoretischen Grundlagen über die strategische Planung bis hin zur praktischen Umsetzung und abschließenden Bewertung der Modernisierung der ISDB-Webanwendung führen.

Kapitel 1 führt in die Arbeit ein und beschreibt die Ausgangssituation, Motivation und Zielsetzung der Untersuchung. Darauf aufbauend werden der Stand der Forschung, die Forschungsfragen sowie die methodische Vorgehensweise dargestellt. Dieses Kapitel bildet den konzeptionellen Rahmen der Arbeit und definiert die zentralen Fragestellungen, die in den folgenden Kapiteln aufgegriffen und beantwortet werden.

Kapitel 2 behandelt die theoretischen und technologischen Grundlagen. Es werden zentrale Konzepte der Geoinformatik, WebGIS und Webmapping erläutert sowie Begriff und Charakteristika von Legacy-Systemen eingeordnet. Darüber hinaus werden moderne Softwarearchitekturen, Interoperabilitätskriterien und relevante Webtechnologien vorgestellt. Ein besonderer Fokus liegt auf der vergleichenden Analyse moderner SPA-Frontend-Frameworks sowie auf API-Konzepten, die für die späteren Architekturentscheidungen maßgeblich sind.

Kapitel 3 beschreibt die organisatorische und technische Ausgangssituation der Infrastruktur-Datenbank der Wiener Linien. Neben der historischen Entwicklung der ISDB werden deren Rolle innerhalb der bestehenden Systemlandschaft sowie das fachliche Datenmodell dargestellt. Dieses Kapitel fungiert als Brücke zwischen den theoretischen Grundlagen und der konkreten Fallstudie.

Kapitel 4 beschreibt die strategische Planung und Zielarchitektur der Modernisierung. Auf Grundlage der Ist-Analyse (Kapitel 3) sowie der technologischen Grundlagen (Kapitel 2) werden zentrale architektonische Entscheidungen hergeleitet. Dazu zählen insbesondere eine entkoppelte SPA-Architektur mit einem Vue.js-basierten Frontend, eine REST-API auf Basis der OpenAPI-Spezifikation sowie eine Übergangsstrategie für den parallelen Betrieb von Legacy-System und Neusystem. Das Kapitel bildet damit die konzeptionelle Grundlage für die praktische Umsetzung.

Kapitel 5 dokumentiert die praktische Umsetzung der Modernisierung und Migration der ISDB-Webanwendung. Es werden der methodische Umsetzungsansatz, die Implementierung der REST-API nach OpenAPI Standard, die Frontend-Architektur mit Vue.js sowie die Integration von GIS-Komponenten detailliert dargestellt. Ergänzend werden zentrale Herausforderungen des Parallelbetriebs von Legacy- und Neusystem analysiert und die angewandten Lösungsansätze erläutert.

Kapitel 6 fasst die Arbeit zusammen und stellt die gewonnenen Erkenntnisse im Hinblick auf die Forschungsfragen dar. Abschließend werden Schlussfolgerungen gezogen, Handlungsempfehlungen formuliert und ein Ausblick auf mögliche Weiterentwicklungen sowie zukünftigen Forschungsbedarf gegeben.

2 Grundlagen der Geoinformatik und Softwaretechnologien

Das folgende Hauptkapitel gibt einen Überblick über die theoretischen Grundlagen dieser Masterarbeit. Zunächst werden zentrale Begriffe und Konzepte im Bereich des Geoinformationssystems (GIS) erläutert, inklusive Definition, Aufbau und Rolle im Infrastrukturmanagement. Anschließend folgt eine Einführung in die wichtigsten Webtechnologien, die für die Entwicklung und Modernisierung von WebGIS-Anwendungen relevant sind. Ergänzend wird auf offene Standards, Interoperabilität sowie Open Source Ansätze im Bereich GIS eingegangen. Diese Grundlagen bilden das notwendige Fundament für das Verständnis der technischen und konzeptionellen Entscheidungen, die im weiteren Verlauf dieser Arbeit getroffen werden.

2.1 Grundlagen der Geoinformatik

Das folgende Kapitel soll die grundlegenden Konzepte und Definitionen von Geoinformationssystem (GIS) darstellen.

2.1.1 Grundlagen und Definition von GIS

GIS werden in zahlreichen Anwendungsbereichen eingesetzt, darunter Geographie, Umweltforschung, Stadtplanung, Ressourcenmanagement und viele weitere Felder, in denen die Analyse und Visualisierung von Geodaten eine zentrale Rolle spielt.

> *Ein Geoinformationssystem ist ein rechnergestütztes System, das aus Hardware, Software, Daten und den Anwendungen besteht. Mit ihm können raumbezogene Daten digital erfasst, gespeichert, verwaltet, aktualisiert, analysiert und modelliert sowie alphanumerisch und graphisch präsentiert werden. (de Lange 2020)*
>
> *Die Besonderheit eines GIS liegt darin, dass die verarbeiteten Daten immer einen Raumbezug aufweisen, also mit einer bestimmten Position auf der Erdoberfläche verknüpft wird. (Bill 2016)*

2.1.2 Komponenten und Begriff von GIS

Ein Informationssystem – wie auch ein GIS – kann grundsätzlich auf zwei Arten betrachtet werden: ***strukturell (HSDA-Modell)***, also hinsichtlich seiner Komponenten, und ***funktional (EVAP-Modell)***, bezogen auf die Aufgaben jeweils gemäß dem Viel-Komponenten-Modell (de Lange, 2020).

Das HSDA-Modell definiert folgende strukturellen Komponenten:

- **Hardware**: Computersysteme einschl. Prozessor, Speichermedien, Peripheriegeräte und Vernetzung
- **Software**: Programmsysteme einschl. Softwarewerkzeuge zur Erfassung, Verwaltung, Analyse und Präsentation der Informationen
- **Daten**: quantitative und qualitative Informationen, die zusammen einen (fachbezogenen) Ausschnitt der realen Welt darstellen
- **Anwender**: Benutzer mit ihren Anforderungen und Fragestellungen bzw. Anwendungen und Einsatzmöglichkeiten

Das EVAP-Modell definiert folgende funktionalen Komponenten:

- **Erfassung**: Daten- oder Informationserfassung und -speicherung (d.h. Input)
- **Verwaltung**: Datenverwaltung (d.h. Management)
- **Analyse**: Datenauswertung (d.h. Analysis)
- **Präsentation**: Wiedergabe der Informationen (d.h. Output bzw. Präsentation)

Zusammenfassend bilden das HSDA-Modell (Hardware, Software, Daten, Anwender*innen) und das EVAP-Modell (Erfassung, Verwaltung, Analyse, Präsentation) die strukturellen und funktionalen Perspektiven eines GIS. Beide Modellen zusammen bieten ein vollständiges Bild des Aufbaus und der Aufgaben allgemeiner Geoinformationssysteme.

Die Entwicklung von Geoinformationssystemen begann in den 1960er Jahren. Eines der ersten Systeme war das *Canada Geographic Information System (CGIS)*, das von Roger Tomlinson für die Analyse und Verwaltung der umfangreichen Daten des *Canada Land Inventory* entwickelt wurde.

Pionierarbeit leisteten auch das *Harvard Laboratory for Computer Graphics and Spatial Analysis* sowie später das *ArcInfo von ESRI* und das *GRASS GIS*. Seit den 1990er Jahren prägten technische Fortschritte die weitere Entwicklung, sodass GIS-Systeme heute in vielen Bereichen zu den wichtigsten Standardwerkzeuge für die Verwaltung und Planung geworden sind (de Lange, 2020).

2.2 WebGIS vs. Webmapping

Das Kapitel untersucht webbasierte Geoinformationssysteme anhand ihrer begrifflichen Grundlagen, historischen Entwicklung und aktueller Trends in der Geoinformatik. Ziel ist es, ein vertieftes Verständnis für WebGIS- und Webmapping-Konzepte zu vermitteln sowie deren Einordnung im technologischen und fachlichen Kontext zu ermöglichen. Aufbauend darauf werden die wesentlichen Unterschiede, Entwicklungslinien und aktuellen Ausprägungen webbasierter GIS-Anwendungen erörtert.

2.2.1 Grundlegende Begriffe und Definitionen

Mit der nahezu zeitgleichen Entwicklung des *World Wide Web (WWW)* und des Internets als Plattform war es nur eine Frage der Zeit, bis sich die Bereiche GIS und Internet zu einem neuen Werkzeug der Geoinformatik fusionierten (de Lange, 2020). Diese Entwicklung führte zu einer stetigen und nachhaltigen Etablierung von WebGIS in der Welt der Datenvisualisierung. Darüber hinaus können zahlreiche Funktionalitäten von Desktop-Geo-Informationssystemen mittels Webtechnologien realisiert und somit einer bereiten Nutzerzahl zur Verfügung gestellt werden (Seip et al., 2017).

WebGIS ist alles, das sich Internet-Technologien bedient und mit dem sich räumliche Informationen darstellen, verteilen und bearbeiten lassen (Seip et al., 2017).

Heute zählt WebGIS zu den gängigsten Methoden für die Bereitstellung, Analyse und Darstellung raumbezogener Daten und ist aus modernen Planungsprozessen, Umweltanalysen sowie der öffentlichen Geodatenbereitstellung nicht mehr wegzudenken.

In der wissenschaftlichen Literatur finden sich zahlreiche Begriffe, die teils synonym, teils jedoch mit unterschiedlichen Schwerpunktsetzungen verwendet werden. Neben dem Terminus *Web-GIS* treten auch alternative Bezeichnungen wie *Online-GIS*, *Internet-GIS, Net-GIS oder Distributed GIS* auf. Ebenso werden die Begriffe *Web-Mapping* und *Internet-Mapping* genutzt, wobei diese häufigen Systeme mit abweichenden funktionalen Schwerpunkten bezeichnen (Behncke et al. 2009; Rudert und Pundt 2008).

Zu Web-Mapping werden demnach Systeme gezählt, die eine einfache Kartendarstellung mit Funktionen zum Zoomen und Verschieben unterstützen und in denen Layer auswählbar sind.

Zu Web-GIS werden Lösungen gezählt, die über eine Sachdatenbank im Hintergrund verfügen, auf der Nutzer eigenständig, themenbezogene Anfragen stellen können und mit denen Such-, Berechnungs-, Buffer-, und ähnliche Analysefunktionen unterstützt werden (Seip et al., 2017).

Im Rahmen dieser Arbeit werden die Begriffe „WebGIS“ und „WebMapping“ genutzt, um die in der Fachliteratur etablierte Unterscheidung zwischen reinen webbasierten Kartendiensten und Systemen mit erweiterten GIS-Funktionalitäten nachvollziehbar zu

machen. Für die Modernisierung der Webanwendung „Infrastruktur Datenbank“ ist jedoch eine Systemarchitektur erforderlich, die sowohl komplexe Sachdatenverarbeitung als auch fortgeschrittene Analyse- und Abfragefunktionen ermöglicht. Daher steht im weiteren Verlauf der Begriff „WebGIS“ im Mittelpunkt der Betrachtung, während klassische Webmapping-Ansätze, die primär auf kartografische Visualisierung und grundlegende Interaktion ausgerichtet sind, nicht vertieft behandelt werden.

2.2.2 Historische Entwicklung von WebGIS

Im Jahr 1993 wurde vom vom Xerox Palo Alo Research Center (PARC) das erste kartengestützte Online-System ***Xerox PARC Map Viewer*** ins Internet gestellt (siehe Abb. 1). Dieser konnte Vektordaten von verschiedenen Datenquellen in Hyper Text Markup Language (HTML) eingebettete Rasterbilder in verschiedenen thematischen Ebenen darstellen (Seip et al., 2017), was eines der frühsten interaktiven Online-Kartensysteme gilt und wesentlichen Impulse für die heutige WebGIS-Technologie lieferte. In vielen weiteren Ländern wurden zunächst Metainformationssysteme (MISe) aufgebaut, um die vorhandenen Datenbestände zunächst zu beschreiben, durchsuchbar zu machen und zu verlinken (Korduan, 2004). Erste kommerzielle Dienstleistungsangebote mit kartographischem Material im Internet waren die System *MapQuest* von *Geosystems Global* und *MapBlast* von *Vicinity* (Dickmann, 2001).

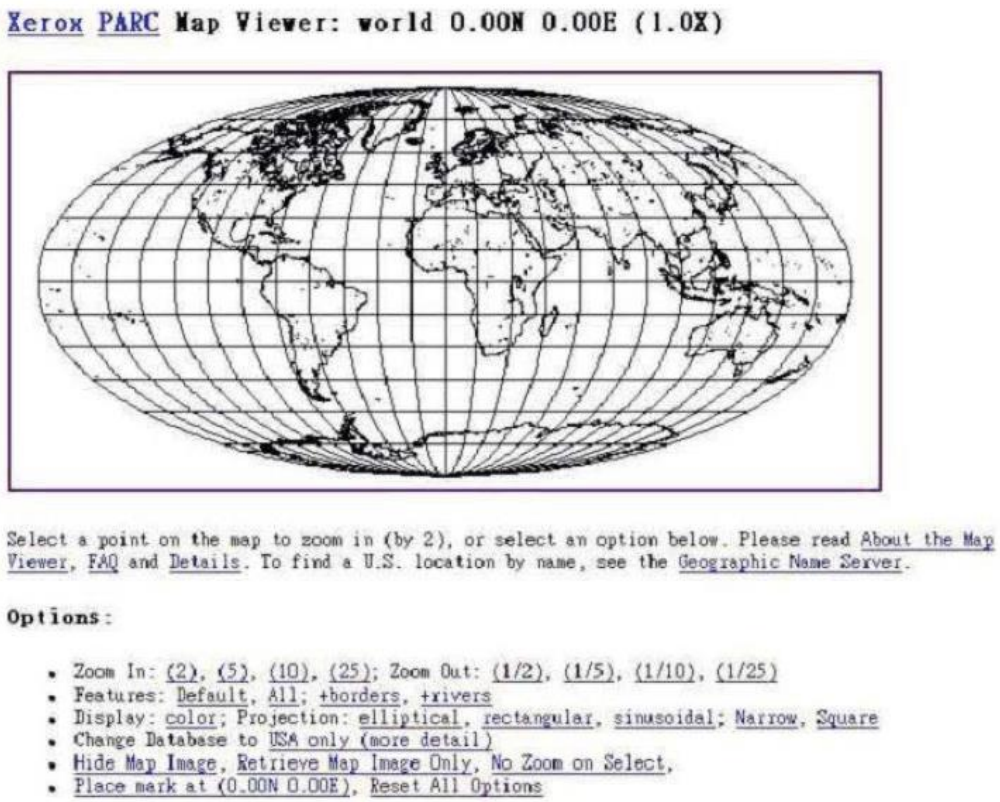


Abbildung 1: Der Xerox PARC Map Viewer aus dem Jahr 1993

In den 2000er-Jahren gewannen kollaborative Plattformen wie *OpenStreetMap* an Bedeutung. Sie ermöglichten es, Geodaten gemeinschaftlich zu erfassen und frei bereitzustellen und etablierten so neue Formen partizipativer Geodatenproduktion (Haklay & Weber 2008; Veenendaal et al. 2017). Ab den 2010er-Jahren rückten mobile Endgeräte und cloudbasierte Lösungen in den Vordergrund. Cloud-Technologien boten flexible Speicher- und Analysekapazitäten, wodurch Web-GIS skalierbarer und breiter nutzbar wurde (Veenendaal et al. 2017).

2.2.3 Aktuelle Trends in Geoinformatik und WebGIS

Die Geoinformatik und insbesondere WebGIS-Systeme entwickeln zunehmend zu integrativen Analyse- und Entscheidungsplattformen. Vor allem in Verwaltung, Stadtentwicklung und Infrastrukturmanagement, wo sich der Fokus von statischen Karten hin zu datengetriebenen Systemen verschiebt (Scanpoint Geomatics Ltd., 2024). Gleichzeig betont MAPID (2025) einen Paradigmenwechsel hin zu Spatial Intelligence, der die Analyse, Prognose und Automatisierung räumlicher Daten in Mittelpunkt stellt. Zentrale technologische Treiber sind die kombinierte Nutzung von WebGIS, Cloud Computing und Künstlicher Intelligenz (KI), wobei WebGIS als kollaborative Plattform, KI als analytischer Motor und Cloud Infrastrukturen als Skalierungsbasis fungieren (MAPISD, 2025; Scanpoint Geomatics Ltd., 2024). Ein weiterer wichtiger Trend ist die Verknüpfung fachlicher Attributdaten mit räumlichen Geodaten, die kontextualisierte Analysen und evidenzbasierte Entscheidungsprozesse in Verwaltung, Wirtschaft und Infrastrukturplanung ermöglicht (Scanpoint Geomatics Ltd., 2024). In diesem Zusammenhang gewinnt Location Intelligence für strategische Unternehmensentscheidungen in Bereichen wie Einzelhandel, Finanzwesen und Logistik an Bedeutung (MAPID, 2025). Ergänzend dazu prägt die Integration von Echtzeitdaten aus IoT-Sensorik die Entwicklung hin zu Digital Twins, die Monitoring, Simulation und Steuerung räumlich referenzierter Prozesse in Echtzeit erlauben (MAPID, 2025; Scanpoint Geomatics Ltd., 2024). Die Branche verlangt zunehmend interdisziplinäre Kompetenzen, die GIS-Grundlagen, Programmierung (Python, JavaScript, Java), Datenbankmanagement (SQL, PostGIS) sowie Cloud-Architekturen und API-Design kombinieren.

Zusammenfassend lässt sich feststellen, dass sich die Geoinformatik in eine stärker vernetzte, skalierbare und intelligente Architektur bewegt. Die Synergie von WebGIS, KI und Cloud-Computing transformiert GIS von einer Visualisierungsplattform zu einem analytischen Ökosystem, das evidenzbasierte, automatisierte und strategisch relevante Entscheidungen im Sinne der Spatial Intelligence ermöglicht.

2.3 Legacy-Systeme und Modernisierung

Legacy-Systeme entstehen häufig durch langfristig gewachsene Daten- und Systemlandschaften, in denen unterschiedliche Erfassungsstandards, Technologien und Nutzungskontexte überlagert sind. Studien zu GIS-fähigen Legacy-Datenbanken zeigen, dass diese historischen Schichten zwar funktional bleiben, jedoch erhebliche Probleme in

Bezug auf Datenqualität, Interoperabilität und Analysefähigkeit verursachen können, wenn sie ungeprüft in moderne Systeme überführt werden (Woywitka & Beaudoin, 2009).

Heutige Modernisierungsstrategien zielen daher auf eine kontrollierte Weiterentwicklung ab, bei der bestehende Strukturen systematisch bereinigt, neu strukturiert und schrittweise an moderne architektonische und methodische Anforderungen angepasst werden.

2.3.1 Begriffe und Einordnung von Legacy-Systemen

Als Legacy-Systeme werden IT-Systeme bezeichnet, die auf zum Zeitpunkt ihrer Entwicklung angemessenen Technologien und Architekturen basieren, deren technische Umsetzung jedoch deutlich von heutigen Entwicklungsstandards abweicht (Annett, 2019). Sie stellen keine fehlerhaften oder minderwertigen Systeme dar, sondern sind vielmehr Ausdruck ihrer Entstehungszeit und ihres ursprünglichen Einsatzkontextes. Charakteristisch ist, dass diese Systeme über Jahre oder Jahrzehnte hinweg stabil betrieben werden und einen wesentlichen Beitrag zur Erfüllung organisatorischer Kernaufgaben leisten, während funktionale Erweiterungen und technologische Anpassungen zunehmend erschwert sind.

Abb.2 veranschaulicht diese Einordnung, indem Legacy-Systeme als Phase im Lebenszyklus von Software dargestellt werden. Nach einer vergleichsweise kurzen Entwicklungs- und Einführungsphase folgt ein oftmals sehr langer Nutzungs- und Wartungszeitraum, in dem der organisatorische Mehrwert primär durch den stabilen Betrieb und nicht durch kontinuierliche Weiterentwicklung entsteht (Annett 2019). In dieser Phase verschiebt sich der Fokus weg von Entwicklung hin zu Wartung, Support und Betriebssicherung, wodurch technologische Erneuerungen häufig aufgeschoben werden.

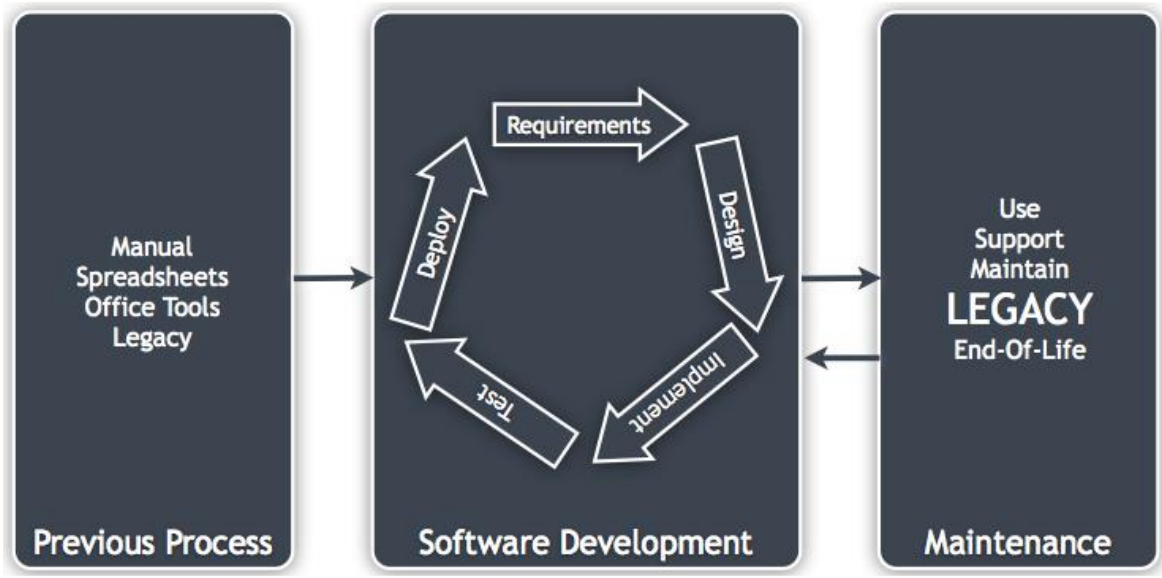


Abbildung 2: Lebenszyklen in Kontext (Annett, R. 2019. S.3)

Legacy Systeme entstehen nicht abrupt, sondern entwickeln sich schrittweise aus ehemals modernen, anforderungsgerechten Anwendungen. Sie sind das Resultat langfristiger Nutzung, organisatorischen Abhängigkeiten sowie ökonomischen und strategischen Entscheidungen, bei denen eine Ablösung oft als zu risikoreich oder kostenintensiv gilt Aus organisatorischer Sicher liegt der Hauptwert eines Systems in seiner stabilen Nutzung, während es aus Entwicklerperspektive (Software Development) häufig nur, als Wartungsobjekt betrachtet wird. Der tatsächliche Nutzen für Organisationen entsteht insbesondere in dieser Nutzungsphase, in der technologische Erneuerung zunehmend in den Hintergrund tritt (Annett, 2019).

Für GIS- und WebGIS Systeme bedeutet dies, dass historisch gewachsene Datenmodelle und Logiken trotz begrenzter Interoperabilität mit modernen Architekturen weiterhin produktiv genutzt werden. Die Herausforderung liegt im Ausgleich zwischen fachlicher Stabilität und technischer Erneuerung (Annett, 2019). Die Lifecycle-Perspektive (siehe Abb. 2) verdeutlicht, dass sich mit wachsender Systemreife der Fokus von Weiterentwicklung zu Wartung und Betriebssicherung verlagert, wodurch technologische Modernisierung häufig aufgeschoben wird (Annett, 2019). Ein Bestandssystem bleibt aktiv gepflegt und anpassbar, während eine Altlast durch Überalterung, Wartungsdefizite und technische Schulden geprägt ist. Legacy steht dabei nicht für schlechte Qualität, sondern für einen Reifegrad im Softwarelebenszyklus, dessen Bewertung wesentlich von Management- und Kostenentscheidungen abhängt (Annett, 2019).

2.3.2 Entscheidung zur Modernisierung

Die Modernisierung eines Legacy-Systems ist dann sinnvoll, wenn betriebliche Risiken und Kosten das Nutzungspotenzial eines Systems zunehmend übersteigen und sich diese Schieflage nicht mehr durch rein wartende Maßnahmen auffangen lässt – wie z.B. veraltete Schnittstellenstandards, die mit modernen Softwareprodukten nicht mehr kompatible sind. Legacy beschreibt dabei eine Lebenszyklusphase mit hohem operativem Nutzen bei gleichzeitig fortschreitender technologischer Überalterung, nicht per se schlechte Softwarequalität (Annett, 2019).

Die Entscheidung zur Modernisierung wird regelmäßig durch zunehmende Systemausfälle, Performanceprobleme, Inoperabilität und sicherheitsrelevante Vorfälle ausgelöst, insbesondere bei fragilen Systemen, die im Normalbetrieb funktionieren jedoch auf die Änderungen moderner Softwarestandards und Entwicklungstrends unvorhersehbar reagieren. Diese Fragilität führt dazu, dass notwendige Anpassungen vermieden werden, was den technischen Rückstand weiter vergrößert (Annett, 2019).

Ein weiterer Kernfaktor ist die Wartbarkeit: Veraltete oder fehlende Dokumentation, der Verlust impliziten Wissens sowie der Einsatz nicht zeitgemäßer Technologien erhöhen den Aufwand für Betrieb und Weiterentwicklung und verstärken Probleme inkonsistenter, historisch gewachsener Architekturen (Annett, 2019). Gleichzeitig sinken dadurch die Produktivität der Entwicklung und die Änderungsfähigkeit des Softwaresystems deutlich (Annett, 2019). Auch die Integrations- und Erweiterungsfähigkeit beeinflussen die

Modernisierungsentscheidung maßgeblich. Besonders hoch wird der Modernisierungsdruck, wenn Integrations- und Erweiterungsanforderungen steigen, monolithische und stark gekoppelte Legacy-Systeme jedoch nur eingeschränkt interoperabel sind und dadurch die Anbindung neuer Anwendungen, externer Dienste oder cloudbasierter Komponenten erschwert wird.

Studien zur Modernisierung mittels Microservices zeigen, dass monolithische Architekturen insbesondere bei steigenden Integrationsanforderungen zu erheblichen technischen und organisatorischen Einschränkungen führen (Knoche & Hasselbring, 2018). Die fehlende Modularität wirkt sich dabei negativ auf Skalierbarkeit, Wartbarkeit und Innovationsfähigkeit aus (Knoche & Hasselbring, 2018).

Neben technischen Aspekten spielen wirtschaftliche Rahmenbedienungen eine zentrale Rolle. Steigende Betriebs- und Wartungskosten bei gleichzeitig sinkender Änderungsfähigkeit verschlechtern das Kosten-Nutzen-Verhältnis eines Systems deutlich. Zusätzlich verlängern sich Entwicklungs- und Release-Zyklen, was die Time-to-Market negativ beeinflusst und strategische Unternehmensziele längerfristig behindert (Knoche & Hasselbring, 2018). Dadurch entsteht eine strategische Lücke, da bestehende Systeme neue digitale Services nicht mehr ausreichend unterstützen. Besonders bei monolithischen Legacy-Systemen verschärft der technologische Alterungsprozess das Problem, da qualifiziertes Fachpersonal zunehmend schwer zu finden ist.

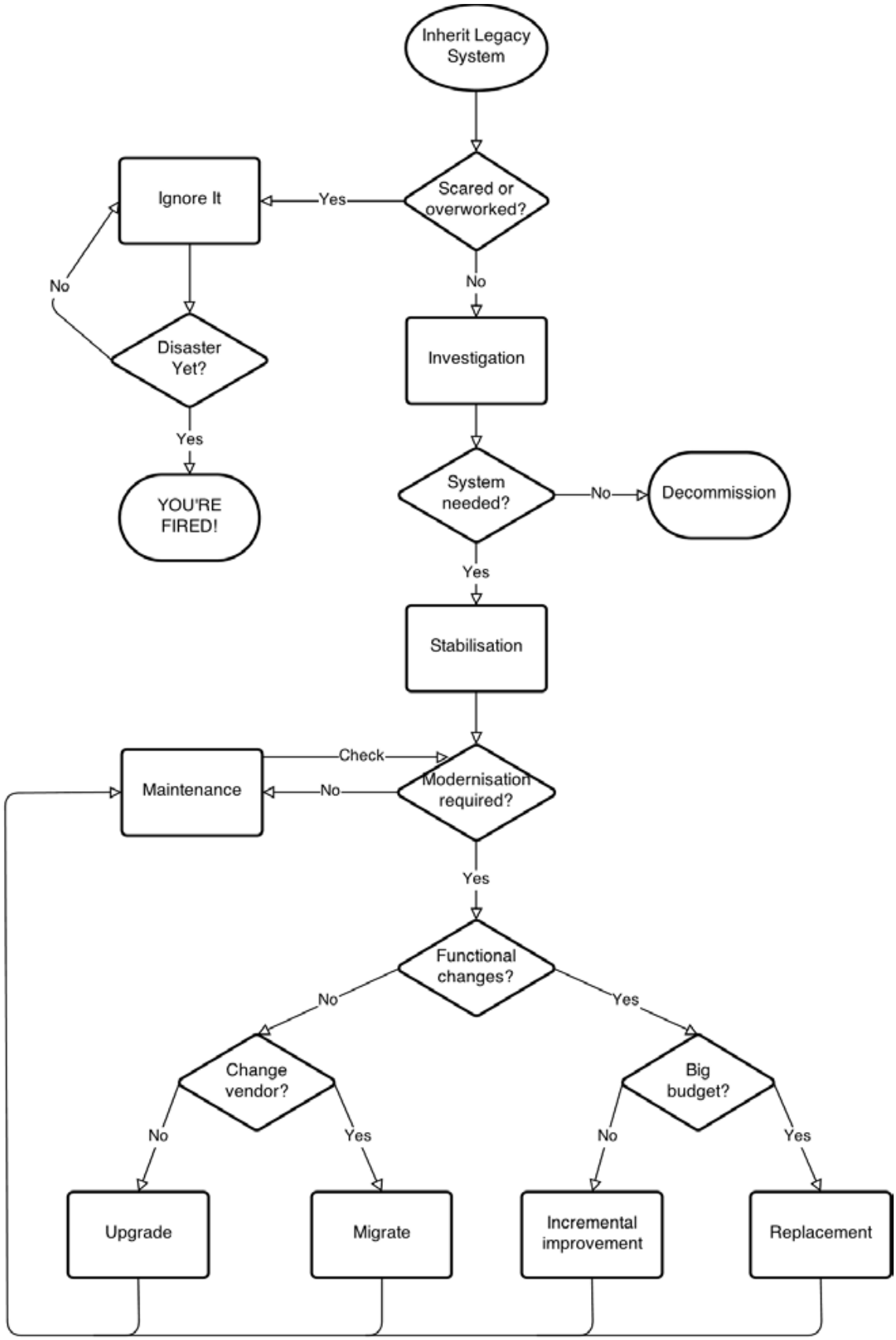


Abbildung 3: Strategy Decision Flowchart (Annett, R. 2019. S.22)

Zur Strukturierung dieses Entscheidungsprozesses stellt die Abbildung 3 „*Strategy Decision Flowchart*“ vor, das den Umgang mit übernommenen Legacy-Systemen systematisch beschreibt (Annett, 2019). Das Modell verdeutlicht, dass ein Ignorieren bestehender Systeme zwar kurzfristig möglich ist, langfristig jedoch erhebliche Risiken birgt.

Die schrittweise Vorgehensweise startet mit einer Untersuchung des Systems, dessen fortbestehenden Nutzen bewertet sowie zunächst auf Stabilisierung abzielt. Auf dieser Grundlage wird entschieden, ob ein reiner Wartungsbetrieb genügt oder eine gezielte Modernisierung notwendig ist (Annett, 2019).

Eine Modernisierung ist jedoch nicht in allen Fällen sinnvoll. Läuft ein System stabil, weist es eine geringe Änderungshäufigkeit auf und verursacht überschaubare Wartungskosten, kann ein fortgesetzter Wartungsbetrieb wirtschaftlich gerechtfertigt sein. Gleiches gilt, wenn der zugrunde liegende Geschäftsprozess bald entfällt oder durch eine Standardlösung ersetzt wird (Annett, 2019). Die schrittweise Vorgehensweise beginnt somit mit der Untersuchung des Systems, der Bewertung seines Nutzens und der Stabilisierung als Grundlage für die anschließende Entscheidung über Wartung oder Modernisierung.

Zusammenfassend ist eine Modernisierung insbesondere dann geboten, wenn geschäftskritische Risiken – etwa in Bezug auf Verfügbarkeit, Sicherheit oder regulatorische Anforderungen – zunehmen und zugleich Änderungs- sowie Betriebskosten deutlich steigen, während der fachliche Mehrwert des Systems erhalten bleiben oder ausgebaut werden soll. Die Modernisierungsentscheidung bildet somit die konzeptionelle Grundlage für die Auswahl geeigneter Strategien, die im folgenden Kapitel näher erläutert werden.

2.4 Software und moderne Frameworks

Moderne Frontend Frameworks bilden eine zentrale Grundlage für die Entwicklung performanter und wartbarer Webanwendungen und ermöglichen die Umsetzung komplexer, interaktiver Benutzeroberflächen. Vor allem JavaScript hat sich in den letzten Jahren von einer reinen Skriptsprache für einfache Interaktionen zu einer vollwertigen Technologie für die Entwicklung komplexer Web- und Anwendungsarchitekturen entwickelt, was maßgeblich durch den Einsatz moderner Frameworks wie Angular, React und Vue.js geprägt wurde (Donvir, 2024).

In diesem Kapitel werden ausgewählte JavaScript-basiert Frontend Frameworks – wie oben genannt – im Hinblick auf ihre technologischen Eigenschaften, ihre Interoperabilität sowie ihre grundsätzliche Eignung für den Einsatz in WebGIS Anwendungen eingeordnet und analysiert. Ausgehend von JavaScript als gemeinsamer technologischer Basis erfolgt ein strukturierter Vergleich anhand definierter Kriterien, der als Grundlage für den Übergang zur nachfolgenden Fallstudie dient.

2.4.1 Grundlagen moderner Webtechnologien

Heutzutage basieren Webanwendungen auf einer klaren Trennung von Präsentations-, Logik- und Datenebene und nutzen standardisierte Webtechnologien zur Umsetzung interaktiver Benutzeroberflächen. Zentrale Grundlage bildet dabei das World Wide Web mit seinen Kernkomponenten HTML zur Strukturierung von Inhalten, CSS zur Darstellung sowie JavaScript zur Implementierung dynamischer Funktionalität und clientseitiger Logik (Lang & Bohne-Lang, 2019). Diese Technologien ermöglichen die Entwicklung plattformunabhängiger Anwendungen, die über standardisierte Protokolle wie HTTP kommunizieren und in Webbrowsern ausgeführt werden.

JavaScript hat sich dabei von einer ursprünglich einfachen Skriptsprache zu einer zentralen Technologie für die Entwicklung komplexer Webanwendungen entwickelt. Durch die Einführung moderner Sprachkonzepte, modularer Architekturen und leistungsfähiger Laufzeitumgebungen bildet JavaScript heute die Basis zahlreicher Frontend-Frameworks, die eine strukturierte Entwicklung großer Anwendungen unterstützen (Donvir, 2024).

Diese Frameworks abstrahieren wiederkehrende Aufgaben wie Datenerfassung, Zustandsverwaltung, Komponentenstrukturierung und Ereignisverarbeitung und tragen damit wesentlich zur Wartbarkeit und Skalierbarkeit von Webanwendungen bei.

Ein weiteres zentrales Merkmal moderner Webtechnologien ist ihre Interoperabilität. Webanwendungen sind in der Lage, über klar definierte Schnittstellen mit unterschiedlichen Backend-Systemen zu interagieren, wodurch eine lose Kopplung der Systemkomponenten ermöglicht wird. Diese Architektur erleichtert insbesondere die schrittweise Modernisierung bestehender Systeme, da einzelne Komponenten unabhängig voneinander weiterentwickelt oder ersetzt werden können (Lang & Bohne-Lang, 2019).

2.4.2 Kriterien der Interoperabilität

Interoperabilität stellt ein zentrales Qualitätsmerkmal moderner Softwaresysteme dar und ist insbesondere im Kontext verteilter Web- und GIS-Anwendungen von grundlegender Bedeutung. In heterogenen Systemlandschaften, wie sie für WebGIS-Anwendungen typisch sind, müssen unterschiedliche Softwarekomponenten zuverlässig zusammenarbeiten, obwohl sie auf verschiedenen Technologien, Plattformen oder Architekturen basieren (Haghwerdi-Poor, 2010). Eine etablierte Definition liefert Wegner, der Interoperabilität als „*the ability of two or more software components to cooperate despite differences in language, interface, and execution platform*“ beschreibt (Wegner, 1996). Diese Aussage verdeutlicht, dass Interoperabilität mehrere Ebenen gleichzeitig adressiert und keine systemspezifischen Anpassungen erfordert.

Für GIS-basierte Informationssysteme besitzt Interoperabilität daher eine besondere Relevanz, da Geodaten, bestimmten Funktionen und Dienste typischerweise aus heterogenen Quellen stammen. Haghwerdi-Poor weist darauf hin, dass fehlende Interoperabilität zu isolierten Anwendungen, proprietären Datenformaten und

eingeschränkter Austauschbarkeit führt, wodurch das Potenzial von GIS als übergreifendes Unterstützungssystem für die Entscheidungsgrundlage erheblich eingeschränkt wird (Haghwerdi-Poor, 2010). Eine interoperable Architektur ist daher Voraussetzung für nachhaltige und integrierte GIS-Nutzung, insbesondere bei der Modernisierung von Legacy-Systemen.

Zur systematischen Bewertung moderner Frontend-Frameworks wird in dieser Arbeit die Kategorisierung der Interoperabilität nach Wegner herangezogen, da sie sich direkt auf technische und architektonische Eigenschaften webbasierter Systeme übertragen lässt. Die drei zentralen Kriterien lauten:

- *Unabhängigkeit von Programmiersprache:* Beschreibt die Fähigkeit, mit Komponenten unterschiedlicher Sprachen zu interagieren. Wegner betont die Rolle vermittelnder Komponenten: „ *Clients and servers from different software platforms or programming languages can talk to each other through mediators that convert data formats* "(Wegner, 1996). In WebGIS wird dies durch Standardformate wie JavaScript Object Notation (JSON)/ Extensible Markup Language (XML) und Web application programming interfaces (Web-API) realisiert, was die Integration heterogener Backend-Systeme erleichtert.

- *Unabhängigkeit von Schnittstellen:* Ermöglicht konsistente Funktionsaufrufe über definierte Interfaces. Wegner unterscheidet hier „interface standardization" (skalierbar) von „interface bridging" (flexibel): „*Interface standardization is more scalable, whereas interface bridging is more flexible*" (Wegner, 1996). Für WebGIS-Anwendungen sind Representational State Transfers (REST-API) und Web-Services entscheidend, da sie lose Kopplung und schrittweise Modernisierung fördern (Haghwerdi-Poor, 2010).

- *Unabhängigkeit von der Plattform*: Garantiert vergleichbare Funktionalität auf unterschiedlichen Endgeräten. „*Interoperability can be realized despite differences in execution platforms*" (Wegner, 1996). Webbrowser als universelle Runtime erfüllen dies und reduzieren den Wartungsaufwand (Haghwerdi-Poor, 2010).

Diese Kategorisierung bildet die methodische Basis für den nachfolgenden Framework Vergleich und bezieht zentrale Anforderungen der Legacy-Modernisierung in WebGIS Anwendungskontexten.

2.4.3 Grundlagen und Begriffserklärungen von Framework

Ein Framework ist eine technologische Grundlage, die Entwicklern eine strukturierte Architektur, wiederverwendbare Bibliotheken sowie vorgegebene Programmierkonventionen und Schnittstellen bereitstellt. Im Gegensatz zu einer einfachen Bibliothek, bei der der Entwickler selbst den Programmfluss kontrolliert, bestimmt bei einem Framework das Framework selbst (*Inversion of Control*) zu wesentlichen Teilen

den Ablauf – der Entwickler „füllt“ lediglich vordefinierte Stellen mit projektspezifischer Geschäftslogik (Sencha, 2022).

Ziel ist die Vereinfachung und Vereinheitlichung komplexer Entwicklungsaufgaben durch ein konsistentes Regelwerk, wodurch Wartbarkeit, Erweiterbarkeit und Codequalität verbessert werden (Fayad & Schmidt 1997).

Frameworks können nach Zweck und Schichtebene unterschieden werden, insbesondere zwischen serverseitigen Frameworks (Back-End) und clientseitigen Frameworks (Front-End).

- **Serverseitige Frameworks** (vgl. Kapitel 2.4.4) – z. B. Java-basierte Webframeworks wie Spring MVC oder Apache Struts – verarbeiten Anfragen auf dem Server, steuern den Datenfluss zum Client und erzeugen serverseitig HTML-Ausgaben. Diese Frameworks orientieren sich häufig am MVC-Architekturmuster (Model–View–Controller), das eine klare Trennung zwischen Datenmodell, Benutzeroberfläche und Steuerungslogik ermöglicht (GeeksforGeeks , 2025).
- **Clientseitige Frameworks** (vgl. Kapitel 2.4.5), insbesondere sogenannte SPA-Frameworks (Single-Page Application Frameworks), laufen im Browser und organisieren die Darstellung, Navigation und Zustandsverwaltung von Webanwendungen. Sie basieren auf JavaScript (bzw. TypeScript) und ermöglichen es, Inhalte dynamisch nachzuladen und im Browser darzustellen, ohne vollständige Seitenneuladevorgänge durchzuführen. Bekannte Vertreter sind React, Angular und Vue.js (AlmaBetter, 2024).

Frameworks nehmen somit eine zentrale Rolle in der modernen Softwareentwicklung ein, da sie als technologisches Gerüst zwischen Infrastruktur und konkreter Anwendung fungieren und je nach Technologie-Stack (z. B. Java-Server-Framework oder JavaScript-SPA-Framework) unterschiedliche Aufgaben im gesamten Anwendungszyklus übernehmen.

2.4.4 Serverseitige Java-Frameworks

In GeeksforGeeks (2025) veröffentlichte Artikel beschreibt, dass serverseitige Java-Frameworks das Fundament moderner Unternehmens- und Webanwendungen bilden, ndem sie die Verarbeitung von Benutzeranfragen, Datenbankzugriffen und Geschäftslogik auf dem Server zentral steuern. Sie abstrahieren wiederkehrende Aufgaben wie Routing, Formularverarbeitung und Sitzungsverwaltung und ermöglichen die Umsetzung des **Model-View-Controller (MVC)**-Architekturmusters, das Logik (Model), Benutzeroberfläche (View) und Steuerung (Controller) klar voneinander trennt.

Zudem betont GeeksforGeeks (2025), dass MVC-Muster als zentrales Architekturprinzip serverseitiger Java-Webframeworks wie Struts und Spring MVC darstellt, um Anwendungslogik, Daten und Darstellung klar zu trennen.

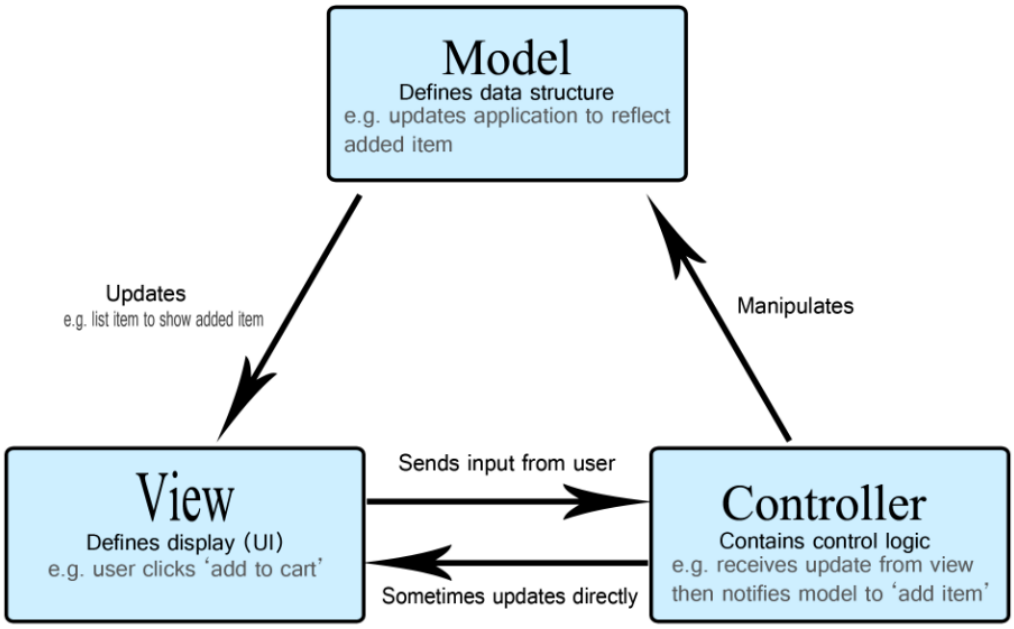


Abbildung 4: Model View Controller Muster. Quelle: https://developer.mozilla.org/en-US/docs/Glossary/MVC

Ein Java Webframework bietet eine strukturierte Softwarearchitektur, in der **Servlets** und **JavaServer Pages (JSP)** grundlegende Bausteine darstellen. Darauf aufbauend sind im Laufe der Zeit erweiterte Frameworks entstanden, die den Entwicklungsaufwand weiter reduzieren und Skalierbarkeit sowie Wartbarkeit verbessern.

Für diese Masterarbeit ist ein überblicksartiger Vergleich der maßgeblichen serverseitigen Java-Webframeworks angebracht, um den thematischen Umfang klar zu begrenzen. Technische Tiefenanalysen würden den Rahmen dieser Arbeit sprengen und zu stark in Detailfragen abdriften. Die Auswahl fokussiert daher auf markt- und community-dominante Vertreter mit Schwerpunkt serverseitiger Logik (MVC sowie komponentenbasierte Modelle), die für Enterprise-Umgebungen – etwa ÖPNV-Infrastrukturen mit langfristigen Betriebsanforderungen sowie aktuellen ISDB technisch entsprechend – hochrelevant sind.

Tabelle 1: Top 10 Java Webframeworks Vergleich (Quelle: GeeksforGeeks 2025)

Framework	Architekturtyp / Eigenschaften	Vorteile	Nachteile
Spring / Spring Boot	Serverseitiges MVC-Framework, modular, konfigurierbar über Annotations & XML. Spring Boot ermöglicht vereinfachte Konfiguration	Sehr flexibel, riesiges Java Ökosystem, für Enterprise-Apps etabliert	Hohe Komplexität, Lernkurve durch viele Module und Konzepte
Grails	Vollständiges, dynamisches MVC-Framework basierend auf Groovy (läuft auf der JVM); unterstützt REST und Java-Integration.	Schnelle Entwicklung, flexible Konfiguration, Groovy-/Java-Mischbetrieb möglich, keine Neustarts erforderlich.	Kleinere Community, Performance im Vergleich zu reinem Java teils eingeschränkt.
Google Web Toolkit (GWT)	Compiled Java-Code in JavaScript; ermöglicht Entwicklung komplexer browserbasierter Anwendungen in Java	Frontend-Entwicklung ohne JavaScript-Kenntnisse, optimierter Code für alle Browser, gute Debugging-Möglichkeiten.	Build-Prozess teils langwierig, weniger geeignet für moderne JS-Framework-Ökosysteme.
Struts	Klassisches serverseitiges, Action-basiertes MVC; XML/Annotation-Konfiguration, Plugin-Architektur.	Klare MVC-Trennung, stabile Architektur, gute Integration mit Spring/Hibernate, bewährt für Legacy-Migration.	Technologisch überholt, als schwergewichtig und unflexibel kritisiert.
JavaServer Faces (JSF)	Komponentenbasiertes serverseitiges MVC; Java-EE-Standard mit Facelets und UI-Bibliotheken.	Umfangreiche Komponenten, gute Tool-Unterstützung, standardisiert.	Komplexes Lebenszyklusmodell, steile Lernkurve.
Hibernate	ORM für Datenzugriff; kein Web-Framework.	De-Facto Object-Relation-Mapping (ORM) Standard, starke DB-Abstraktion.	Kein UI-Layer, benötigt Webframework.
Play	Reaktives, leichtgewichtiges Webframework; asynchrones, REST-orientiertes Modell; basiert auf reaktiven	Hohe Skalierbarkeit, produktive Entwicklung, schnelle Hot-Reload-Funktion, reaktive Architektur.	Event-basiertes Design kann komplex und schwer nachvollziehbar

	Prinzipien.		werden.
Vaadin (Classic)	Client-Server-Modell mit Web Components; ermöglicht UI-Erstellung rein in Java.	Produktivitätssteigerung, viele UI-Komponenten, Automatisierung der Client-Server-Kommunikation.	Großer Ressourcenbedarf eingeschränkte Kontrolle über Frontend-Details.
Wicket	Komponentenorientiertes Lightweight-Framework basierend auf Java und HTML (ohne XML-Konfiguration).	Einfaches, klares OOP-Modell, einfache Tests, Wiederverwendbarkeit von Komponenten.	Für große Projekte eingeschränkt geeignet, geringere Verbreitung als Spring/Play.
Dropwizard	Lightweight-Framework kombiniert Jetty, Jersey, Jackson, Metrics zu einem REST-basierten Stack.	Schneller Projektstart (Bootstrap), integrierte Stabilitäts- und Security-Libraries, gut geeignet für Microservices.	Begrenzte Flexibilität außerhalb REST-APIs, weniger stark erweiterbar als Spring Boot.

Die Tabelle 1 bietet einen vergleichenden Überblick über ausgewählte serverseitige Java-Webframeworks hinsichtlich ihrer architektonischen Eigenschaften, Vorteile und Einschränkungen (GeeksforGeeks, 2025). Moderne Frameworks wie Spring Boot und Play zeichnen sich durch eine hohe Flexibilität, Skalierbarkeit sowie die Unterstützung aktueller Architekturprinzipien wie REST und reaktive Programmierung aus, wenngleich sie mit einer erhöhten Komplexität und einer steileren Lernkurve verbunden sind.

Klassische Frameworks wie Struts und JavaServer Faces (JSF) bieten zwar bewährte und stabile Architekturen, stoßen jedoch aufgrund monolithischer Strukturen und eingeschränkter Erweiterbarkeit zunehmend an technologische Grenzen und eignen sich daher primär für bestehende Legacy-Systeme. Grails und Dropwizard positionieren sich hingegen als produktivitätsorientierte Alternativen, die schnellere Entwicklungszyklen ermöglichen, jedoch teilweise Einschränkungen hinsichtlich Flexibilität und langfristiger Wartbarkeit aufweisen.

Der Vergleich verdeutlicht, dass die Wahl eines geeigneten Frameworks maßgeblich von den Anforderungen an Skalierbarkeit, Wartbarkeit und Integrationsfähigkeit abhängt. Für die Modernisierung komplexer WebGIS-Systeme erweisen sich insbesondere modulare, API-zentrierte Ansätze wie Spring Boot als zukunftsfähige Lösung.

Vor dem Hintergrund aktueller Anforderungen an moderne Webanwendungen – etwa entkoppelte Mikroservices-Architekturen und API-first-Designs – stoßen historische Frameworks wie Apache Struts zunehmend an konzeptionelle Grenzen, insbesondere in komplexen Systemlandschaften mit standardisierten REST/GraphQL-Schnittstellen.

2.4.5 Clientseitige SPA-Frameworks

Im Zuge der zunehmenden Anforderungen an Interoperabilität (vgl. Kapitel 2.4.2), Performance und Benutzerbedürfnis haben sich clientseitige Frontend Frameworks als zentrales Element moderner Webanwendungen etabliert. Sie verlagern wesentliche Teile der Präsentations- und Interaktionslogik in den Browser und unterscheiden sich insbesondere hinsichtlich ihres Architekturansatzes, ihres Abstraktionsgrades sowie ihrer Integrationsfähigkeit in bestehende Backend-Strukturen (Tyson & Maier, 2025).

Single-Page-Application (SPA) Frameworks entstanden ab etwa 2010er Jahren als Reaktion auf die strukturellen Einschränkungen klassischer Multi-Page Anwendungen. Diese waren durch häufige vollständige Seitenneuladungen, hohe Server-Roundtrip-Zeiten und eine eingeschränkte Benutzererfahrung geprägt. Frühere Anwendungen wie Gmail (2004) und Google Maps demonstrierten erstmals das Potenzial dynamischer, browserzentrierter Interaktionen auf Basis von AJAX-Techniken und markierten einen grundlegenden Paradigmenwechsel in der Webentwicklung.

Ein **SPA-Framework** ist ein übergeordneter Begriff für JavaScript-Frameworks oder -Bibliotheken und stellt den technischen Rahmen zur Entwicklung von Single-Page-Webanwendungen dar, indem es zentrale Funktionen wie Routing, Komponentenmodelle, Zustandsverwaltung und UI-Rendering bereitstellt (Flanagan, 2006).

Auf dieser Grundlage etablierten sich spezialisierte Frameworks, die diesen Ansatz systematisierten:

Tabelle 2: Überblick ausgewählter Single-Page-Application-Frameworks (Quelle: Tyson & Maier 2025)

Framework	Jahr	Kerninnovation und Architekturansatz
AngularJS	2010	Browserseitiges MVVM-Ansatz mit Two-Way-Data-Binding (früher MVC-Muster); Hoher Abstraktionsgrad für Enterprise-Anwendungen
React	2013	Komponentenbasierte UI-Entwicklung mit Virtual DOM; Deklarative Programmierung und einseitiges Data Binding

Vue.js	2014	Progressive, inkrementell integrierbares Framework; Reaktives Data Binding mit Fokus auf Einfachheit und Flexibilität

Grundlagen von SPA:

- Eine SPA besteht technisch aus einem initial geladenen HTML-Dokument, dessen Inhalte clientseitig dynamisch aktualisiert werden.
- Der Server stellt primär Daten (z. B. JSON) bereit, während Navigation, Zustandsverwaltung und UI-Updates im Browser erfolgen.

Eigenschaften eines SPA-Frameworks:

- Das Framework stellt Struktur und Bausteine bereit (Routing, Komponenten, State-Management, Templating), um solche SPA effizient zu bauen.
- Technologie Vertreter sind React, Angular, Vue, die genau diese Funktion als Rahmenwerk („Framework“) für die eigentliche Anwendung erfüllen (siehe Tabelle 2).

Angrenzung zu „SPA vs. SPA-Framework“ selbst:

- Die SPA ist die konkrete, im Browser ausgeführte Anwendung.
- Das SPA-Framework fungiert als generisches Werkzeug, mit dem viele unterschiedliche SPAs entwickelt werden können, jedoch für sich allein noch keine komplette Anwendung darstellt

2.4.5.1 JavaScript

JavaScript ist eine hochrangige, dynamische Skriptsprache, die als zentrale Technologie moderner Webanwendungen in nahezu allen Webbrowsern ausgeführt werden kann. Durch die Ausführung in leistungsfähigen JavaScript-Engines, die häufig Just-in-Time (JIT) Kompilierung einsetzen, ermöglicht die Sprache die effiziente Umsetzung interaktiver und dynamischer Benutzeroberflächen (Jones, 2017).

Historisch entstand JavaScript Mitte der 1990er-Jahre als Reaktion auf den steigenden Bedarf an Interaktivität im World Wide Web und verbreitete sich aufgrund seiner geringen Einstiegshürden rasch. Trotz anfänglicher konzeptioneller Schwächen entwickelte sich JavaScript zur dominierenden clientseitigen Programmiersprache und bildet heute die Grundlage zahlreicher moderner Frontend-Frameworks (Jones, 2017).

Mit der zunehmenden Nutzung von JavaScript für komplexe Webanwendungen ist der Wettbewerb zwischen Browserherstellern vor allem durch Standardkonformität und die Leistungsfähigkeit der JavaScript-Engines geprägt. Mit der Einführung der V8-Engine durch Google im Jahr 2008 wurde ein Wendepunkt erreicht, da sie eine signifikante

Leistungssteigerung bewirkte und eine bis heute anhaltende Optimierungsdynamik der JavaScript-Ausführungsumgebungen moderner Browser auslöste (Jones, 2017).

2.4.5.2 Node.js

Mit der Entwicklung von Node.js durch Ryan Dahl im Jahr 2009 wurde JavaScript erstmals umfassend für serverseitige Anwendungen nutzbar. Basierend auf der leistungsfähigen V8-Engine ermöglicht die plattformübergreifende Laufzeitumgebung ereignisgesteuerte, nicht-blockierende Architekturen für skalierbare Echtzeit-Anwendungen und förderte das „JavaScript everywhere"-Konzept – einschließlich isomorpher Codebases für Client und Server (Jones, 2017). Für Frameworks wie React, Angular und Vue.js ist Node.js essenziell, da es „Node Package Manager" (npm)-Abhängigkeiten, Builds (Webpack/Vite) und Full-Stack-APIs (Express/NestJS) in einer einheitlichen Sprache realisiert.

Die Auswertung der Programmiersprachennutzung in Anhang 1 zeigt eine deutlich höhere Verbreitung von JavaScript, die primär auf dessen umfassendes Ökosystem sowie die zentrale Rolle des zustandsbasierten UI-Managements in modernen Webframeworks wie React, Angular und Vue.js zurückzuführen ist. Die Korrelation mit der Webframework-Nutzung in Anhang 2 unterstreicht das Ergebnis: Die hohe Verbreitung von React, Angular und Vue.js resultiert aus ihrer Schlüsselrolle bei der Entwicklung komplexer, interaktiver Webanwendungen, bei denen Wartbarkeit, Skalierbarkeit und eine klare Trennung von Zuständigkeiten zentrale Anforderungen darstellen.

Node.js und Deno dienen ausschließlich als Laufzeitumgebungen für serverseitige JavaScript-Anwendungen, ohne UI-Framework-Funktion. jQuery ist eine klassische Hilfsbibliothek zur DOM-Manipulation und Browser-Abstraktion, die kein komponenten- oder zustandsbasiertes Modell verfolgt und für komplexe Webanwendungen nur bedingt geeignet ist.

Da der Fokus dieser Masterarbeit auf der Modernisierung der Frontend-Weboberfläche eines bestehenden WebGIS-Systems liegt, beschränkt sich der technologische Vergleich gezielt auf aktuell wichtigsten Frontend-Frameworks. Ziel ist es, geeignete Technologien für die Umsetzung wartbarer, erweiterbarer und performanter WebGIS-Oberflächen zu identifizieren und deren Eignung im Kontext moderner Webarchitekturen zu bewerten.

Backend-Technologien (vgl. Kapitel 2.4.4), die im bestehenden System unter anderem in Form von Java-basierten Lösungen mit Spring Boot realisiert sind, bilden nicht den primären Untersuchungsgegenstand dieser Masterarbeit und werden lediglich im Rahmen notwendiger Anpassungen berücksichtigt. Die ergänzende Nutzungsauswertung von Datenbanksystemen in Anhang 3 vervollständigt diese Betrachtung und bildet das typische technologische Gesamtsetup moderner Webanwendungen ab.

2.4.5.3 React

React wurde erstmals 2013 von Facebook veröffentlicht und wird explizit als *JavaScript Library zur Erstellung von Benutzeroberflächen* beschrieben, nicht als vollständiges Framework. Der Fokus liegt auf der komponentenbasierten Darstellungsschicht, während Routing, State-Management und Build-Prozesse bewusst ausgelagert sind (Kaluža & Vukelic, 2018).

Zentraler architektonischer Baustein ist das Virtual *Document Object Model* (DOM), das Änderungen zunächst im Speicher berechnet und anschließend effizient mit dem realen DOM synchronisiert. Dadurch lassen sich auch komplexe UI-Strukturen performant aktualisieren. React eignet sich besonders für modulare, zustandsgetriebene Single-Page-Anwendungen (SPA), setzt jedoch für größere Projekte eine bewusste Auswahl und Integration externer Bibliotheken voraus (z. B. Router, State-Management) (Kaluža & Vukelic, 2018).

Im Vergleich zu Angular und Vue.js weist React geringen Overhead, aber gleichzeitig hohe Abhängigkeit von Drittbibliotheken auf. Server-Side-Rendering ist möglich, jedoch nicht Bestandteil des Kernsystems und wird über zusätzliche Lösungen realisiert.

Tabelle 3: Vergleich React

Kriterium	**Bewertung**
Erstveröffentlichung	2013
Architektur	Komponentenbasiert, Virtual DOM
Technischer Laufzeit-Overhead	Gering
Workflow erforderlich	Teilweise
State Management	Extern (z. B. Redux)
Server Side Rendering	Nur mit Zusatzlösungen
WebGIS Anwendung Eignung	Gut, bei klarer Architekturdisziplin

React verwendet ausschließlich One-Way Data Binding, wodurch Datenänderungen stets explizit über den Anwendungszustand gesteuert werden. Dieser Ansatz erhöht die Vorhersagbarkeit bei komplexen Datenflüssen und eignet sich besonders für WebGIS-Anwendungen, die große, relationale Datensätze über REST-Schnittstellen beziehen.

Vorteil: keine versteckte Seiteneffekte

Nachteil: mehr Boilerplate (JSX-Templates)

2.4.5.4 Angular

Angular wurde ursprünglich 2010 als AngularJS entwickelt und 2014 komplett neu konzipiert. Es handelt sich um ein vollständiges Frontend-Framework, das sämtliche Aspekte der Client-Entwicklung integriert: Komponenten, Routing, Dependency Injection, Build-Workflow und State-Management (Kaluža & Vukelic, 2018).

Angular verfolgt einen stark strukturierten, modularen Ansatz auf Basis von TypeScript. Dies begünstigt die Entwicklung und Wartung großer, komplexer Anwendungen, führt jedoch gleichzeitig zu höherem initialem Aufwand und größerem Overhead. Ein Einstieg ohne Build-Workflow ist nicht vorgesehen, da TypeScript vor der Ausführung kompiliert werden muss (Kaluža & Vukelic, 2018).

Besondere Eigenschaft ist die offizielle Unterstützung für Server-Side-Rendering sowie klar definierte Styleguides und Best Practices, die Angular für langfristige Enterprise-Anwendungen prädestinieren. Für kleinere oder inkrementelle Modernisierungsszenarien ist Angular jedoch weniger flexibel (Kaluža & Vukelic, 2018).

Tabelle 4: Vergleich Angular

Kriterium	**Bewertung**
Erstveröffentlichung	2010 (Neufassung 2014)
Architektur	Vollständiges Framework
Technischer Laufzeit-Overhead	Hoch
Workflow erforderlich	Ja
State Management	Integriert/ erweitert
Server Side Rendering	Offiziell unterstützt
WebGIS Anwendung Eignung	Gut für große, monolithische SPAs

Angular unterstützt Two-Way Data Binding, setzt in größeren Anwendungen jedoch auf reaktive Datenströme (RxJS). Dies erlaubt eine sehr präzise Steuerung komplexer Datenabhängigkeiten, führt jedoch zu höherer Komplexität und Lernkurve.

Vorteil: stabil bei sehr komplexen Datenmodellen

Nachteil: hoher Overhead für UI-lastige GIS-Anwendungen

2.4.5.5 *Vue.js*

Vue.js wurde von Evan You 2014 gestartet und 2015 die erste stabile Version veröffentlicht. Somit positionierte sie sich bewusst zwischen React und Angular. Es kombiniert eine niedrige Einstiegshürde mit einer klaren komponentenbasierten

Architektur und bietet sowohl einfache Script-Integration als auch skalierbare Projektstrukturen mit Build-Workflow (Kaluža, M & Vukelic B 2018, S.268-269). Vue.js nutzt ebenfalls ein Virtual DOM und trennt Darstellung, Logik und Zustand klar voneinander. Im Gegensatz zu React existieren offizielle Lösungen für Routing und State-Management, was die Abhängigkeit von Drittanbietern reduziert. Gleichzeitig bleibt das Framework modular und flexibel erweiterbar. Diese Ausgewogenheit macht Vue.js besonders geeignet für schrittweise Modernisierungsstrategien bestehender Webanwendungen (Kaluža & Vukelic, 2018).

Tabelle 5: Vergleich Vue.js

Kriterium	**Bewertung**
Erstveröffentlichung	2016
Architektur	Komponentenbasiert, Virtual DOM
Technischer Laufzeit-Overhead	Gering
Workflow erforderlich	Optional
State Management	Offiziell Vuex
Server Side Rendering	Offiziell unterstützt
WebGIS Anwendung Eignung	Sehr gut auch für Migration

Vue.js bietet integriertes Two-Way Data Binding mit einem sehr transparenten Reaktivitätsmodell. Änderungen in API-Daten werden unmittelbar und nachvollziehbar in der Oberfläche reflektiert, was insbesondere für kartenbasierte, filterintensive WebGIS-Oberflächen vorteilhaft ist.

Vorteil: hohe Produktivität bei datengetriebenen Benutzeroberflächen

Nachteil: geringere Standardisierung als Angular

2.4.5.6 *Svelte*

Svelte unterscheidet sich grundlegend von den zuvor genannten Frameworks, da es keinen Virtual DOM zur Laufzeit verwendet. Stattdessen erfolgt die Verarbeitung zur Build-Zeit, wobei reaktiver Code in optimiertes JavaScript kompiliert wird. Dadurch entstehen sehr kleine Bundle-Größen und hohe Laufzeitperformance.

Die Studie von Kaluža et al. (2018) berücksichtigt Svelte noch nicht explizit, da das Framework erst später an Relevanz gewonnen hat. Dennoch lässt sich Svelte konzeptionell als kompilierendes Frontend-Framework einordnen, das insbesondere bei performancekritischen Oberflächen Vorteile bietet.

Tabelle 6: Vergleich Svelte

Kriterium	Bewertung
Erstveröffentlichung	2016
Architektur	Compile-time, kein Virtual DOM
Technischer Laufzeit-Overhead	Sehr gering
Workflow erforderlich	Ja
State Management	Integriert
Server Side Rendering	Möglich
WebGIS Anwendung Eignung	Potenziell, aber noch sehr begrenzt

Für datenintensive Anwendungen ist der Ansatz technisch möglich, jedoch fehlen derzeit Best Practices sowie nötigen Toolchans im Zusammenspiel mit komplexen, relationalen GIS-Backends.

Vorteil: Performance bei kleineren Anwendungen

Nachteil: geringere Projektreife

2.4.6 SPA-Framework Zusammenfassung

Zur vergleichenden Bewertung moderner Frontend-Frameworks werden die Ergebnisse der Studie von Kaluža et al. (2018) herangezogen, in der Angular, Vue.js und React anhand eines strukturierten Kriterienkatalogs für die Entwicklung von Multi-Page- (MPA) und Single-Page-Applications (SPA) untersucht wurden. Die Studie kombiniert qualitative Einschätzungen mit quantitativen Bewertungen und ermöglicht dadurch eine nachvollziehbare, methodisch fundierte Gegenüberstellung der Frameworks.

Die Ergebnisse zeigen, dass Vue.js im Gesamtranking sowohl für MPA- als auch für SPA-Anwendungen die besten Bewertungen erzielt. Dies wird auf die geringe Einstiegshürde, die ausgewogene Architektur und die starke Unterstützung für Performance-Optimierung und State-Management zurückgeführt. React erreicht ebenfalls hohe Werte, bleibt jedoch insgesamt hinter Vue.js zurück. Angular überzeugt vor allem im SPA-Kontext, weist jedoch im MPA-Bereich Schwächen auf – insbesondere durch seine hohe Komplexität und den vergleichsweise großen technischen Overhead.

Diese Ergebnisse stehen in engem Zusammenhang mit den im Rahmen dieser Arbeit durchgeführten Framework-Analysen. Tabelle 6 fasst die zentralen Unterschiede im Hinblick auf State-Management, den Umgang mit komplexen Datenstrukturen sowie die Eignung für datenbankgestützte WebGIS-Anwendungen zusammen.

Tabelle 7: Vergleichstabelle - Fokus Data Bindung & Datenbank nahe Nutzung

Framework	State Management & Reaktivität	Umgang mit komplexen Daten	Eignung für DB-basierte WebGIS
React	Explizit, zustandsgetrieben	Sehr gut bei großen JSON-Strukturen, klare Kontrolle	Sehr hoch – gut geeignet für REST-/API-basierte GIS-Backends
Angular	Stark formalisiert (RxJS)	Sehr gut, aber höherer bis extremer Overhead	Hoch – robust für komplexe Enterprise-Datenmodelle
Vue.js	Reaktiv, transparent	Sehr gut, geringer Boilerplate	Sehr hoch – ideal für datenintensive UI-Layers
Svelte	Compiler-gesteuert	Wenig erprobt	Eingeschränkt – geringere Praxisreife bei großen GIS-Daten

React überzeugt durch sein explizites, zustandsgetriebenes Datenmodell und seine gute Integration in REST- und API-basierte Backends. Angular bietet dank seines RxJS-basierten Reaktivitätsmodells eine hohe Robustheit für komplexe Enterprise-Datenmodelle, erfordert jedoch aufgrund des Framework-Umfangs mehr Ressourcen. Vue.js kombiniert eine transparente Reaktivität mit geringem Boilerplate-Code und eignet sich daher besonders für datenintensive, wartbare UI-Schichten. Svelte verfolgt mit seinem Compiler-Konzept einen innovativen Ansatz, ist aber für großskalige WebGIS-Szenarien noch weniger erprobt, insbesondere im Umgang mit umfangreichen Geodatenbeständen.

Für den Einsatz in WebGIS-Anwendungen sind diese Unterschiede insbesondere im Hinblick auf **Wartbarkeit, Erweiterbarkeit** und **Performance** relevant. WebGIS-Systeme weisen eine hohe funktionale Komplexität auf und erfordern zugleich flexible Integrationsmöglichkeiten bestehender Backend- und GIS-Dienste. Frameworks mit modularer Struktur und geringer Einstiegshürde bieten hier Vorteile, da sie inkrementelle Modernisierungen und eine schrittweise Migration bestehender Systeme erleichtern.

Die dargestellten Vergleiche liefern eine fundierte Grundlage für die Auswahl geeigneter Frontend-Technologien, ersetzen jedoch keine projektspezifische Bewertung. Architekturentscheidungen, organisatorische Rahmenbedingungen und spezifische Anforderungen müssen stets ergänzend unter Berücksichtigung von *Arbeitsfrage 1* einbezogen werden. Insgesamt zeigt die vergleichende Analyse, dass die Eignung eines Frameworks nie isoliert, sondern stets im Kontext der jeweiligen Systemlandschaft betrachtet werden muss.

Während allgemeine Kriterien wie Architektur, Performance und Wartbarkeit erste Orientierung bieten, entscheidet sich die Praxistauglichkeit im Zusammenspiel mit

vorhandenen Legacy-Strukturen, Datenmodellen und organisatorischen Gegebenheiten. Das folgende Kapitel greift diese Erkenntnisse auf und überführt sie in diese Fallstudie, die untersucht, wie sich moderne Webtechnologien in bestehende WebGIS-Infrastrukturen integrieren lassen und welche architektonischen Entscheidungen sich daraus ableiten.

2.4.7 Grundlagen von API

Eine Application Programming Interface (**API**) stellt eine klar definierte Softwareschnittstelle dar, über die Softwarekomponenten miteinander interagieren können, ohne Kenntnis der jeweiligen internen Implementierungsdetails zu benötigen. APIs kapseln Funktionalität und stellen diese über wohldefinierte Operationen, Datentypen und Protokolle zur Verfügung, wodurch lose Kopplung und Wiederverwendbarkeit von Softwarekomponenten ermöglicht werden (Reddy, 2011).

Im modernen Softwarearchitekturen fungiert eine API als vertraglich definierte Kommunikationsschnittstelle zwischen unterschiedlichen Systemteilen oder Anwendungen. Sie legt fest, welche Funktionen ein System nach außen anbietet, ohne deren interne Umsetzung offenzulegen (Reddy, 2011).

2.4.7.1 API-Design und Spezifikation

Als Folge der architektonischen Neuausrichtung wurde die Konzeption einer klar definierten API-Schnittstelle zu einem zentralen Bestandteil des Projekts. Anstelle des bislang verfolgten „*Code-first*"-Ansatz wurde ein „*Contract-first*"- oder „*API-first*"-Vorgehen gewählt. Dabei wird die Schnittstelle zunächst unabhängig von der Implementierung spezifiziert – also, bevor Code entsteht (Wagner, 2021).

Im Kontext dieser Masterarbeit dient die API als **verbindende Schicht** zwischen serverseitiger Business-Logik (Legacy-Struts/ISDB) und clientseitigen SPA – als RESTful HTTP-Schnittstelle (*REST-API*) mit JSON-Payloads.

Eine API-Spezifikation, beispielsweise in Form der OpenAPI Specification (OAS) im YAML- oder JSON-Format, beschreibt auf standardisierte Weise die Struktur, Endpunkte, Datentypen und zulässigen Operationen der Schnittstelle (siehe Tabelle 8). Systeme, die eine solche Spezifikation implementieren oder bereitstellen, verpflichten sich zur Einhaltung dieses definierten Vertrags (OpenAPI 2024).

Tabelle 8: Kernmerkmale der API-Spezifikation

Merkmal	Beschreibung	OpenAPI Beispiele
Endpoints	URLs und HTTP-Methoden (CRUD)	`GET: '/api/v1/infrastruktur/strecken'`
Datentypen	JSON-Schemata für Request/Response	`Strecke: { id: integer, name: string }`
Operationen	Validierung, Fehlercodes, Pagination	`200 OK, 404 Not Found`
Sicherheit	Authentifizierung/-torisierung	`OAuth2, JWT Bearer`

2.4.7.2 API-Entwicklungsansätze: Code-First vs. API-/Design-First

Die Entwicklung von APIs als Schnittstelle zwischen serverseitiger Business-Logik (z.B. Struts-basierten ISDB-Systemen) und clientseitigen SPAs erfordert eine strategische Entscheidung über den Entwicklungsansatz.

Tabelle 9 zeigt dass sich im Rahmen der geplanten Migration – Business Logic → REST-API (OpenAPI/YAML-Vertrag) → SPA-Frontend – unterscheiden sich drei grundlegende Paradigmen: Code-First, API-First und Design-First.

Tabelle 9: API-Entwicklungsansätze im Vergleich (Quelle: Wagner J. 2021)

Ansatz	Ablauf	Stärken für Migration
Code-First	Business-Logik → Code → nachtr. Spezifikation	Schnell, aber fehlende Vertragsgarantie
API-First	APIs als Unternehmensasset, Kulturwandel	Wiederverwendbarkeit, Standardisierung fördernd
Design-First	OpenAPI-Design vor Code OAS 3 Standard	Mock-Server, parallele Entwicklung möglich

Der Design-first-Ansatz fördert eine grundlegende Methodik, bei der Stakeholder frühzeitig einbezogen werden und die Spezifikation als „Single Source of Truth“ dient. Dies reduziert Entwicklungsfehler, ermöglicht parallele Arbeiten und generiert automatisch Client – Software Development Kits (SDKs), Dokumentation sowie Tests (Wagner, 2021).

2.4.7.3 API-Spezifikationssprachen

Die OpenAPI Specification (OAS 3.x) hat sich als de-facto-Standard durchgesetzt und verdrängt ältere Alternativen wie WADL oder RAML (Mallisetty, 2023).

Tabelle 10: API-Spezifikationssprachen im Vergleich

Spezifikations-sprache	Status	Merkmale	Einsatz
OpenAPI (OAS 3.x)	Dominant (JSON/YAML)	Hohe Interoperabilität, umfangreiches Tooling-Ökosystem (Swagger, Stoplight), unterstützt REST/GraphQL-ähnliche Features, Extensions, Callbacks.	Enterprise-APIs, Microservices, Cloud-Native.
RAML	Nische (YAML)	Ausdrucksstark, flexibel (Annotations, Overlays), design-orientiert.	Legacy-Projekte, MuleSoft-Ökosystem.
WADL	Veraltet (XML)	JAX-RS-nativ, maschinell generierbar.	Frühe REST-APIs (heute selten).

OpenAPI dominiert durch YAML/JSON-Syntax, breites Ökosystem und Unterstützung für Design-First-Workflows: Spezifikation → Code-Generierung → Validierung (Mallisetty, 2023).

OpenAPI beschreibt vollständige REST-APIs inklusive:

- Endpoints, HTTP-Methoden, Parameter (Path/Query/Header)
- Request/Response-Schemas (JSON Schema)
- Authentifizierung (OAuth 2.0, API Keys, JWT)
- Fehlercodes, Beispiele, Tags
- Erweiterungen (x-*) für vendor-spezifische Features

Beispielstruktur (YAML):

```yaml
openapi: 3.0.0
info:
  title: WebGIS API
  version: 1.0.0
paths:
  /features/{id}:
    get:
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
      responses:
        '200':
          description: Feature gefunden
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Feature'
components:
  schemas:
    Feature:
      title: Feature
      type: object
      properties:
        type:
          type: string
          enum:
            - Feature
        id:
          type: string
        properties:
          type: object
        geometry:
          $ref: '#/components/schemas/Geometry'
```

2.4.7.4 OpenAPI Tooling-Ökosystem

Das OpenAPI-Ökosystem umfasst ein reifes Spektrum an Tools, die den gesamten Contract-First-Workflow abdecken – von der Spezifikationserstellung über Validierung bis hin zur Code-Generierung und Dokumentation (Swagger/OpenAPI 2023).

OpenAPI Generator ist das Herzstück der bi-direktionalen Code-Generierung für Frontend und Backend.

Tabelle 11: OpenAPI relevante Bibliotheken

System	Generator	Ausgabe	WebGIS Beispiel
Frontend	*typescript-fetch, typescript-axios*	Typensichere React/Angular/Vue-Clients	`FeatureService.getLayer(id)` → automatische GeoJSON-Typen
Backend	*spring, go-server, nodejs-express*	Controller Interfaces + Models	Spring Boot `@RestController` mit Feature-Schema

Beispiel WebGIS-Generierung:

```
# Frontend: React/Vue.js/TypeScript Client
openapi-generator-cli generate -i webgis-api.yaml -g typescript-fetch -o
./webgis-client

# Backend: Spring Boot Server
openapi-generator-cli generate -i webgis-api.yaml -g spring -o ./webgis-
server
```

2.4.7.5 Relevanz für das Legacy WebGIS-Projekt

Für die Frontend-Modernisierung mit SPA-Frameworks (React, Angular, Vue.js) ermöglicht der OpenAPI 3.x Contract-first-Ansatz (vgl. Kapitel 2.4.7.2) eine unabhängige Frontend- und Backend-Entwicklung, indem er eine standardisierte Schnittstelle zwischen bestehenden Backend und modernen SPA-Clients definiert. Aus der zentralen OAS-Spezifikation lassen sich automatisiert typensichere TypeScript-Clients generieren, die GIS-Datenstrukturen wie GeoJSON-FeatureCollections konsistent abbilden und Runtime-Fehler eliminieren (Swagger/OpenAPI 2023). Swagger UI stellt standardisierte, interaktive Dokumentation für GIS-Teams bereit, während parallele REST-Endpunkte die Produktionsstabilität der Struts-Anwendung gewährleisten.

Langfristig schafft dieser Ansatz technische Unabhängigkeit und die Grundlage für skalierbare Microservices mit Spring Boot, ohne grundlegende Frontend-Anpassungen zu erfordern.

3 Einführung und Ausgangssituation

Dieses Kapitel gibt einen Überblick über aktuelle sowie kurz- und mittelfristige Maßnahmen im IT- und Organisationsbereich und leitet daraus Handlungsempfehlungen für die langfristige Weiterentwicklung der ISDB ab.

Zusätzlich wird hier die Ausgangssituation der ISDB detailliert im organisatorischen und technischen Kontext der Wiener Linien beschrieben. Damit bildet dies die Grundlage für die nachfolgende Analyse und die Entwicklung einer Modernisierungskonzeption. Das Ziel ist die nachvollziehbare Darstellung des aktuellen Ist-Zustands sowie der Rahmenbedienungen, unter denen die ISDB betrieben und weiterentwickelt wird.

Dabei wird bewusst auf eine detaillierte Betrachtung der Backend-Architektur und der zugrunde liegenden Geschäftslogik verzichtet, da der Schwerpunkt dieser Arbeit auf der Modernisierung der Weboberfläche und der Integration über REST-basierte Schnittstellen liegt.

3.1 Organisatorische Ist-Zustand

In den vergangenen Jahren wurden bei den Wiener Linien mehrere IT- und Organisationsprojekte initiiert, um interne Strukturen effizienter, transparenter und zukunftsfähiger zu gestalten. Im Zuge dieser Vorhaben entstanden verschiedene Fachsysteme und teils voneinander unabhängig geführte Datenbestände. Dazu zählen unter anderem die Fahrwegdatenbank als frühere Vorstufe der heutigen Infrastrukturdatenbank (ISDB), die Bescheid-Datenbank sowie das Wochenprogramm zur Fahrdienst- und Einsatzplanung.

Vor dem Hintergrund dieser historisch gewachsenen Systemlandschaft gewinnt die Vernetzung bestehender Anwendungen und die konsolidierte Nutzung vorhandener Daten zunehmend an Bedeutung. Eine stärkere Integration der Fachanwendungen unter Anwendung moderner IT-Architekturen und standardisierter Schnittstellen wird sowohl aus betriebswirtschaftlicher als auch aus technischer und organisatorischer Sicht als notwendig erachtet. Ziel dieser Integration ist es, betriebliche Abläufe zu optimieren, das Sicherheits- und Wartungsmanagement zu unterstützen, die Verfügbarkeit und Wartbarkeit der Systeme zu erhöhen sowie eine belastbare Informations- und Entscheidungsgrundlage für operative Fachaufgaben und strategische Planungsprozesse bereitzustellen.

In diesem Kontext nimmt die Infrastrukturdatenbank (ISDB) eine zentrale Rolle ein. Sie fungiert als fachliches Kernsystem zur Verwaltung gleisbezogener Infrastrukturdaten und bildet eine gemeinsame, organisationsübergreifend nutzbare Datenbasis. Die ISDB ist in der Abteilung ***I62d – Datenmanagement*** der ***Hauptabteilung I6 – Infrastrukturmanagement*** organisatorisch verankert. Dort wird das System operativ betrieben und durch ein internes Softwareentwicklungsteam kontinuierlich weiterentwickelt.

Diese Anwendung wird konzernweit eingesetzt und verzeichnet aktuell rund 2.500 aktive Nutzer*innen innerhalb der Wiener-Stadtwerke-Gruppe. Neben den Wiener Linien pflegen auch die Wiener Lokalbahnen eigene Datenbestände in der ISDB, einschließlich getrennter Benutzer- und Gruppenstrukturen sowie über 700 fachspezifischer GIS-Layer. Derzeit umfasst der Datenbestand der Wiener Linien etwa 60 bis 70 fachliche Datensätze, jener der Wiener Lokalbahnen rund 40 bis 45 Datensätze.

Insgesamt stellt die ISDB im aktuellen Ist-Zustand ein etabliertes, konzernweit genutztes Fachsystem dar, das zentrale infrastrukturelle Informationen bündelt und die Zusammenarbeit zwischen unterschiedlichen Organisationseinheiten unterstützt. Diese Ausgangssituation bildet zugleich die Grundlage für die weiterführende Analyse zur historischen Entwicklung und zur Modernisierung der bestehenden Anwendung.

Die folgende Darstellung beschreibt die historische Entwicklung der ISDB, die maßgeblich durch den Bedarf nach einer konsistenten, prozessnahen Datenhaltung und die technische Weiterentwicklung der Systemlandschaft geprägt wurde.

3.1.1 Historische Entwicklung der ISDB

Die heutige Infrastrukturatenbank (ISDB) ist nicht als klassisches GIS-System konzipiert worden, sondern entstand als Reaktion auf die begrenzte Aussagekraft isolierter technischer Messdaten. Erste Messfahrzeuge der ende 1995er-Jahre lieferten zwar umfangreiche Zustandswerte, diese waren jedoch ohne räumlichen und infrastrukturellen Kontext nur eingeschränkt interpretierbar. Daraus ergab sich die Anforderung, das Gleisnetz sowie zugehörige Infrastrukturelemente digital, referenzierbar und prozessnah abzubilden (Kail, 2009-2012).

Frühere Versuche, diese Anforderungen mit Desktop-GIS-Lösungen umzusetzen, erwiesen sich aufgrund hoher Bedienkomplexität, eingeschränkter Prozessintegration und technischer Abhängigkeiten als nicht nachhaltig. Daher wurde zu Beginn der 2000er-Jahre mit der **Fahrwegdatenbank** (FWDB) eine erste datenbankgestützte Fachanwendung für die strukturierte Erfassung und Pflege von Gleisanlagen, Geometrien und Zustandsinformationen entwickelt (Fiby, 2004). Ihr Schwerpunkt lag auf der systematischen Dokumentation technischer Infrastrukturelemente und deren Attributierung zur Unterstützung von Instandhaltungs- und Planungsprozessen. Die heutige ISDB ist das Ergebnis einer kontinuierlichen Weiterentwicklung dieser frühen, datenorientierten Systeme.

Mit zunehmender Komplexität der Systemlandschaft und steigenden Anforderungen an Integration, Aktualität und Auswertbarkeit der Daten setzte ab 2009 ein strategischer Paradigmenwechsel im Datenmanagement ein. In diesem Zusammenhang wurde ein komponentenbasiertes Architektur- und Datenhaltungskonzept entwickelt, das die Vereinheitlichung von Referenzdaten, die strukturierte Erfassung von Zustandsinformationen und die Einbindung fachlicher Prozesse in den Mittelpunkt stellte (Kail, 2009-2012). Ziel war es, bestehende Datensilos zu konsolidieren,

Redundanzen zu vermeiden und eine nachhaltige, integrierte Grundlage für die zukünftige Systementwicklung zu schaffen.

In den folgenden Jahren entwickelte sich die ISDB zu einem zentralen Informations- und Referenzsystem für gleis- und ortsbezogene Infrastrukturdaten. Sie wurde schrittweise in eine übergreifende digitale Systemarchitektur eingebettet, in der mehrere Kernapplikationen über definierte Schnittstellen miteinander interagieren. Neben technischen Aspekten beeinflussen inzwischen auch organisatorische, rechtliche und datenschutzrelevante Rahmenbedingungen den Betrieb und die Weiterentwicklung des Systems maßgeblich (Betriebsvereinbarung B6 Digitale Systemarchitektur, 2022).

3.1.2 ISDB in der Systemlandschaft

Die Infrastrukturdatenbank (ISDB) fungiert als eines der zentralen Fachdatensysteme der Wiener Linien zur Verwaltung gleis- und infrastruktureller Referenzdaten und ist fest in die digitale Systemlandschaft des Konzerns eingebettet. Die Nutzung erfolgt überwiegend durch interne Fachabteilungen – insbesondere in den Bereichen Planung, Betrieb und Instandhaltung – und dient dort als verbindliche Informationsgrundlage für operative und strategische Entscheidungsprozesse.

Zur kartografischen Darstellung nutzt sie die von der WienIT bereitgestellten Geoservices und Basiskarten-Layer (Orthofotos, Mehrzweckkarten, Naturbestandsdaten). Diese ermöglichen die räumlich präzise Verortung von Gleisachsen, Segmenten und Infrastrukturobjekten innerhalb der integrierten WebGIS-Oberfläche und stellen damit die fachliche Grundlage für Planungs-, Wartungs- und Analyseprozesse dar.

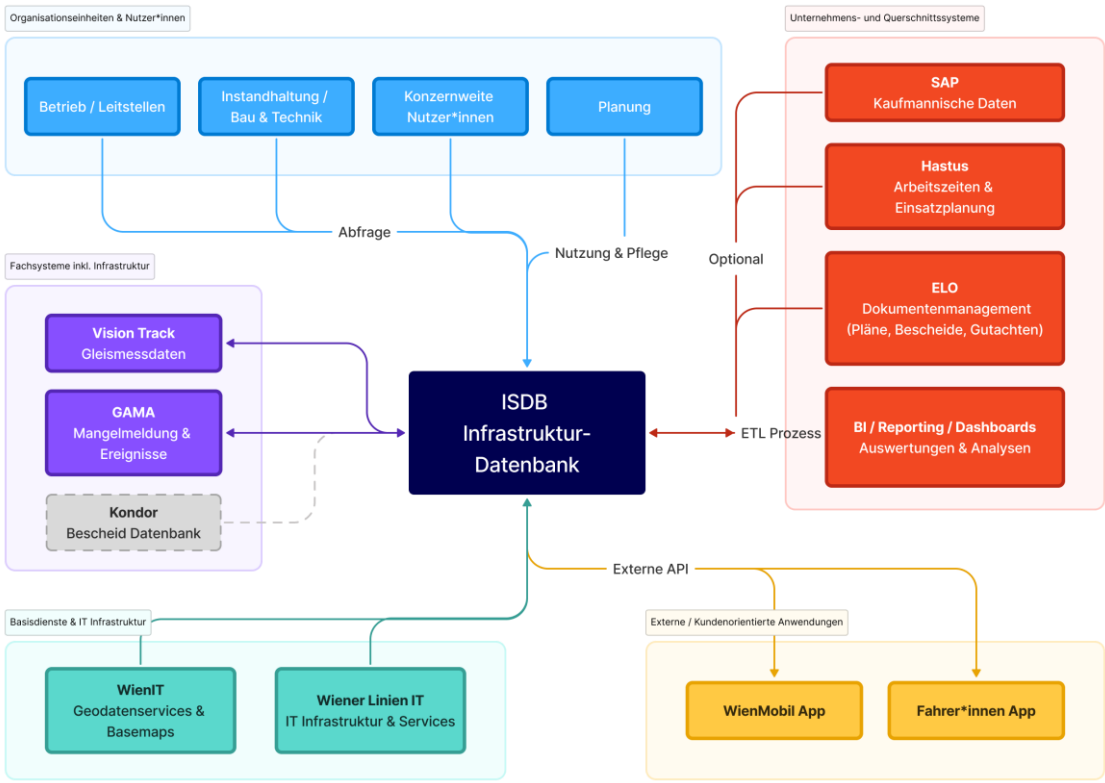


Abbildung 5: ISDB in die Systemlandschaft der Wiener Linien

Konzernweit dient die ISDB als verlässliche Datenquelle („Golden Source“) für gleis- und objektbezogene Infrastrukturdaten. Sie liefert konsistente und geprüfte Informationen für statistische Auswertungen, **Reportings** und **Business-Intelligence** (BI) Dashboards sowie für nachgelagerte Fachanwendungen. Zahlreiche digitale Services der Wiener Linien – darunter interne Analysewerkzeuge ebenso wie kundenseitige Applikationen wie die WienMobil-App – beziehen über definierte Schnittstellen aktuelle georeferenzierte Infrastrukturdaten inkl. Fachinformationen direkt aus der ISDB. Zudem ist die ISDB operativ in Prozesse eingebunden. Über MobGIS Anwendung werden automatisiert z.B. die Mangelmeldungen von Gleis- und Weichanlagen übernommen, objektbezogen klassifiziert und mit ihrer Geometrie verknüpft. Diese Kopplung von Fach- und Raumdaten ermöglicht eine konsistente, lebenszyklusorientierte Betrachtung der Gleisinfrastruktur und bildet eine zentrale Grundlage für Instandhaltungs- und Entscheidungsprozesse.

Abb. 5 zeigt die Einbettung der ISDB in die Systemlandschaft der Wiener Linien. Sie fungiert dabei als Fachdatenbank für gleis- und ortsbezogene Infrastrukturdaten und ist über definierte Schnittstellen mit Prozess-, Analyse- und Dokumentationssystemen sowie mit konzernweiten Basisdiensten und diversen kundenorientierten Anwendungen verbunden.

3.2 Infrastrukturdatenbank – ihre Funktionen und Rollen

Wie oben erläutert übernimmt die ISDB als referenzführendes Fachsystem die zentrale Rolle für geometrische, technische und fachlich strukturierte Geoinformationen zu Gleisanlagen. Sie dient als verbindliche Informationsquelle für Planung, Betrieb und Instandhaltung und bildet die Grundlage für eine konsistente, lebenszyklusorientierte Betrachtung der Gleisinfrastruktur.

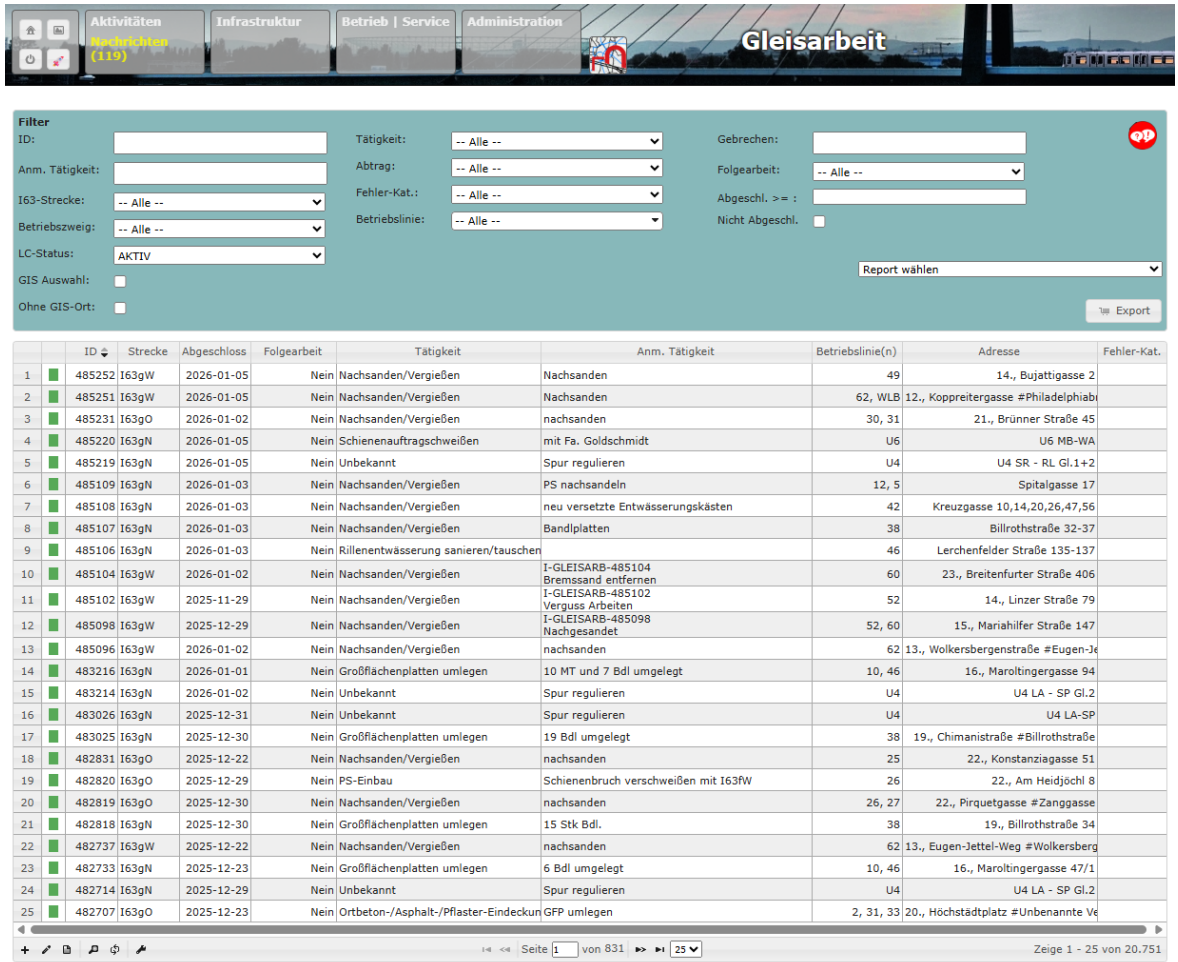
Abbildung 6: Aktuelle Startseite der ISDB

Die browserbasierte WebGIS Anwendung (siehe Abb. 6) ist über das interne Netzwerk der Wiener Linien erreichbar und wird überwiegend von Fachabteilungen der Bereiche Planung, Betrieb und Instandhaltung genutzt. Innerhalb der vernetzten Systemlandschaft fungiert die ISDB als zentrale Referenz- und Datendrehscheibe für gleis- und ortsbezogene Infrastrukturdaten, während die fachliche Prozessausführung in angebundenen Systemen erfolgt. Der inhaltliche Fokus dieser Arbeit liegt auf den Prozessen und Datenbestände der Hauptabteilung **Infrastrukturmanagement I6** der Wiener Linien sowie auf angrenzenden Bereichen.

Die Datengrundlagen ergeben sich weitgehend automatisiert aus der elektronisch unterstützten Prozessführung des operativen Bereichs, dass für die Aufbereitung der Daten nur in begrenztem Umfang manuelle Tätigkeiten erforderlich sind.

3.2.1 Datenerfassung für Übersichten und Auswertungen

Die fachspezifischen Datensätze werden von den jeweils zuständigen Fachabteilungen auf Basis relevanter Prozess- und Betriebsdaten systematisch erfasst, konsolidiert, strukturiert aufbereitet und fortlaufend aktualisiert. Die Datenerhebung geht dabei über die Anforderungen der operativen Prozessabwicklung hinaus und ist gezielt auf die Erstellung von Übersichten sowie auf Auswertungs- und Analysezwecke ausgerichtet.



		ID	Strecke	Abgeschloss	Folgearbeit	Tätigkeit	Anm. Tätigkeit	Betriebslinie(n)	Adresse	Fehler-Kat.
1		485252	I63gW	2026-01-05	Nein	Nachsanden/Vergießen	Nachsanden	49	14., Bujattigasse 2	
2		485251	I63gW	2026-01-05	Nein	Nachsanden/Vergießen	Nachsanden	62, WLB	12., Koppreitergasse #Philadelphiabı	
3		485231	I63gO	2026-01-02	Nein	Nachsanden/Vergießen	nachsanden	30, 31	21., Brünner Straße 45	
4		485220	I63gN	2026-01-05	Nein	Schienenauftragschweißen	mit Fa. Goldschmidt	U6	U6 MB-WA	
5		485219	I63gN	2026-01-05	Nein	Unbekannt	Spur regulieren	U4	U4 SR - RL Gl.1+2	
6		485109	I63gN	2026-01-03	Nein	Nachsanden/Vergießen	PS nachsandeln	12, 5	Spitalgasse 17	
7		485108	I63gN	2026-01-03	Nein	Nachsanden/Vergießen	neu versetzte Entwässerungskästen	42	Kreuzgasse 10,14,20,26,47,56	
8		485107	I63gN	2026-01-03	Nein	Nachsanden/Vergießen	Bandlplatten	38	Billrothstraße 32-37	
9		485106	I63gN	2026-01-03	Nein	Rillenentwässerung sanieren/tauschen		46	Lerchenfelder Straße 135-137	
10		485104	I63gW	2026-01-02	Nein	Nachsanden/Vergießen	I-GLEISARB-485104 Bremssand entfernen	60	23., Breitenfurter Straße 406	
11		485102	I63gW	2025-11-29	Nein	Nachsanden/Vergießen	I-GLEISARB-485102 Verguss Arbeiten	52	14., Linzer Straße 79	
12		485098	I63gW	2025-12-29	Nein	Nachsanden/Vergießen	I-GLEISARB-485098 Nachgesandet	52, 60	15., Mariahilfer Straße 147	
13		485096	I63gW	2026-01-02	Nein	Nachsanden/Vergießen	nachsanden	62	13., Wolkersbergenstraße #Eugen-Je	
14		483216	I63gN	2026-01-01	Nein	Großflächenplatten umlegen	10 MT und 7 Bdl umgelegt	10, 46	16., Maroltingergasse 94	
15		483214	I63gN	2026-01-02	Nein	Unbekannt	Spur regulieren	U4	U4 LA - SP Gl.2	
16		483026	I63gN	2025-12-31	Nein	Unbekannt	Spur regulieren	U4	U4 LA-SP	
17		483025	I63gN	2025-12-30	Nein	Großflächenplatten umlegen	19 Bdl umgelegt	38	19., Chimanistraße #Billrothstraße	
18		482831	I63gO	2025-12-22	Nein	Nachsanden/Vergießen	nachsanden	25	22., Konstanziagasse 51	
19		482820	I63gO	2025-12-29	Nein	PS-Einbau	Schienenbruch verschweißen mit I63fW	26	22., Am Heidjöchl 8	
20		482819	I63gO	2025-12-30	Nein	Nachsanden/Vergießen	nachsanden	26, 27	22., Pirquetgasse #Zanggasse	
21		482818	I63gN	2025-12-30	Nein	Großflächenplatten umlegen	15 Stk Bdl.	38	19., Billrothstraße 34	
22		482737	I63gW	2025-12-22	Nein	Nachsanden/Vergießen	nachsanden	62	13., Eugen-Jettel-Weg #Wolkersberg	
23		482733	I63gN	2025-12-23	Nein	Großflächenplatten umlegen	6 Bdl umgelegt	10, 46	16., Maroltingergasse 47/1	
24		482714	I63gN	2025-12-29	Nein	Unbekannt	Spur regulieren	U4	U4 LA - SP Gl.2	
25		482707	I63gO	2025-12-23	Nein	Ortbeton-/Asphalt-/Pflaster-Eindeckun	GFP umlegen	2, 31, 33	20., Höchstädtplatz #Unbenannte Ve	

Seite 1 von 831 — Zeige 1 - 25 von 20.751

Abbildung 7: Überblickseite eines Datensatzes (Gleisarbeit)

Zur Zusammenführung von Daten aus verschiedenen Quellsystemen werden produktauflaufend eingesetzte Data-Warehouse-Strukturen (DWH) mittels „Extract-Transform-Load"-Prozessen (ETL) genutzt, die einheitlichen Zugriff ermöglichen. Die so konsolidierten Daten dienen der operativen Entscheidungsunterstützung, taktischen Auswertungen sowie als verlässliche Grundlage für Analysen und Studien – insbesondere zum Personal- und Ressourceneinsatz.

3.2.2 Datenmodell – Grundkonzept

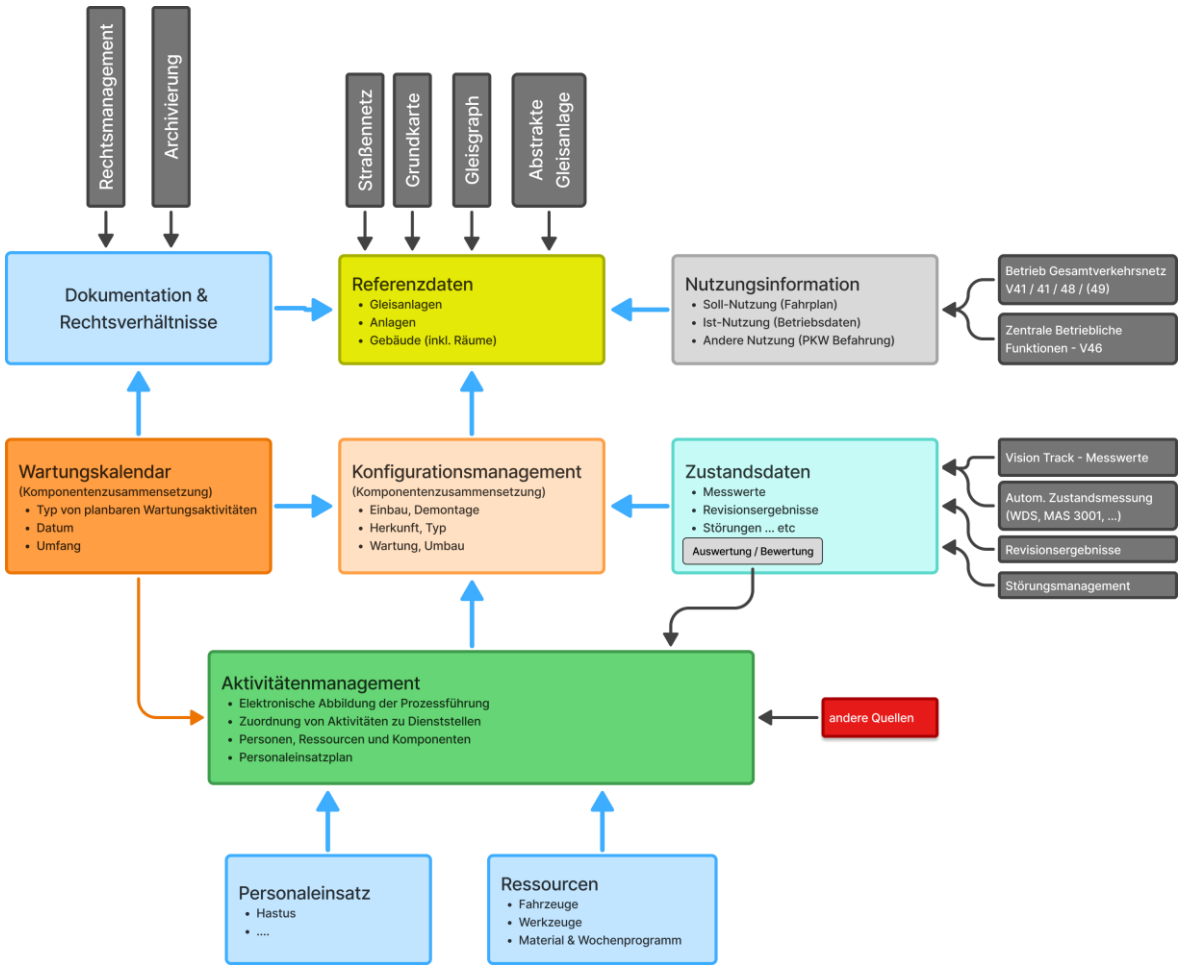


Abbildung 8: Fachliches Komponentenmodell

Die Abb.8 zeigt ein fachliches Komponentenmodell zur Veranschaulichung der bestehenden Prozessführung, das zentrale Funktionsbereiche wie Referenzdaten, Zustands- und Nutzungsinformationen, Konfigurations- und Aktivitätenmanagement sowie unterstützende Bereiche (z.B. Dokumentation, Wartung, Personal und Ressourcen) umfasst.

Diese Funktionsblöcke sind über fachliche Schnittstellen verbunden, wie sie auch in der Organisation durch Zuständigkeiten und Verantwortungsbereichen abgebildet werden. Das Modell stellt bewusst keine technischen Systemgrenzen dar, sondern bildet die inhaltlich-fachliche Struktur der Prozessunterstützung ab. Die aktuelle Ausprägung hinsichtlich Integration, Durchgängigkeit und Auswertung ist jedoch nicht ausreichend, um den zukünftigen fachlichen und organisatorischen Anforderungen gerecht zu werden.

Im Folgenden werden die gezeigten Komponenten beschrieben:

3.2.2.1 Referenzdaten

Referenzdaten sind Beschreibungen und Attributinhalte grundlegender Einheiten, auf die sich die Tätigkeiten von I6 Abteilungen beziehen. Konkret umfassen sie Gleisverläufe, Stationseinrichtungen, Streckenabschnitte u. ä. Diese Daten dienen der eindeutigen Identifikation und Benennung von Objekten, treffen jedoch keine Aussagen zu deren Art, Ausführung oder Herkunft. Ihr Hauptzweck besteht darin, ein konsistentes datentechnisches Gerüst zu schaffen, auf das sich alle weiteren Datenobjekte beziehen können. Referenzdaten werden grundsätzlich extern an das System geliefert und nicht systemintern generiert. Die Komponente ist so ausgelegt, dass Änderungen versioniert werden und vergangene Datenstände nachvollziehbar bleiben.

Dazu gehören unter anderem:

- Digitale Darstellung des Wiener Straßennetzes mit Hausnummern
- Grundkarten (als Grundlage für die Darstellung weiterer Objekte. vgl. Kapitel 3.1.2)
- Gleisanlage und -graph (Gleisachsen, koordinativer Verlauf, Stationierung)
- Stationen (Namen, Ort)
- Streckenabschnitte (Namen, zuständige Dienststellen)

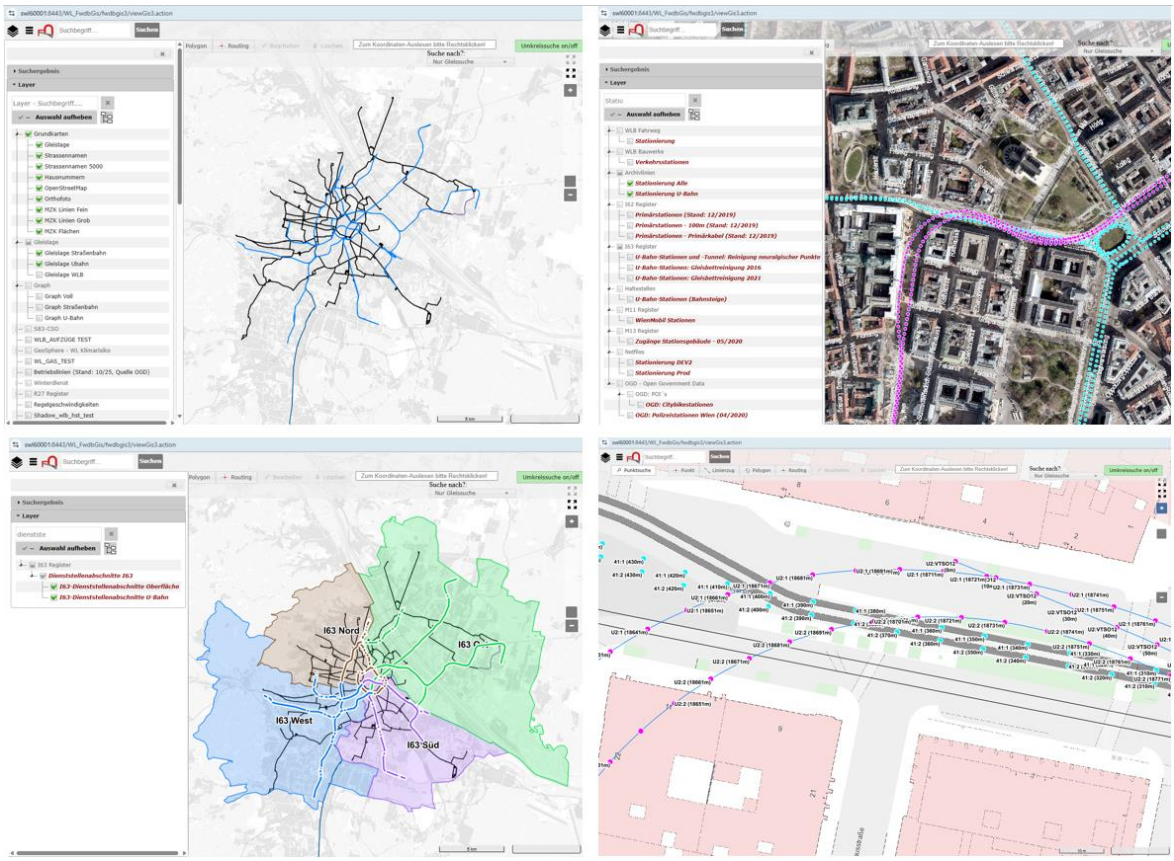


Abbildung 9: Gleisgraph GIS-Layers als Referenzdaten dargestellt auf einer Basemap

Der konkrete Umfang richtet sich nach der tatsächlichen Nutzung des Systems und dem jeweiligen Bedarf. Änderungen erfolgen stets außerhalb der Anwendung in übergeordnet geprüfter Form. Vorgesehen sind insbesondere Neuanlage, Löschung, Umbenennung sowie Änderungen von Ort oder Verlauf. Bestimmte Änderungen lösen Prozesse aus, die sich mit der Adaptierung der Daten in allen Komponenten auf die Referenzdaten beziehen.

3.2.2.2 Konfigurationsmanagement

Konfigurationsmanagement beschreibt, wie Komponenten des Tätigkeitsgegenstands (Elemente/ Objekte) aktuell erfasst und definiert sind. In diesem Bereich werden Informationen zu Art, Typ, Bauweise und Lebenszyklus (z.B. Herstellung, Revision, Einbau, Ausbau, Herkunft) verwaltet. Jedes Element erhält im Rahmen der bestehenden Aufzeichnungen einerseits einen Bezug zu einem Referenzdatenobjekt und enthält andererseits lebenszyklusbezogene Informationen, die aus anderen Komponenten ableitbar sind.

Derzeit werden zwei grundlegende Arten von Konfigurationselementen unterschieden. Konkrete Elemente sind Komponenten, die tatsächlich als Bauteil oder Bauart existieren, wie etwa Schienen, Unterbauarten, Tunnel, Rückleiteranschlüsse oder Stromverteiler. Abstrakte Elemente existieren, um Bezüge zwischen anderen Komponenten herzustellen. Dazu zählen beispielsweise Planabschnitte, Baustellen oder Messabschnitte.

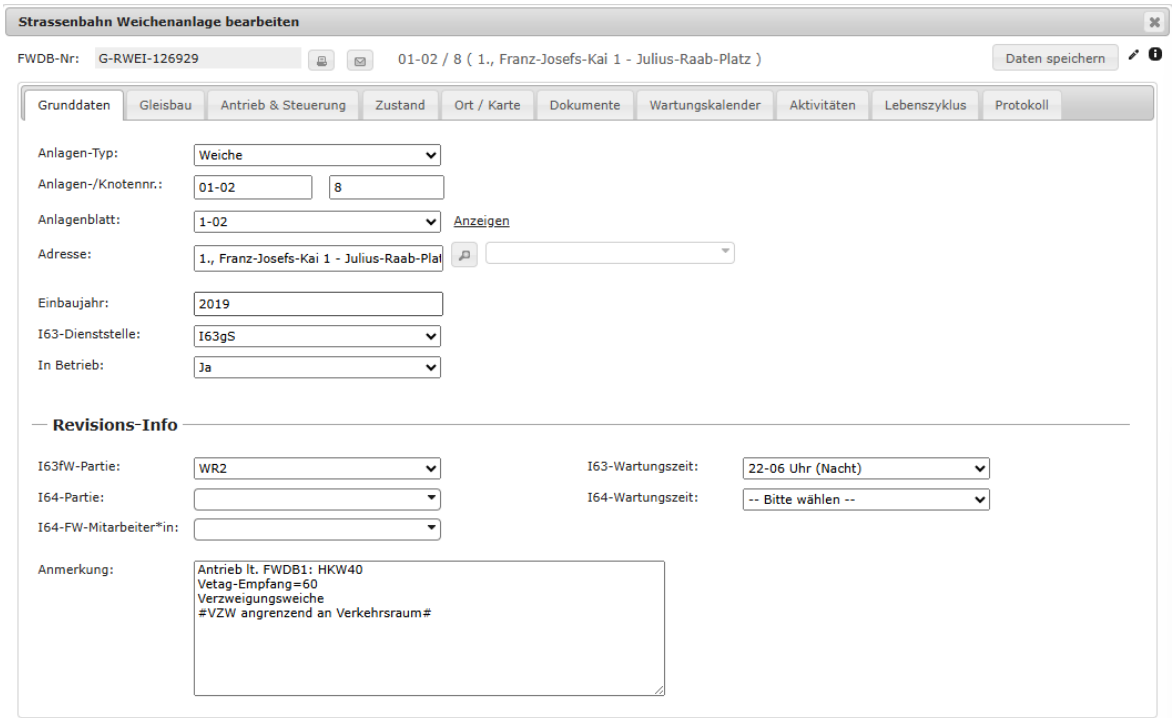


Abbildung 10: Weboberfläche zum Konfigurationsmanagement für Komponentenparameter

Änderungen an Konfigurationselementen erfolgen im Ist-Zustand ausschließlich im Rahmen definierter und geordneter Prozesse. Daten zu Konfigurationselementen weisen dabei stets einen zeitlichen Bezug auf, um Änderungen über den Lebenszyklus nachvollziehbar abzubilden.

3.2.2.3 Nutzungsinformationen

Nutzungsinformationen beschreiben, wann, wie und in welcher Art Anlagen, beispielsweise Gleisanlagen, genutzt werden. Dazu zählen insbesondere Fahrplandaten als Soll-Nutzung sowie – sofern verfügbar – Betriebsdaten als Ist-Nutzung. Ergänzend können vergleichbare externe Daten berücksichtigt werden, etwa Informationen zur PKW-Befahrung von Gleisanlagen oder zu Witterungseinflüssen.

Die Nutzungsinformationen werden im Wesentlichen aus der Abteilung V4 – Betrieb des Gesamtverkehrsnetzes sowie aus zentralen betrieblichen Funktionen bereitgestellt. Die Komponente verwaltet die Nutzungsdaten und stellt diese für Informations- und Auswertungszwecke zur Verfügung. Sämtliche Nutzungsdaten sind mit einem zeitlichen Bezug versehen.

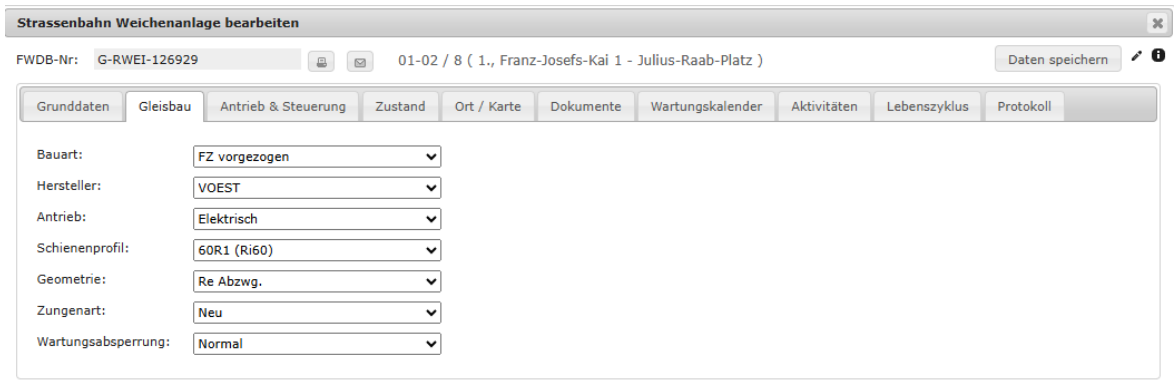


Abbildung 11: Detaillierte Nutzungsinformation einer Weichenanlage

Die Nutzungsdaten werden von externen Systemen geliefert. Änderungen an Nutzungsdaten lösen grundsätzlich keine automatischen Anpassungsprozesse in anderen Datenkomponenten aus.

3.2.2.4 Zustandsdaten

Zustandsdaten sind Daten, die den Erhaltungszustand oder die Benutzbarkeit von Anlagen und Einrichtungen beschreiben. Das sind unter anderem Messwerte, Inspektionsergebnisse, Daten auch automatischen Störungs- und Betriebserfassungssystemen sowie Daten über Vorkommnisse bei der Nutzung z.B. Betriebsstörung.

Die Aufgabe dieser Komponente besteht darin, diese Informationen zu sammeln und in einer im Hinblick auf Referenzdaten und Konfigurationsmanagement konsistenten Form zu speichern und für auf Auswertungen zur Verfügung zu halten. Dazu werden die Daten periodisch rechnerisch ausgewertet und bei Bedarf automatisch oder manuell das Aktivitätenmanagement angesteuert. Die Zustandsdaten haben grundsätzlich einen zeitlichen Bezug und werden von außen geliefert.

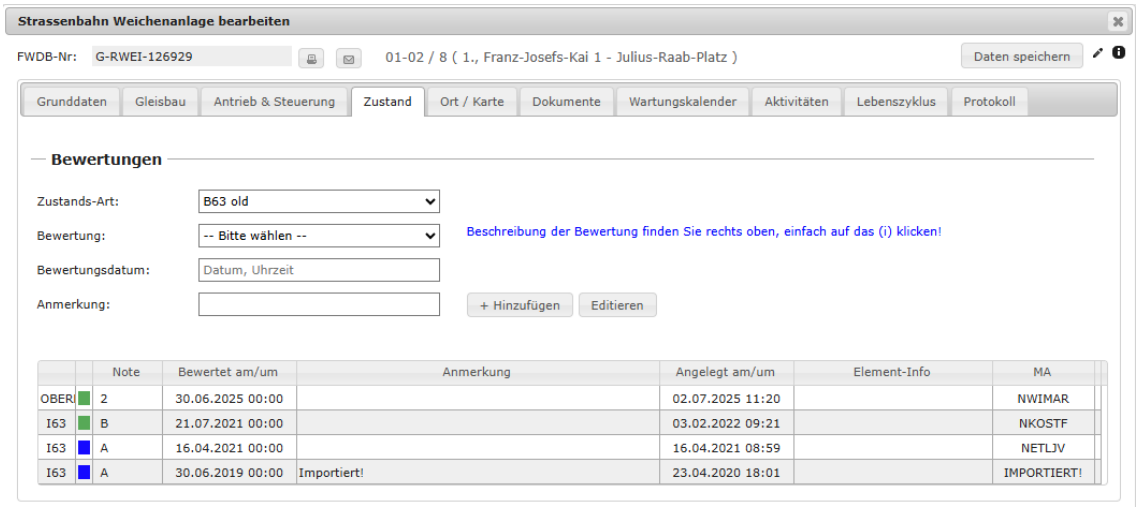


Abbildung 12: Zustandsdaten einer Weichenanlage inkl. Bewertungsnoten sowie Datum

3.2.2.5 Räumliche Verortung

In dieser Komponente erfolgt die räumliche Verortung der jeweiligen Elementobjekte. Abhängig von der Objektart werden unterschiedliche, vordefinierte GIS-Funktionalitäten zur Verfügung gestellt.

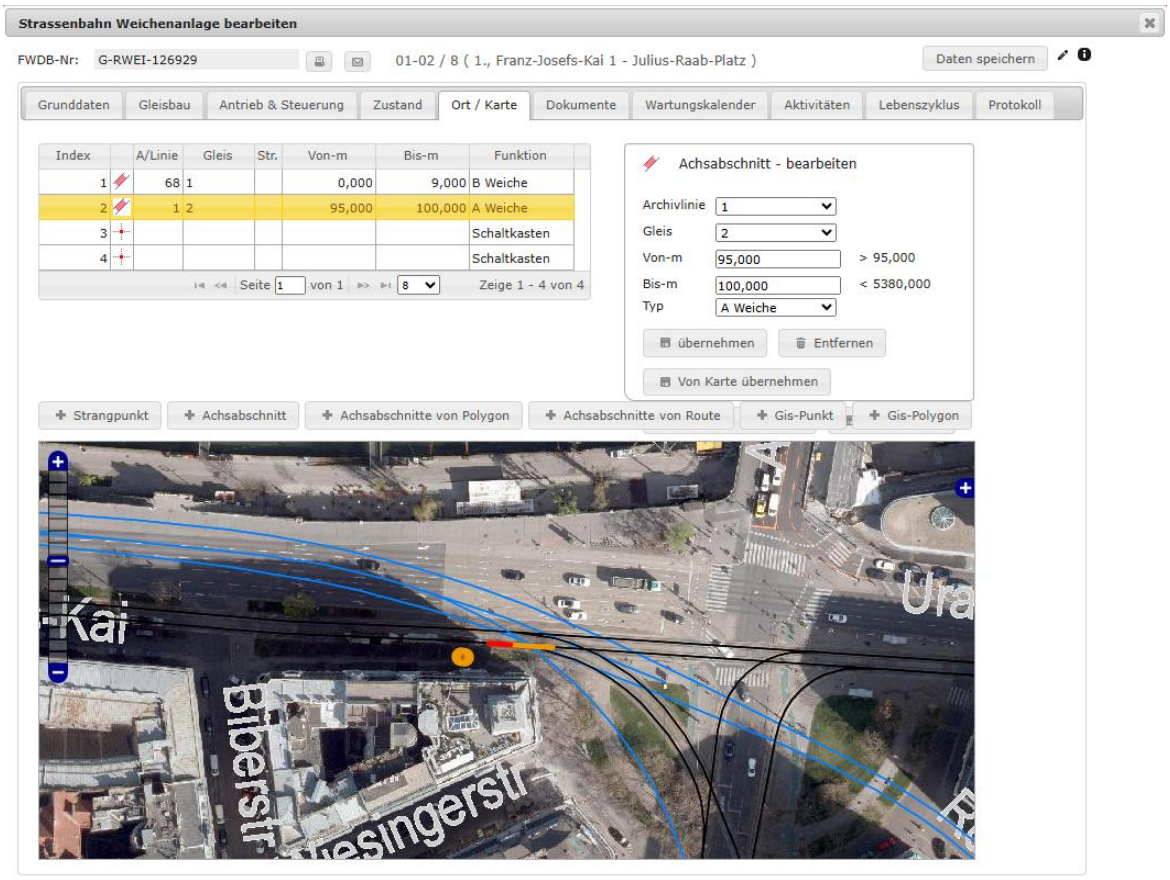


Abbildung 13: GIS-Verortung von Infrastrukturkomponenten anhand Gleisgraph-Referenzdaten mit metergenauer Präzision (ISDB)

Die Objektarten unterscheiden sich insbesondere danach, ob es sich um gleisbezogene oder ortsbezogene Elemente handelt. Gleisbezogene Elemente, wie beispielsweise Weichenanlagen, werden über Referenzen auf Achsabschnitte eines Gleises oder auf Strangabschnitte verortet. Ortsbezogene Elemente, wie etwa Schaltkästen, werden hingegen über eine direkte geographische Verortung abgebildet, beispielsweise durch GIS-Punkte zur Darstellung des exakten Standorts oder durch Polygone zur Abgrenzung einer Gesamtanlage. Beide Verortungsarten basieren maßgeblich auf den zugrunde liegenden Referenzdaten und sind eng mit diesen verknüpft.

3.2.2.6 Dokumentation und Rechtsverhältnisse

Mit dieser Komponente ist die Anbindung an den Bereich gemeint, in dem technisch und rechtlich relevante Dokumente in unternehmensinternen digitalen Dokumentenmanagementsystem (DMS) archiviert werden und in denen die rechtlichen Verhältnisse (Vorschriften, Bescheide und deren Auflagen, Normen usw..) behandelt und verwaltet werden. Eine Aufgabe dieser Anbindung ist, für jedes Element des Konfigurationsmanagements einen Verweis auf Pläne und andere Konstruktions- und bautechnischen Unterlagen herzustellen und das für das Element anzuwendende Recht (Gesetze, Verordnungen, Bescheide, Normen) zu hinterlegen. Dazu muss diese auch in der Lage sein, dokumentartige Daten (z.B. Inspektionsprotokolle, Messdatenaufzeichnungen, Baupläne, Detailanleitungen, …) aufzunehmen und bei Bedarf wiederzugeben.

Abbildung 14: Dokumentenarchivierung und -auflistung im DMS für Konfigurationsmanagement

Jedes Dokument kann über spezifischen Key (z.B. FWDB-Nr: G-RWEI-126929) des Konfigurationsmanagements referenziert werden. Aufbau, Inhalt und Verwaltung der über diese Anbindung gelieferten Daten wird durch Archiv-Prozess überwacht und gestaltet.

3.2.2.7 Wartungskalendar

Der Inspektionskalendar oder Wartungskalendar ist eine Komponente, die alle regelmäßig erforderlichen Inspektionen, Wartungen oder Revisionen von Elementen des Konfigurationsmanagements in Evidenz hält, gemeinsam darstellt und die Ergebnisse der Inspektionen aufzeichnet. Diese Komponente verwaltet insbesondere auch den Inspektionszyklus und die mögliche Vorverlegung und Verzögerung von Inspektionen.

Abbildung 15: Wartungskalender-Tabelle: Tätigkeitsbezeichnung, letztes/nächstes Datum, Status

Die Aufgabe dieser Komponente ist es, einen Überblick über alle regelmäßig anfallenden Inspektionen und Wartungsarbeiten sowie den dabei zu erwartenden Aufwand zu geben, die Vorbereitung der Inspektionen zu unterstützen und die Ergebnisse aufzuzeichnen.

3.2.2.8 Aktivitätenmanagement

Das Aktivitätenmanagement ist eine Komponente, die elektronisch Prozessführung ermöglicht. Die Prozesse haben idealerweise einen oder mehrere Bezüge zu Konfigurationsmanagement-Elementen. Im Aktivitätenmanagement werden alle wesentlichen Schritte des Lebenszyklus einer Aktivität abgedeckt.

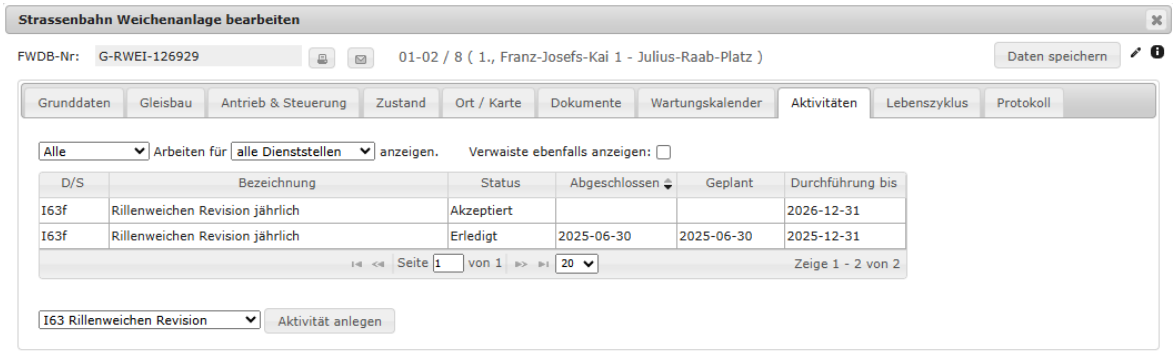


Abbildung 16: Aktivitätenmanagement-Tabelle: Tätigkeitslebenszyklus, Planung, Zuordnung, Status

Eine Aktivität beschreibt eine konkrete, fachliche Tätigkeit, wie z.B. eine Revisionsprüfung, eine Instandhaltungs-, Wartungs- oder Reparaturmaßnahme. Aktivitäten sind grundsätzlich planungsrelevant und werden in dieser Komponente strukturiert erfasst und gesteuert. Die Entstehung von Aktivitäten erfolgt dabei aus unterschiedlichen Anlässen. Einerseits ergeben sich Aktivitäten aus festgelegten, intervallbasierten Vorgaben, etwa regelmäßigen Inspektionen von Straßenbahn-Weichenanlagen, die beispielsweise zweimal jährlich durchzuführen sind. Andererseits können Aktivitäten ereignisgetrieben entstehen, etwa durch Störungsmeldungen, Mängelmeldungen oder Ergebnisse aus Zustandsüberwachungen und Revisionen. Das betrifft insbesondere Erzeugung einer Aktivität, Planung, Zuordnung von Mitarbeitern und Ressourcen, Durchführung, Abschluss und Kontrolle.

Neben der Planung, Evidenzhaltung und Verteilung von Aktivitäten sollen die Umstände der Durchführung der Aktivitäten möglichst genau erfasst werden. Das bedeutet

insbesondere, dass der Personal- und Materialaufwand sowie Leistungen durch Dritte genau aufgezeichnet werden. Das bildet die Basis für die Lebenszykluskostenermittlung und liefert Grundlagen für Optimierungsüberlegungen.

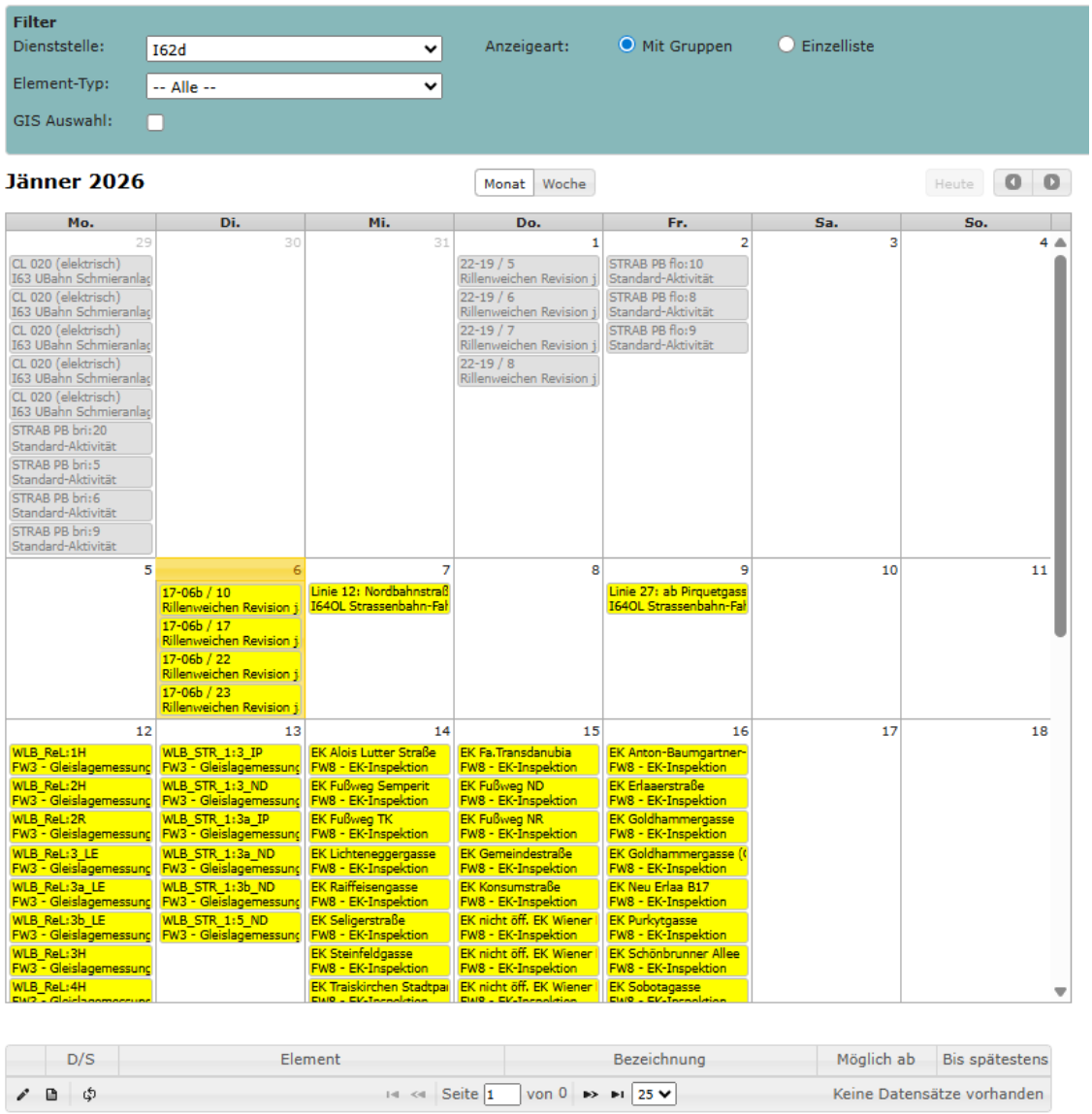


Abbildung 17: Kalenderansicht des Aktivitätenmanagements der ISDB mit Darstellung geplanter und laufender Infrastrukturmaßnahmen

Im Unterschied zu den anderen Komponenten, die primär tabellarische Bestandsdaten verwalten und statische Zustandssituationen beschreiben, ist das Aktivitätenmanagement als dynamische Prozesskomponente konzipiert. Es enthält hinterlegte Prozessintervall-Templates und Workflows, auf deren Basis Aktivitäten erzeugt, gesteuert und abgewickelt werden. Das Aktivitätenmanagement zeigt zudem eigenständige Systemfähigkeiten, etwa durch automatisierte Eskalationsmechanismen bei Fristüberschreitungen oder kritischen Ereignissen.

3.2.2.9 Lebenszyklus und Protokollierung

Der Bereich Lebenszyklus dient der Abbildung des aktuellen Status eines Elements sowie dessen zeitlicher Gültigkeit. Er zeigt insbesondere an, ob ein Element aktiv oder deaktiviert ist und seit welchem Zeitpunkt das Element gültig ist. Die Deaktivierung eines Elements kann durch Benutzer mit entsprechenden administrativen Berechtigungen vorgenommen werden.

Abbildung 18: Lebenszyklus-Management – Aktuelle Status, Gültigkeitsdaten und Deaktivierungsfunktion

Zusätzlich besteht die Möglichkeit, beim Anlegen eines neuen Elements bestehende Inhalte eines anderen Elements zu übernehmen. Diese Funktion unterstützt die effiziente Neuerfassung von Elementen und wird insbesondere von den Fachabteilungen genutzt und nachgefragt.

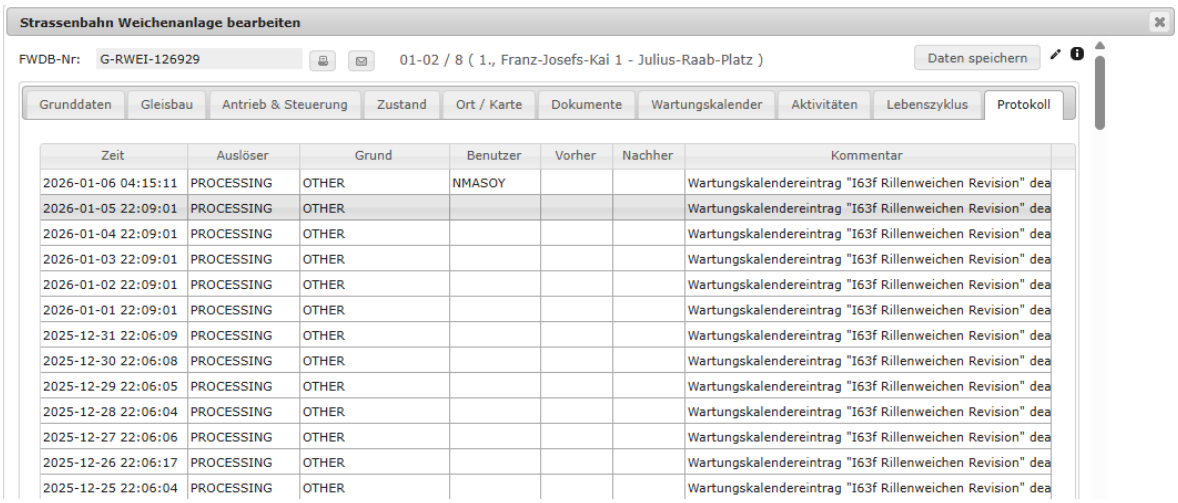


Abbildung 19: Protokoll- und Lebenszyklusübersicht – Historische Verarbeitungsdaten mit Audit-Tracking

Die Protokollierung erfolgt auf Basis definierter Audit-Regelungen. Dabei wird nachvollziehbar aufgezeichnet, welcher Benutzer zu welchem Zeitpunkt welche Änderungen vorgenommen hat. Die Protokollierung dient der Sicherstellung der Nachvollziehbarkeit von Änderungen sowie der Erfüllung von Revisions-, Rechts- und Richtlinienanforderungen.

4 Strategische Planung und Zielarchitektur

Nach der Analyse des Ist-Zustands der ISDB sowie der relevanten technologischen Grundlagen in den vorhergehenden Kapiteln wendet sich dieses Kapitel der strategischen Planung und der Definition der Zielarchitektur zu. Im Mittelpunkt steht die Herleitung zentraler architektonischer Leitentscheidungen für die Modernisierung der bestehenden WebGIS-Anwendung sowie deren fachliche und technische Begründung. Darüber hinaus werden die Erfahrungswerte dieser Entscheidungen für die nachfolgende Umsetzung der Frontend-Modernisierung und der API-Architektur aufgezeigt. Ziel ist es, eine langfristig tragfähige technische Grundlage für die Weiterentwicklung der Anwendung zu schaffen.

4.1 Zielsetzung der Modernisierungsstrategie

Die strategische Zielsetzung der Modernisierung ist der Übergang von einer veralteten, monolithischen Server-Side-Rendering (SSR) Architektur auf Basis von Apache Struts zu einer entkoppelten, serviceorientierten Systemstruktur mit einem clientseitigen Single-Page-Application-(SPA-)Frontend (vgl. Kapitel 2.4.5) und einer klar definierten API-Struktur (vgl. Kapitel 2.4.7). Zentrales Ziel ist die konsequente Trennung von Geschäftslogik und Darstellungsschicht, um Wartbarkeit, Integrationsfähigkeit und langfristige Zukunftssicherheit der WebGIS-Anwendung ISDB zu gewährleisten.

4.1.1 Technischer Ist-Zustand und Problemstellung

Die bestehender WebGIS Anwendung, ISDB basiert auf einer klassischen, serverseitig gerenderten Architektur mit Apache Struts, die durch eine enge Kopplung von Präsentationslogik, Steuerungslogik und fachlicher Geschäftslogik bzw. MVC-Modell (vgl. Kapitel 2.4.4) gekennzeichnet ist. Struts-Actions übernehmen zentrale fachliche Funktionen und zugleich UI-nahe Aufgaben wie Request-Verarbeitung, Formularvalidierung, Navigation und View-Auswahl, wodurch eine systematische Trennung zwischen fachlichen Use Cases und darstellungsspezifischen Aspekten fehlt.

Diese Grundfunktionalitäten von Legacy-Framework führt zu gravierenden strukturellen Einschränkungen: UI-Änderungen (neue Interaktionsmuster, View-Anpassungen) erfordern tiefgreifende Modifikationen der Action-Klassen, technologische Updates sind aufgrund der Framework-Verkoppelung nur mit unverhältnismäßigem Refactoring-Aufwand möglich. Die resultierende hohe Änderungsanfälligkeit, erhöhte Fehlerwahrscheinlichkeit und Abhängigkeit von Legacy-Wissen manifestieren sich in steigenden Wartungskosten und eingeschränkter Innovationsfähigkeit. Diese Action-Interaktionen sind tief in die Backend-Logik eingebettet und mit serverseitiger Präsentationsverantwortung verknüpft, wodurch sie für moderne SPA-Architekturen unbrauchbar sind.

Dadurch fehlen standardisierte Schnittstellen, die Situation weiter verschärfend: Fachdaten sind ausschließlich über serverseitig gerenderte JSP-Seiten zugänglich, was Systemintegrationen (Fachanwendungen, externe Konsumenten) behindert und redundante Logik sowie inkonsistente Zugriffspfade erzeugt.

Nicht-funktionale Defizite verstärken das Problem:

- *Performance*: Synchrone Roundtrips bei Karteninteraktionen
- *Sicherheit*: Lückenhafte Absicherung auf UI-Ebene statt Service-Ebene

Diese akkumulierte technische Schuld gefährdet Wartbarkeit und Zukunftsfähigkeit. Eine grundlegende Modernisierung zu entkoppelter SPA-Architektur mit standardisierten REST-APIs ist zwingend erforderlich, um Interaktivität, Integration und Skalierbarkeit zu sichern.

4.2 Zielarchitektur: ISDB-Webanwendung

Vor dem Hintergrund der analysierten strukturellen Problemstellungen der Struts-basierten Anwendung wird als zentrale architektonische Leitentscheidung eine entkoppelte Zielarchitektur verfolgt. Diese realisiert eine systematische Trennung zwischen einer serviceorientierten Backend-Schicht und einem eigenständigen SPA-Frontend, wodurch Präsentationslogik, Interaktionssteuerung und fachliche Geschäftslogik unabhängig weiterentwickelbar werden.

4.2.1 Kernentscheidung: Entkoppelte Architektur mit SPA und API

Das Backend übernimmt ausschließlich fachliche Verantwortlichkeiten und Geschäftslogik, somit strukturiert sich nach einem etablierten Schichtenmodell: REST-Controller (Transportschicht), Service-Schicht (Use Cases) und Datenzugriffsschicht (Hibernate/PostGIS). Diese technologieunabhängige Struktur entkoppelt fachliche Logik von UI- und Framework-Abhängigkeiten.

Frontend-Backend-Kommunikation erfolgt über standardisierte REST-APIs nach **Contract-first-Prinzip** mit OpenAPI Specification (OAS 3.x) als verbindlichem Vertrag (vgl. Kapitel 2.4.7.2). Separate APIs für unterschiedliche Konsumenten (SPA-Frontend, Fachanwendungen, externe Systeme über eigene definierten Schnittstellen) gewährleisten Konsumentenunabhängigkeit und minimieren Seiteneffekte.

Die resultierende Zielarchitektur, dargestellt in Abbildung 8, bildet die strukturelle Grundlage für parallele Entwicklungsströme, schrittweise Migration und langfristige Skalierbarkeit. Sie adressiert die identifizierten Defizite der bestehenden Struts-Monolithen konsequent und schafft zugleich die Voraussetzung für zukünftige Erweiterungen, etwa mobile Clients oder spezialisierte GIS-Anwendungen, ohne erneute Architekturbrüche.

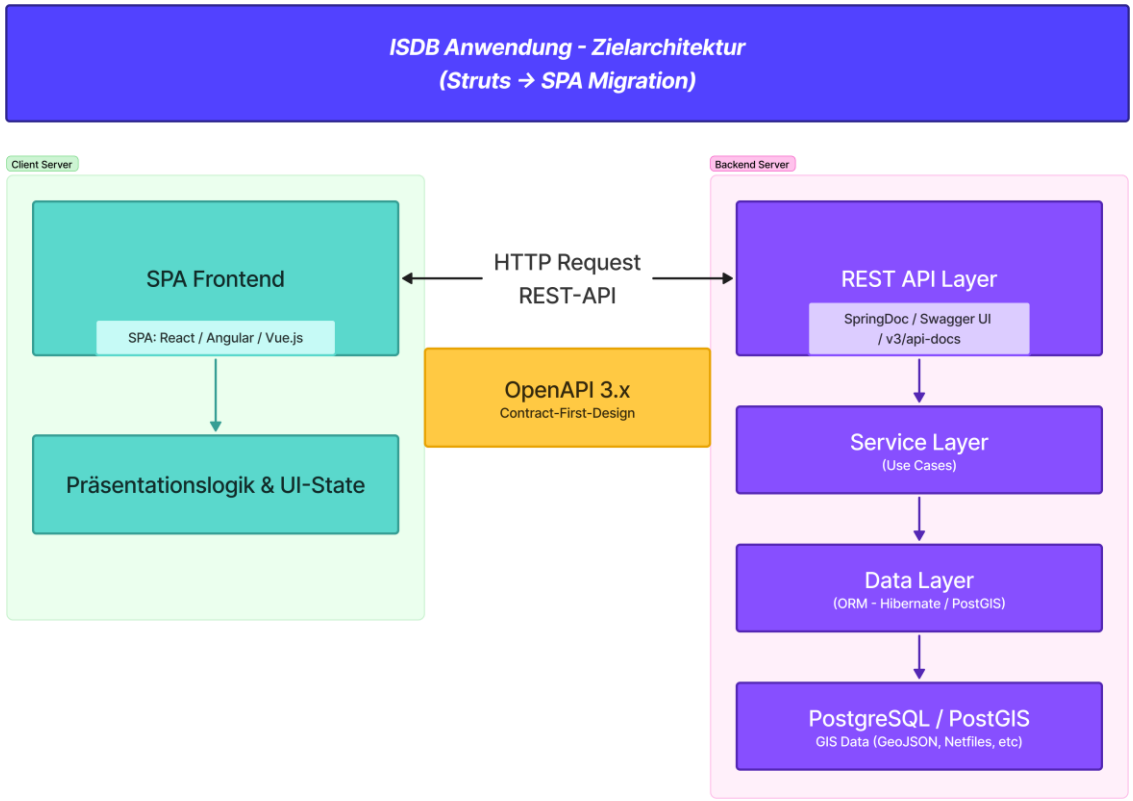


Abbildung 20: Zielarchitektur WebGIS ISDB – Entkoppelte SPA mit REST-API nach OpenAPI Standard (vgl. Kapitel 2.4.7.3)

4.2.2 Festlegung von Vue.js als SPA-Zieltechnologie

Für die Frontend-Modernisierung der ISDB-Anwendung wird **Vue.js** als clientseitiges SPA-Framework verbindlich festgelegt. Diese Entscheidung ergibt sich direkt aus den vergleichenden Analysen moderner Frontend-Frameworks (vgl. Kapitel 2.4.6) und stellt eine systematische Ableitung aus den dort etablierten Bewertungskriterien dar. Sie beruht nicht auf kurzfristigen Technologietrends, sondern resultiert aus einer strategischen Abwägung unter Berücksichtigung der Java-dominierten Enterprise-Umgebung sowie der spezifischen Anforderungen datenintensiver komplexer WebGIS-Anwendungen.

Die Studie von Kaluža et al. (2018) zeigt, dass Vue.js gegenüber Angular und React sowohl bei Single- als auch Multi-Page-Anwendungen die besten Gesamtbewertungen erzielt. Maßgeblich sind hierbei die geringe Einstiegshürde, die ausgewogene Architektur sowie die effiziente Unterstützung von State-Management-Lösungen und Performance-Optimierungsmechanismen. Die Vergleichsergebnisse in Tabelle 7 bestätigt, die insbesondere die Überlegenheit von Vue.js hinsichtlich Reaktivität, Verarbeitung komplexer Datenstrukturen und Eignung für Datenbank-nahe WebGIS-Anwendungen nachweist.

Zudem ermöglicht die komponentenbasierte Architektur von Vue.js eine modulare Umsetzung komplexer Karten- und Layer-Strukturen bei gleichzeitiger sauberer Trennung von UI-Komponenten, Zustandsverwaltung und fachlicher Interaktionslogik. Dies ist für WebGIS-Oberflächen von zentraler Bedeutung, die typischerweise aus wiederverwendbaren Bausteinen wie Layer-Verwaltungen, Filterdialogen, Feature-Detailansichten und Karteninteraktionskomponenten bestehen. Die Integration etablierter GIS-Bibliotheken wie OpenLayers erfolgt dabei strukturiert und ohne enge technologische Kopplung.

Spezifische Anforderungen für WebGIS-Komponentenstruktur:

- Layer – Listen (v-for + reactive data)
- Filter – Dialoge (Composition API)
- Feature – Detailansichten (v-slot + Teleport)
- Karteninteraktionen (Openlayers + Vue3)

Im Vergleich zu Angular reduziert Vue.js den erforderlichen Abstraktions- und Konfigurationsaufwand erheblich. Die in Kapitel 2.4.6 identifizierten Schwächen Angulars – insbesondere die hohe Framework-Komplexität und der resultierende technische Overhead – erscheinen für mittelgroße Anwendungen mit begrenzten Frontend-Ressourcen als nachteilig. Gegenüber React minimiert Vue.js den Boilerplate-Code-Bedarf und bietet ein kohärenteres Framework-Modell, was die Codebasis-Konsistenz steigert und den Technologieübergang für serverseitig geprägte Entwicklungsteams erleichtert.

Ein kritischer Aspekt stellt der Umgang mit komplexen GeoJSON-Strukturen dar. Wie in Kapitel 2.4.6 dargelegt, eignet sich Vue.js besonders für datenintensive UI-Komponenten-Layer. Die native TypeScript-Unterstützung gewährleistet typsichere Verarbeitung von FeatureCollections, Layer-Konfigurationen und Filterparametern und minimiert Integrationsfehler zur REST-API. Ergänzend verbessern Lazy-Loading-Mechanismen sowie das reaktive Datenmodell die Performance interaktiver Kartenanwendungen.

```
// Native TypeScript Unterstützung
interface FeatureCollection {
  type: "FeatureCollection";
  features: GeoJSON.Feature[];
}
```

Performance – Mechanismen:

- Lazy-Loading für Karten-Layer
- Reaktivität für Echtzeit-Updates
- Teleport für modale Feature-Infos

Daher erfolgt die Technologieentscheidung für Vue.js nicht als starre Festlegung. Durch die konsequente Entkopplung von Frontend und Backend mittels OpenAPI-definierter REST-API-Schnittstelle (vgl. Kapitel 2.4.7.3) bleibt die Benutzeroberfläche als austauschbare System-Schicht konzipiert, was ein Wechsel zu alternativen Frameworks jederzeit ohne Backend-Anpassungen ermöglicht.

Die Festlegung von Vue.js konstituiert die logische Umsetzung der in Kapitel 2.4.6 gewonnenen empirischen Erkenntnisse. Vue.js vereint hohe Praxistauglichkeit, optimierte Performance-Eigenschaften und geringe Einstiegshürden mit einer Architektur, die den Anforderungen schrittweiser WebGIS-Modernisierung gerecht wird und langfristige Wartbarkeit sowie Erweiterbarkeit der ISDB-Anwendung gewährleistet.

4.3 Modernisierungsstrategie von Backend und Frontend

Eine erfolgreiche Einführung eines SPA-Frontends setzt eine schrittweise Modernisierung sowohl des Backends als auch des Frontends voraus, bei gleichzeitiger Wahrung der Produktionsstabilität der bestehenden ISDB Struts-Anwendung. Dieser Kapitel beschreibt die Übergangsarchitektur für den parallelen Betrieb von Legacy-Struts und neuen REST-APIs, die strategischen Maßnahmen zur schrittweisen SPA-Einführung sowie das inkrementelle Migrationskonzept mit integrierter Risikominimierung. Ziel ist die Etablierung einer hybriden Systemlandschaft, die kontinuierliche Weiterentwicklung bei gleichzeitiger Funktionsgarantie ermöglicht.

4.3.1 Backend-Übergangsarchitektur: Parallelbetrieb von Struts und REST-API

Voraussetzung für die Einführung eines clientseitigen SPA-Frontends ist die Bereitstellung einer stabilen, standardisierten REST-Schnittstelle im Backend. Da die bestehende ISDB-Anwendung vollständig auf serverseitigem Rendering mittels Apache Struts basiert, ist eine vollständige Ablösung der Struts-Architektur zu Beginn der Modernisierung weder technisch noch organisatorisch tragbar. Stattdessen wird eine hybride Übergangsarchitektur etabliert, in der bestehende Struts-Actions und neue REST-Endpunkte parallel betrieben werden.

Abbildung 21 zeigt die parallele Betriebsarchitektur während der schrittweisen Migration. In dieser Übergangsphase bleibt die produktive Struts-Anwendung unverändert und stellt weiterhin JSP-basierte Benutzeroberflächen bereit. Parallel dazu wird eine unabhängige REST-API (OpenAPI/OAS3) entwickelt, die ausschließlich dem neuen SPA-Frontend (Vue.js) als Schnittstelle dient.

Beide Systeme greifen nahtlos auf dieselbe fachliche Geschäftslogik und Datenbasis (Hibernate/PostGIS) zu, ohne sich gegenseitig funktional oder performativ zu beeinträchtigen.

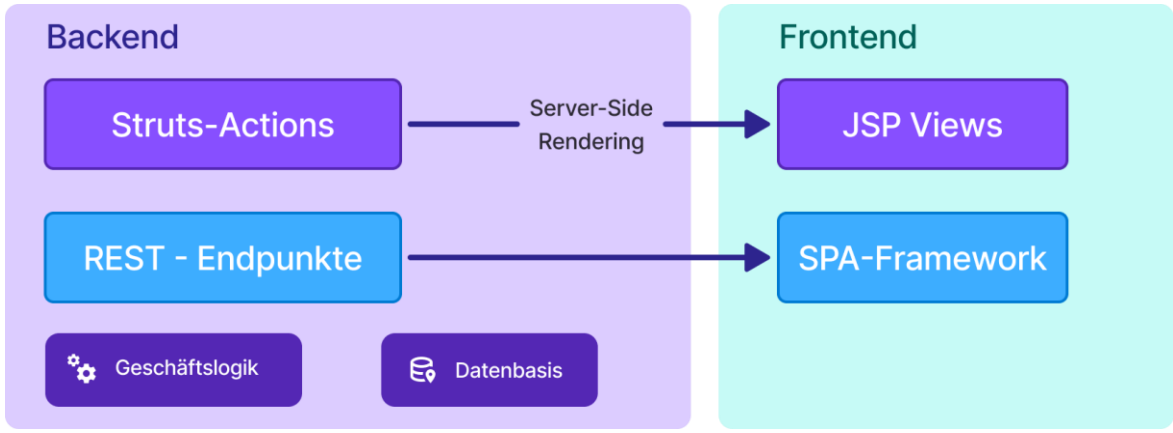


Abbildung 21: Übergangsarchitektur - Backend Parallelbetrieb

Kernvorteile ergeben sich daraus:

- Kontinuierlicher Betrieb ohne Ausfälle
- Gemeinsame Geschäftslogik ohne Duplikation
- Risikofreie Migration durch schrittweisen Funktionsumsetzung
- Parallele Entwicklung von Alt- und Neusystem sowie Systemupgrades

4.4 Strategie zur Frontend-Modernisierung

Dieses Kapitel beschreibt die strategische Vorgehensweise zur Modernisierung der Benutzeroberfläche der ISDB-WebGIS-Anwendung. Aufbauend auf der in Abschnitt 4.3 beschriebenen Backend-Übergangsarchitektur wird dargelegt, wie das neue SPA-Frontend schrittweise eingeführt wird und anhand welcher Kriterien die eingesetzte Technologie bewertet wurde. Im Mittelpunkt stehen Wartbarkeit, Integrationsfähigkeit und Migrationsfähigkeit als zentrale Qualitätsmerkmale einer inkrementellen Modernisierung.

4.4.1 Strategische Zielkriterien

Die Frontend-Modernisierung verfolgt nicht primär eine technologische Erneuerung, sondern die nachhaltige Verbesserung struktureller Qualitätsmerkmale. Zentrale Bewertungskriterien:

- **Wartbarkeit**: Klare Trennung von Zuständigkeiten, modulare Struktur, reduzierte Kopplung zwischen UI-Komponenten und fachlicher Logik. Für komplexe WebGIS-Anwendungen entscheidend, dass UI-Bausteine unabhängig angepasst werden können.
- **Integrationsfähigkeit**: Konsequente API-Zentrierung ohne implizite Backend-Abhängigkeiten. Notwendig für die Einbettung in größere Systemumfelder.

- **Migrationsfähigkeit**: Parallele Existenz zur Struts-UI mit sukzessiver Funktionsübernahme, ohne Betriebsunterbrechungen.

Da Vue.js als SPA-Framework gewählt wurde, ist eine präzise Anforderungen einer inkrementellen Migration erfüllt. Die komponentenorientierte Architektur ermöglicht die isolierte Entwicklung neuer UI-Funktionalitäten, die parallel zu bestehenden JSP-Seiten integriert werden können, ohne eine vollständige Umstellung zu erzwingen – dies entspricht dem Strangler-Pattern der Backend-Migration. Das Frontend bleibt strikt API-Konsument der REST-Architektur (vgl. Kapitel 4.3), wobei Präsentationslogik, UI-Zustand (Pinia) und Backend-Kommunikation klar getrennt sind. Die niedrige Einstiegshürde reduziert Boilerplate-Code (vs. React) und Framework-Overhead (vs. Angular), während TypeScript-Unterstützung typsichere GIS-Datenverarbeitung (GeoJSON) sichert.

Tabelle 12: Strategische Vorteile im Vergleich

Kriterium	Struts / JSP (Legacy)	Vue.js SPA (Ziel)
Wartbarkeit	Monolithisch	Komponentenbasiert
Integration	Framework-gebunden	API-zentriert
Migration	Komplexe Umsetzung notwendig	Inkrementell möglich
Teamfähigkeit	Backend Only	Java + JS parallel

Die Vue.js-Entscheidung schafft eine modulare, austauschbare Benutzeroberfläche, die unabhängig vom Backend-Migrationsfortschritt entwickelt werden kann, Risiken minimiert, Akzeptanz maximiert und die Grundlage für weitere Erweiterungen schafft.

4.5 API-Strategie und Designprinzipien

Dieses Kapitel beschreibt die zentrale Rolle der API als strategisches Fundament der modernisierten ISDB-WebGIS-Architektur. Die API konstituiert das maßgebliche verbindende Element zwischen Backend, SPA-Frontend (Vue.js) und weiteren Konsumenten. Ihre strategische Bedeutung kann nicht hoch genug eingeschätzt werden: Sie bildet die langlebige, stabile Schnittstelle, die Wartbarkeit, Interoperabilität und Evolvierbarkeit der gesamten Plattform gewährleistet und die Grundlage für ein API-zentriertes Unternehmensökosystem schafft.

Die REST-Schnittstellen werden gemäß einem API-First-/Design-First-Paradigma spezifiziert: Die Kontraktdefinition erfolgt vorab und implementierungsunabhängig als verbindliche Spezifikation. Dies realisiert funktionale Entkopplung von Frontend- und Backend-Entwicklung, eliminiert implizite Annahmen zu Datenstrukturen und minimiert Abstimmungsaufwand zwischen Teams.

Ein zentrales Gestaltungsprinzip ist die Segmentierung nach Konsumentengruppen mit konsumentenspezifischen APIs statt monolithischer Schnittstellen: Die SPA-Frontend-API ist nach CRUD (Create, Read, Update, Delete) Operationen orientiert und GeoJSON-optimiert für Vue.js, während Fachabteilungs-APIs use-case-spezifisch und stabil versioniert sind. Diese Granularität verhindert Seiteneffekte, optimiert die Performance und sichert domänenspezifische Stabilität.

Die OpenAPI Specification 3.x (OAS 3.x) fungiert als Single Source of Truth mit präziser Definition von Endpunkten, Schemata und HTTP-Semantik. Sie ermöglicht automatisierte Artefakt Generierung (Swagger UI, TypeScript-Clients), Validierung und CI/CD-Integration.

Räumliche Fachdaten werden kanonisch als GeoJSON (RFC 7946) serialisiert, was standardisierte Geometrie-/Feature-Repräsentation, OpenLayers-Integration und frontend-agnostische Persistenz sicherstellt.

Tabelle 13: Vergleichende Analyse der API-Designprinzipien

Prinzip	Traditioneller Ansatz	API-First (OAS 3.x)
Spezifikation	Code-first, implizit	Contract-first, explizit
Entwicklung	Sequential	Parallelisiert
Wartbarkeit	Implementierungsabhängig	Vertragskonform
Interoperabilität	Proprietär	Standardisiert

Konklusion: Die API-Strategie instituiert eine vertragsbasierte Schnittstellensemantik, die systemische Entkopplung, inkrementelle Evolution und API-zentriertes Ökosystemdesign ermöglicht – fundamentale Voraussetzung für die softwaretechnische Nachhaltigkeit komplexer WebGIS-Plattformen.

4.6 Transformationsstrategie

Dieses Kapitel beschreibt die risikominimierende Transformationsstrategie für die schrittweise Modernisierung der ISDB-Anwendung.

Eine Big-Bang-Migration wird ausgeschlossen, da sie für eine produktionskritische WebGIS-Anwendung untragbar wäre. Stattdessen verfolgt die Strategie einen inkrementellen Ansatz nach dem Strangler-Pattern: Die produktive Struts-Anwendung bleibt vollständig verfügbar, während parallel das neue Vue.js SPA-Frontend mit REST-API entwickelt wird. Beide Systeme greifen auf dieselbe Geschäftslogik (Spring-Services, Hibernate/PostGIS) zu, wodurch Code-Duplikation vermieden und Konsistenz gewährleistet wird.

Im parallelen Betrieb übernehmen neue Anforderungen ausschließlich das SPA, während bestehende Funktionalität weiterhin über Struts/JSP bereitgestellt wird. Die schrittweise

Ablösung des Legacy-Frontends erfolgt priorisiert nach Nutzungsstatistiken: Zuerst werden hochfrequent genutzte Module migriert, validiert durch iteratives Fachbereichs-Feedback und schrittweise in Produktion überführt. Dies garantiert 100% Verfügbarkeit, ermöglicht parallele Entwicklung und erlaubt jederzeitigen Rückfall auf das Legacy-System.

Kapitel 5 "Umsetzung und Migration der ISDB" detailliert die operative Umsetzung dieser Strategie mit konkreter Migrationsroadmap, Testkonzepten, Rollout-Phasen und Qualitätssicherungsmaßnahmen.

4.6.1 Steuerung, Entscheidungsfindung und Rückfallmechanismen

Die schrittweise Transformation der ISDB-Anwendung erfordert neben der technischen Parallelität von Alt- und Neusystem eine klare Steuerung der Migrationsentscheidungen. Jede Phase der Funktionsübernahme wird daher anhand fachlicher, technischer und betrieblicher Kriterien bewertet, bevor eine dauerhafte Ablösung von Legacy-Komponenten erfolgt. Maßgeblich sind dabei Nutzungshäufigkeit, fachliche Kritikalität, Stabilität im Produktivbetrieb sowie die Rückmeldungen der Fachanwender.

Ein zentrales Element der Transformationsstrategie ist die jederzeitige Rückfallfähigkeit. Da Alt- und Neusystem parallel betrieben werden, können migrierte Module bei unerwarteten Problemen temporär wieder auf die bestehende Struts-basierte Benutzeroberfläche zurückgeführt werden. Diese Möglichkeit reduziert das Risiko von Betriebsstörungen erheblich und erlaubt es, neue Frontend-Komponenten zunächst unter realen Nutzungsbedingungen zu erproben, bevor sie endgültig übernommen werden.

Die enge Einbindung des Fachbereichs unterstützt die Entscheidungsfindung zusätzlich. Durch iteratives Feedback zu Benutzerführung, Performance und Funktionalität kann frühzeitig bewertet werden, ob eine migrierte Komponente den fachlichen Anforderungen genügt. Erst nach erfolgreicher fachlicher und technischer Validierung erfolgt die endgültige Ablösung der entsprechenden Legacy-Funktionalität.

Durch diese Kombination aus klarer Steuerung, iterativer Bewertung und jederzeitiger Rückfalloption wird sichergestellt, dass die Transformation nicht nur technisch kontrolliert, sondern auch fachlich abgesichert erfolgt. Die Modernisierung der ISDB-Anwendung wird damit als gesteuerter Evolutionsprozess verstanden und nicht als einmaliges Migrationsereignis.

5 Umsetzung und Modernisierung der ISDB

Dieses Kapitel beschreibt die konkrete Umsetzung der in Kapitel 4 definierten Zielarchitektur sowie die schrittweise Migration der ISDB-Anwendung. Während die vorhergehenden Kapitel die strategischen und architektonischen Entscheidungen begründet haben, liegt der Fokus nun auf der praktischen Realisierung, der Anwendung der gewählten Methoden und der Validierung der Modernisierung im laufenden Betrieb. Die Umsetzung erfolgt kontrolliert, inkrementell und in enger Abstimmung mit dem Fachbereich.

5.1 Methodischer Umsetzungsansatz

Die Umsetzung der Frontend- und Backend-Modernisierung folgt einem iterativen, agilen Vorgehensmodell, das sich an den im Methodenkapitel beschriebenen Prinzipien orientiert. Zentrale Elemente dieses Ansatzes sind iteratives Prototyping, die gezielte Entwicklung von „*Proofs of Concept*“ sowie die kontinuierliche Einbindung des Fachbereichs in den Entwicklungsprozess.

Die Entwicklung wurde in aufeinanderfolgenden Iterationen s.g. Sprints organisiert, die jeweils einen vollständigen Zyklus von Konzeption, Implementierung, Validierung und Integration umfassten. Innerhalb einer Iteration wurden neue Anwendungskomponenten – bestehend aus Frontend-Funktionalitäten von Vue.js Framework sowie den zugehörigen Backend-Schnittstellen – prototypisch umgesetzt.

Zur fachlichen Validierung dienten sowohl visuelle Mockups als auch lauffähige Versionen auf Testumgebungen, die dem Fachbereich im Rahmen von Reviews und Live-Demonstrationen präsentiert wurden. Die dabei gemeldeten Feedbacks flossen unmittelbar in die Weiterentwicklung der jeweiligen Funktionalitäten ein.

Technisch wurden die neu entwickelten Komponenten schrittweise in den bestehenden Systembetrieb integriert. Dabei kam ein Parallelbetrieb von Legacy- und Neusystem zum Einsatz, um Risiken für den laufenden Betrieb zu minimieren. Die Kernfunktionalitäten wurden zunächst im Rahmen eines Pilotbetriebs einer begrenzten Fachbereichen zur Verfügung gestellt, bevor eine produktive Version Freigabe erfolgte.

Der beschriebene Sprint-Umsetzungsansatz ermöglichte eine frühzeitige Identifikation technischer und fachlicher Risiken sowie eine kontinuierliche Anpassung der Architektur und Implementierung vor einer flächendeckenden Einführung.

5.2 Implementierung der Backend-Architektur

Die Modernisierung des Backends der ISDB-Anwendung verfolgte das Ziel, eine stabile und standardisierte REST-Schnittstelle zu etablieren, welche als technische Grundlage für die Integration eines modernen Single-Page-Application-(SPA)-Frontends dient. Ausgangspunkt bildete eine bestehende, auf dem Struts-Framework basierende Webanwendung, deren Architektur nicht auf den externen Zugriff über APIs ausgelegt war. Die durchgeführten Anpassungen zielten daher nicht auf eine vollständige Substitution des Legacy-Systems, sondern auf eine kontrollierte Erweiterung durch eine API-zentrierte Zugriffsschicht ab.

5.2.1 ISDB Backend Strukturumstellung

Im initialen Modernisierungsschritt erfolgte die strukturelle Neuausrichtung der bestehenden Anwendung sowie die Migration in ein Maven-basiertes Build-System. Dadurch konnten eine konsistente Abhängigkeitsverwaltung und die Integration aktueller, Spring-basierter Komponenten realisiert werden. Auf dieser technologischen Basis wurde eine eigenständige REST-API-Schicht implementiert, welche unabhängig von den Struts-Actions operiert und parallel zum bisherigen Web Frontend eingesetzt werden kann.

Dazu wurden die relevanten Bibliotheken von Springframework für REST-API zum ISDB-Systemstruktur hinzugefügt, wie unten in ***pom.xml*** beispielsweise dargestellt.

```
<project xmlns="https://maven.apache.org/xsd/maven-4.0.0"
    xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>at.wienerlinien</groupId>
    <artifactId>WL_FwdbGis</artifactId>
    <version>12.12.2025</version>
    <packaging>war</packaging>
    <name>isdb</name>
    <description>Infrastructure Database</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-dependencies</artifactId>
        <version>2.0.2.6.04</version>
    </parent>
    <properties>
        <java.version>17</java.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
```

```
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>2.3.1</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-ldap</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-taglibs</artifactId>
        </dependency>
...........
</project>
```

5.2.2 REST-API Implementierung

Die OpenAPI Generator-Integration wurde als Maven-Profil „generate-api-profile“ hinterlegt, um die automatisierte Erzeugung von Java-Interfaces und DTO-Modellen aus der Spezifikation *isdb.v1.yaml* zu realisieren. Durch Ausführung des Maven-Profils werden folgende Artefakte kontinuierlich generiert:

- REST-Controller-Interfaces Endpunkte (z.B. GisSegmentApi, GeneralApi)
- Typsichere “Data Transfer Object”-Modelle (z.B. FeatureDTO, GeoJsonDTO)
- API-Utility-Klassen (ApiUtil.java) für Serialisierung/Validierung

```
<profile>
      <id>generate-api-profile</id>
      <build>
        <plugins>
          <plugin>
            <groupId>org.openapitools</groupId>
              <artifactId>openapi-generator-maven-plugin</artifactId>
              <version>5.3.0</version>
              <executions>
              <execution>
                <goals>
                  <goal>generate</goal>
                </goals>
                <configuration>
                  <inputSpec>${project.basedir}/isdb.rest.yaml</inputSpec>
                  <generatorName>spring</generatorName>
                  <apiPackage>at.wl.isdb.api</apiPackage>
                  <modelPackage>at.wl.isdb.api.model</modelPackage>
                  <output>${project.basedir}/gen/src</output>
                  <supportingFilesToGenerate>
                    ApiUtil.java
                  </supportingFilesToGenerate>
                  <configOptions>
                    <sourceFolder>/</sourceFolder>
                    <interfaceOnly>true</interfaceOnly>
                    <dateLibrary>legacy</dateLibrary>
                  </configOptions>
                </configuration>
              </execution>
            </executions>
          </plugin>
        </plugins>
      </build>
    </profile>
```

Die WebSecurityConfig.java (siehe unten) konstituiert einen zentralen Baustein der Backend-Modernisierung und bildet die Voraussetzung für den stabilen Parallelbetrieb von Legacy-Frontend (Struts/JSP) und neuer REST-API. Die Einführung datenorientierter HTTP-Endpunkte erweitert die Angriffsfläche erheblich, da sensible Fach- und Geodaten nun neben serverseitiger Rendering auch direkt über API-Zugriffe exponiert werden.

Konsistente Sicherheitsintegration von:

- **Legacy-Pfade** (Struts/JSP-Views)
- **Neue REST-Endpunkte** (OpenAPI-generiert)

```
// WebSecurityConfig.java Configuration

package at.wienerlinien.xy.security;

import org.springframework.web.servlet.handler.HandlerMappingIntrospector;

@Configuration
@EnableWebSecurity
@ComponentScan(basePackages =
{"at.wienerlinien.xy.security","at.wienerlinien.xy.api"})
@EnableAutoConfiguration(exclude = { DataSourceAutoConfiguration.class,
FreeMarkerAutoConfiguration.class })
public class WebSecurityConfig {

  private static final String LOGIN_JSP = "/login.jsp";

  private static final String ROLE_SERVICES = "SERVICES";
  private static final String ROLE_GRWLFWDB = "FWDBXYXY";

  @Configuration
  @EnableAutoConfiguration(exclude = { OAuth2ClientAutoConfiguration.class
})
  @ConditionalOnProperty(prefix = "xy", name = "sso.enabled", havingValue =
"false")

  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
    configureAuthenticationManager(http.getSharedObject(
AuthenticationManagerBuilder.class));
```

Die Konfiguration realisiert eine saubere Trennung zwischen UI-orientierten und serviceorientierten Zugriffen ohne gegenseitige Beeinträchtigung. Strategisch bedeutet, dass diese Konfiguration Sicherheitsfragmentierung während der Transformation verhindert und ermöglicht sicheren Parallelbetrieb. Sie schafft die technische Grundlage für inkrementelle Migration ohne Kompromisse bei der Sicherheitsarchitektur – essenziell für produktionskritische WebGIS-Systeme.

5.2.2.1 REST-API für performante GIS-Datenstruktur

Für GIS-Daten ist die polymorphe Modellierung unterschiedlicher Geometrietypen (Punkt, Linie, Polygon) zentral. Ein Contract-first-Design mit OpenAPI 3.x löst dies durch ein GeometryDTO mit folgenden Discriminator:

```yaml
# isdb.api.v1.yaml – Geometry Grundlage
  GeometryDTO:
    title: Geometry
    type: object
    properties:
      type:
        type: string
        enum:
          - Point
          - LineString
          - Polygon
    discriminator:
      propertyName: type
      mapping:
        Point: '#/components/schemas/PointDTO'
        LineString: '#/components/schemas/LineStringDTO'
        Polygon: '#/components/schemas/PolygonDTO'
```

Die eigentlichen Koordinaten werden anschließend je nach Spezialisierung strukturiert abgebildet (einfaches Array bei Punkten, verschachtelte Arrays bei Linien und Polygonen). Das erlaubt eine präzise Abbildung des GeoJSON-Konzepts, ohne die API auf einen einzigen Geometriedatentyp zu verengen:

```yaml
# isdb.api.v1.yaml - FeatureDTO Bausteine
  PointDTO:
    title: Point
    allOf:
      - $ref: '#/components/schemas/GeometryDTO'
      - type: object
        properties:
          coordinates:
            type: array
            items:
              type: number
              format: double
  LineStringDTO:
    title: LineString
    allOf:
      - $ref: '#/components/schemas/GeometryDTO'
      - type: object
        properties:
          coordinates:
```

```
            type: array
            items:
              type: array
              items:
                type: number
                format: double
  PolygonDTO:
    title: Polygon
    allOf:
      - $ref: '#/components/schemas/GeometryDTO'
      - type: object
        properties:
          coordinates:
            type: array
            items:
              type: array
              items:
                type: array
                items:
                  type: number
                  format: double
```

Auf Ebene einzelner Features wird die Geometrie dann in ein FeatureDTO eingebettet, wodurch sich GIS-Objekte konsistent transportieren lassen:

```
# isdb.api.v1.yaml – Feature Object
  FeatureDTO:
    title: Feature
    type: object
    properties:
      type:
        type: string
        enum:
          - Feature
      id:
        type: string
      properties:
        type: object
      geometry:
        $ref: '#/components/schemas/GeometryDTO'
```

Damit die REST-API nicht nur syntaktisch korrekt, sondern auch performant nutzbar ist, wird die Bereitstellung von GIS-Daten so gestaltet, dass datenintensive Geometrien bedarfsorientiert ausgeliefert werden. Für typische WebGIS-Anwendungsfälle bedeutet dies insbesondere die Unterstützung selektiver Abfragen, beispielsweise anhand räumlicher Parameter (z. B. Bounding-Box), klar abgegrenzter Ressourcenstrukturen sowie stabiler, konsumentenorientierter API-Verträge.Zum Schluss liefert der vordefinierten Endpunkt mit bestimmten Parametern FeatureCollection als Array von ISDB-Segmente inkl. Feature Objekten (siehe unten).

Ein vordefinierter API-Endpunkt liefert in diesem Zusammenhang eine FeatureCollection in Form einer strukturierten Liste von ISDB-Segmenten, wobei jedes Segment ein vollständiges GeoJSON-konformes Feature-Objekt enthält. Die Rückgabe umfasst sowohl Metadaten (z. B. Segment-ID, Modus, Index) als auch die zugehörige Geometrie, deren Typ (Point, LineString oder Polygon) gemäß dem im API-Vertrag definierten Geometriemodell bestimmt wird. Ein exemplarischer Ausschnitt einer solchen API-Antwort ist nachfolgend dargestellt.


```
[
  {
    "id": 2173070,
    "indexNr": 1,
    "gsbId": 299320,
    "feature": {
      "type": "Feature",
      "id": "2173070",
      "properties": null,
      "geometry": {
        "type": "Point",
        "coordinates": [1687.6801449777813, 342969.9305353842]
      }
    }
  },
  {
    "id": 2173074,
    "indexNr": 2,
    "modus": "GIS_POLY",
    "gsbId": 299320,
    "feature": {
      "type": "Feature",
      "id": "2173074",
      "properties": null,
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [1693.9282207303459, 342980.58353489323],
            [1691.3798469592052, 342981.78225412784],
            [1671.7423784874757, 342948.66763527214],
            [1644.6096930418032, 342960.2802278573],
            [1643.1106496470147, 342957.58310957946]
          ]
        ]
      }
    }
  }
]
```

5.2.3 Swagger Dokumentation

Die Integration einer interaktiven API-Dokumentation auf Basis von Swagger UI stellt einen essenziellen Bestandteil der Backend-Modernisierung des ISDB-Systems dar. Sie wird automatisiert aus der OpenAPI-Spezifikation generiert und gewährleistet dadurch eine konsistente, stets aktuelle Darstellung der implementierten REST-Schnittstellen. Eingebettet in die bestehende Sicherheitsarchitektur unterliegt sie denselben Authentifizierungs- und Autorisierungsmechanismen wie die produktiven Endpunkte.

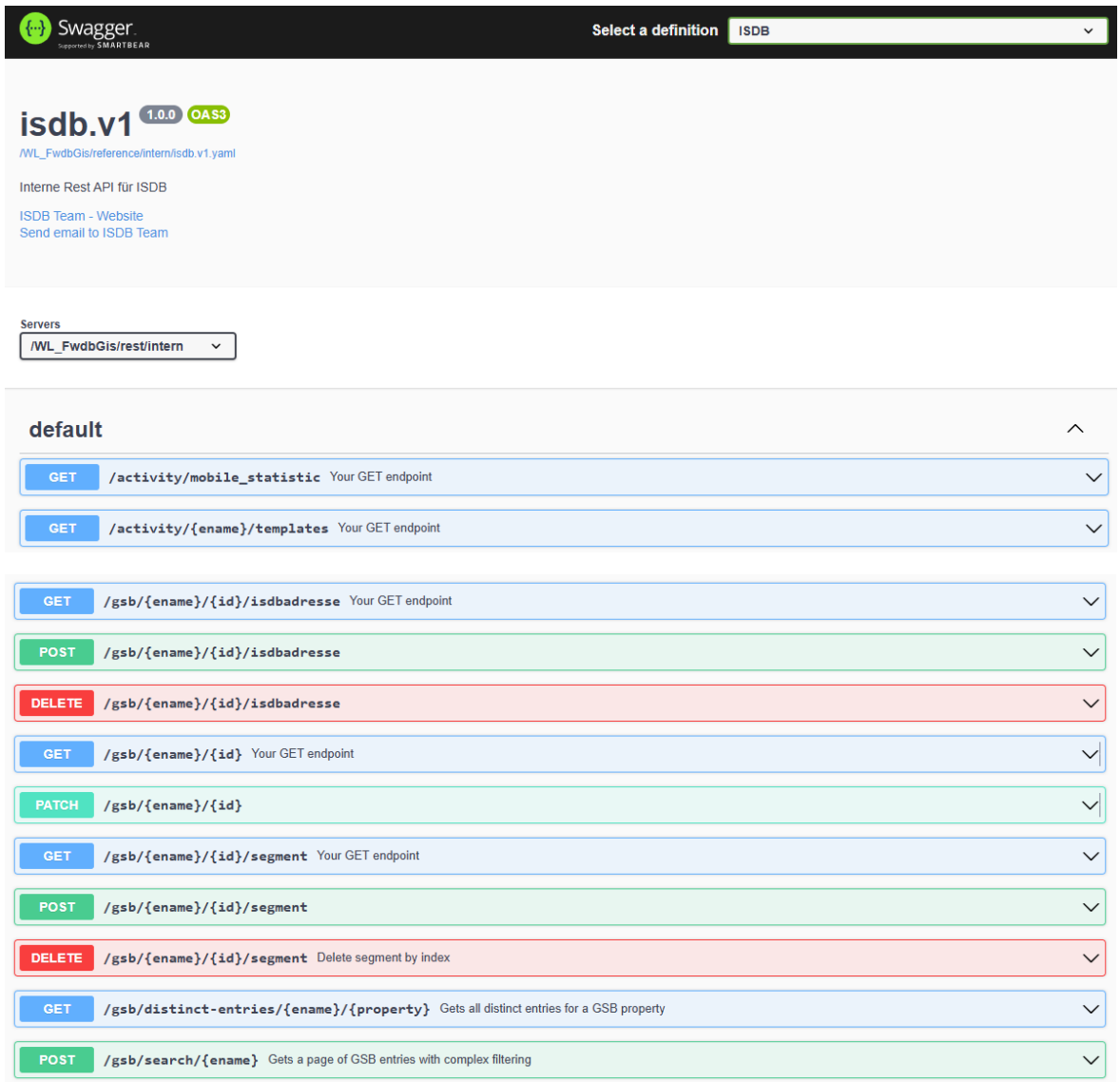


Abbildung 22: Swagger Dokumentation der ISDB

Die Swagger-Dokumentation (siehe Abbildung 22) fungiert sowohl als transparente Referenz für Entwickler und Fachanwender als auch als aktives Validierungswerkzeug. Über die integrierte Testfunktionalität können Endpunkte unmittelbar geprüft und fachliche Anforderungen frühzeitig verifiziert werden, was die Entwicklungs- und Abstimmungsprozesse deutlich vereinfacht.

Im Kontext der schrittweisen Frontend-Migration besitzt sie zudem strategische Relevanz wodurch die klare Dokumentation neu entwickelter REST-Services entsteht eine eindeutige Abgrenzung gegenüber Legacy-Funktionalität. Insgesamt unterstützt die Swagger-Dokumentation die API-First-Strategie, erhöht Transparenz und Wartbarkeit und leistet einen wesentlichen Beitrag zur Qualitätssicherung der modernisierten Systemarchitektur.

5.3 Frontend Implementierung mit Vue.js

Im folgenden Kapitel wird die konkrete Umsetzung des SPA-Frontends auf Basis von Vue.js im Rahmen der definierten Zielarchitektur beschrieben. Aufbauend auf den in Kapitel 4 hergeleiteten Architekturprinzipien wird dargestellt, wie die komponentenbasierte Struktur, die Anbindung an die REST-API sowie das clientseitige Zustandsmanagement realisiert wurden. Der Fokus liegt dabei auf der praktischen Umsetzung architektonischer Konzepte und nicht auf UI- oder Designaspekten.

Die Frontend-Implementierung folgt konsequent dem Prinzip der Entkopplung: Sämtliche fachlichen Daten werden ausschließlich über die definierte REST-API bezogen, während Präsentationslogik, UI-Komponenten-Zustand und Karteninteraktion vollständig im SPA-Frontend verbleiben, die unabhängig vom Backend weiterentwickelt werden kann.

5.4 Architekturstruktur des Vue.js – Frontends

Die Frontend-Architektur des ISDB-Systems basiert auf einem strikt komponentenorientierten Strukturprinzip, das auf dem Framework Vue.js in der Version 3 aufbaut. Ergänzend kommen zentrale Bibliotheken aus dem Node.js-Ökosystem (vgl. Kapitel 2.4.5.2) zum Einsatz in *package.json* File mit folgenden Einsatzbereichen:

- **Vue Router**: Realisierung und Steuerung der Anwendungsnavigation
- **Pinia**: Offizieller Vue 3 State Manager
- **Vuetify**: Material Design UI-Komponentenbibliothek
- **TanStack Vue Query**: Moderne Datenabfrage/Caching--Mechanismen
- **Axios**: De-Facto-Standard für HTTP-Requests
- **OpenLayers**: Branchenstandard für GIS-Kartenanwendungen

Die Abhängigkeitsverwaltung erfolgt Mithilfe von „Node Package Manager“, wodurch eine konsistente, reproduzierbare Build- und Laufzeitumgebung sichergestellt wird (siehe Abbildung 23). Die Auswahl dieser Bibliotheken erfolgte primär entlang klar abgegrenzter Verantwortlichkeiten innerhalb der Frontend-Architektur und nicht aus Gründen technologischer Vollständigkeit.

Das Architekturkonzept folgt dem Prinzip der klar abgegrenzten Komponentenverantwortlichkeit: Jede Vue-Komponente kapselt eine wohldefinierte Funktionalität und interagiert ausschließlich über explizite Schnittstellen. Dieses Vorgehen unterstützt Wartbarkeit, Testbarkeit und schrittweise Erweiterbarkeit der Anwendung. Durch die lose Kopplung einzelner UI-Bausteine kann die Benutzeroberfläche sukzessive modernisiert werden, ohne den Bestandscode vollständig ersetzen zu müssen.

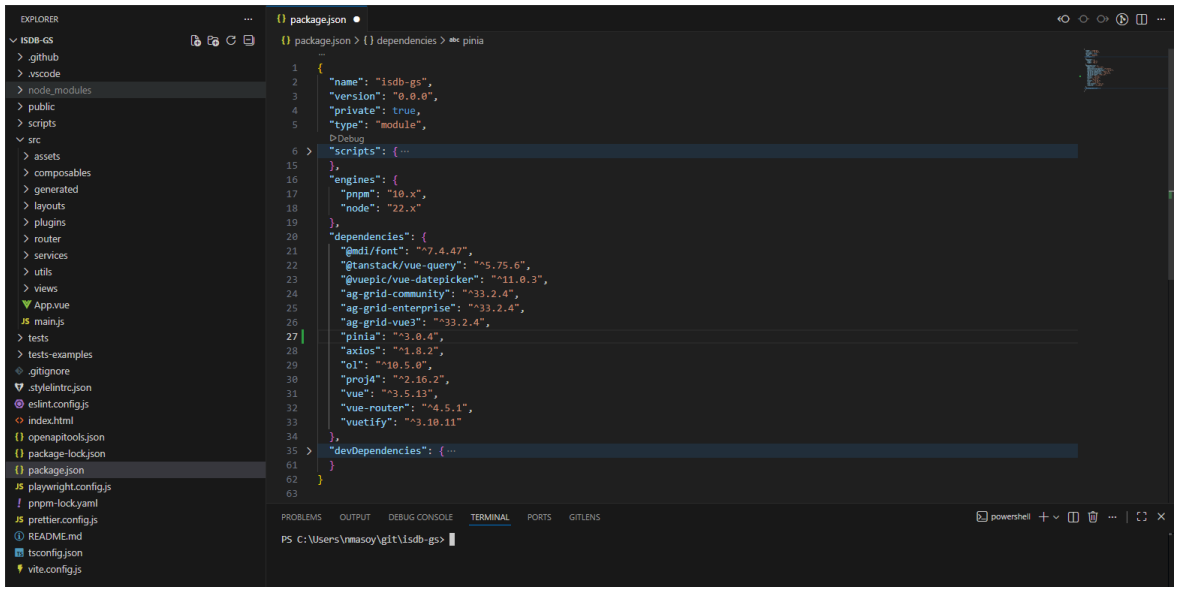


Abbildung 23: Projektstruktur und Abhängigkeitsdefinition des Vue.js Frontends

Die Strukturierung der Anwendung erfolgt primär innerhalb des ***src***-Verzeichnisses, das die fachliche und technische Modularisierung des Frontends abbildet. Tabelle 13 zeigt die zentralen Verzeichnisse und deren jeweilige Funktion innerhalb der Architektur.

Tabelle 14: Struktur des src-Verzeichnisses der Vue-Frontend-Anwendung

Verzeichnis / Datei	**Funktion und Verantwortlichkeit**
main.js	Einstiegspunkt der Anwendung; Initialisierung der Vue-App sowie Registrierung globaler Plugins (z. B. Router, Pinia, Vuetify, Vue Query).
App.vue	Einstiegskomponente der Anwendung; Kapselung globaler Layouts und Navigationslogik, Container für geroutete Views.
assets/	Statische Ressourcen wie Icons, Stylesheets oder Bilder.
composables/	Wiederverwendbare Hooks auf Basis der Composition API (z. B. useUserdata, useFetchModels), kapseln logische Funktionalität ohne UI-Abhängigkeiten.

generated/	Generierter Code, insbesondere aus OpenAPI-Generator/Swagger; dient der sauberen Trennung von Anwendung Codes und generierten Klassen Artefakten.
layouts/	Wiederverwendbare Layout-Templates (z. B. Standard- oder Overview/CreateOrUpdate Pages), fördern Konsistenz und reduzieren Code-Duplizierung.
plugins/	Konfiguration und Registrierung externer Frameworks und Bibliotheken (z. B. Vuetify, Vue Query).
router/	Definition der Routen und optionaler Guards; Trennung von Navigations- und UI-Logik.
services/	Abstraktionsschicht für externe Schnittstellen (API-Clients, Mapper, Adapter), entkoppelt UI von Transport- und Integrationslogik.
utils/	Allgemeine Hilfsfunktionen ohne Vue-spezifische Abhängigkeiten (Integrationsfunktionalitäten, Funktionslogiken Konstanten).
views/	*Domain-Driven-Design* Komponenten der Datensätze, die über Routen angesprochen werden und kohärente Unterkomponenten zusammenfassen.

Diese strukturierte Organisation unterstützt eine skalierbare und modulare Weiterentwicklung der Benutzeroberfläche. Jede Schicht der Anwendung übernimmt eine klar definierte Rolle innerhalb des Frontend-Ökosystems. Die konsequente Trennung zwischen Präsentationslogik, Datenzugriff und Konfigurationsartefakten trägt wesentlich zur Nachhaltigkeit, Wartbarkeit und Konsistenz der Modernisierung bei.

5.5 GIS-Komponenten und OpenLayers-Integration

Die GIS-Funktionalität der ISDB-Anwendung wird im Frontend durch eine eigenständige, wiederverwendbare Kartenkomponente realisiert, welche die Integration der OpenLayers-Bibliothek in sich kapselt. Diese GIS-Komponente übernehmen ausschließlich Aufgaben der geographischen Darstellung und der clientseitigen Interaktionen – etwa Zoom, Pan, Overlays oder Feature-Selektion – und enthält explizit keine fachliche Geschäftslogik. Fachliche Entscheidungen, Validierungen oder Geodaten-Übertragungen (Persistenz-Vorgänge) werden außerhalb der Karte in den verantwortlichen Backend-Logiken über REST-API Validierungen umgesetzt.

Diese klare Trennung zwischen Visualisierung (Frontend) und Fachlogik (Backend) bildet ein zentrales Architekturprinzip der Modernisierung, da sie die Wartbarkeit, Austauschbarkeit und Wiederverwendung der Kartenkomponente sicherstellt.

Die Kartenintegration ist in die sogenannten *Create-or-Update-Ansichten* (EditDataPage) eingebettet, wobei die Komponente als unabhängiger Slot in verschiedenen Templates (z. B. ReadOnlyMap.vue oder FeatureMap.vue) eingesetzt werden kann.

Dadurch bleibt die Kartenlogik vollständig unabhängig von der Datensatzstruktur und kann sowohl zur reinen Anzeige als auch für interaktive Editierfunktionen verwendet werden. Kommunikation zwischen je Datensatzformular und Karte erfolgt ausschließlich über wohldefinierte *Properties* und *Events*, womit die lose Kopplung zwischen UI-Komponente-Logik und Karten-Rendering gewahrt bleibt.

5.5.1 Einbindung der Karte als wiederverwendbarer UI-Baustein

Die Kapselung zeigt sich exemplarisch in der Einbindung der ReadOnlyMap-Komponente, die in verschiedenen Ansichten als eigenständiger visueller Block referenziert wird.

```
<!-- Einbindung der Kartenkomponente als austauschbarer UI-Baustein -->
  <ReadOnlyMap
    :id="Number(id)"
    :type="fwdbType"
    min-height="20rem"
    max-height="20rem"
    class="rounded border overflow-hidden"
  />
```

Die Komponenteninstanz erhält ausschließlich darstellungsrelevante Parameter wie ID oder Datensatz spezifischen Typen. Fachlogik, Validierungsregeln oder Geodatenzugriffe liegen außerhalb der Karte. Damit ist nachweisbar, dass die Kartenkomponente eine reine Präsentationsschicht darstellt (siehe Abbildung 24).

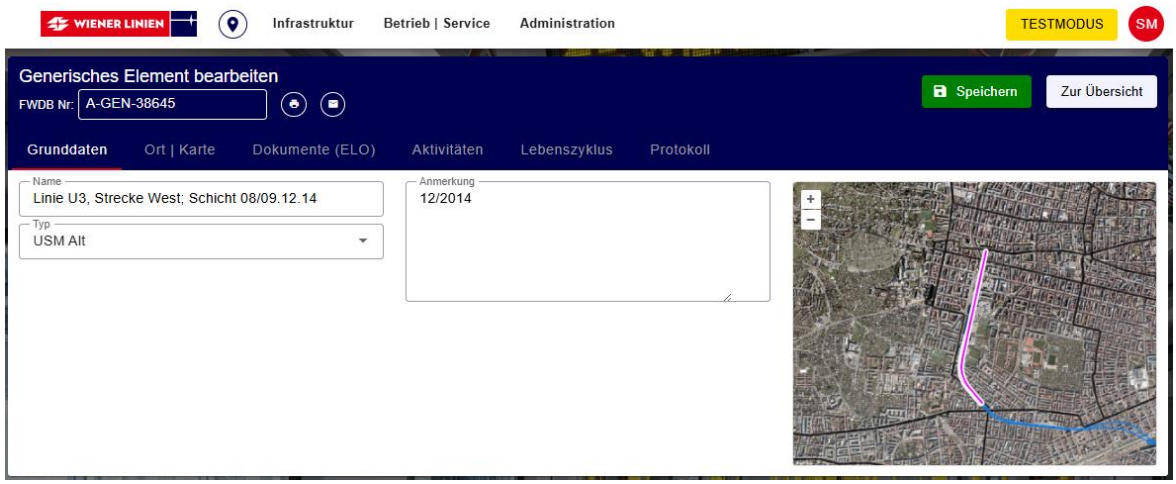


Abbildung 24: ReadOnlyMap-Komponente – Wiederverwendbarer UI-Baustein zur Visualisierung von Element-Features

5.5.2 Karteneditierung in Feature-Detailansichten

Für editierbare Szenarien der Features wird die Kartenansicht als separater Tab innerhalb des Datensatzes eingebunden. Dadurch bleibt die Karteninteraktionslogik von Datensatzvalidierung und Zustandsverwaltung entkoppelt.

Zusätzlich validiert das Backend über vordefinierten REST-API Endpunkte die GIS-Datentypen (via geom-types), sodass pro Segmenttyp ausschließlich konforme Geodaten-Geometrien editierbar und speicherbar sind: Baume nur als Point (Positionsgenauigkeit), Gleiselemente als LineString (Netztopologie) und Streckenabschnitt als Polygon (Arealbezug). Dieser Ansatz gewährleistet fachliche Konsistenz durch serverseitige Typenprüfung vor Persistierung.

```
<!-- Integration der editierbaren Kartenansicht in separatem Tab -->
  <v-tabs-window-item :value="tabNameEditTemplateO['Ort | Karte']">
    <EditFeatureMap
      v-if="id"
      :gsb-id="Number(id)"
      :geom-types="[]"
      :gsb-type="type as FwdbElementTypeDTO"
      :refetch-on-segment-change
    />
    <TabPlaceholder v-else />
  </v-tabs-window-item>
```

Die editierbare Kartenansicht in der Feature-Detailansicht ermöglicht die interaktive Bearbeitung räumlicher Feature-Geometrien. Abbildung 25 zeigt die resultierende Benutzeroberfläche mit aktivierter Karteninteraktion im Anwendungskontext.

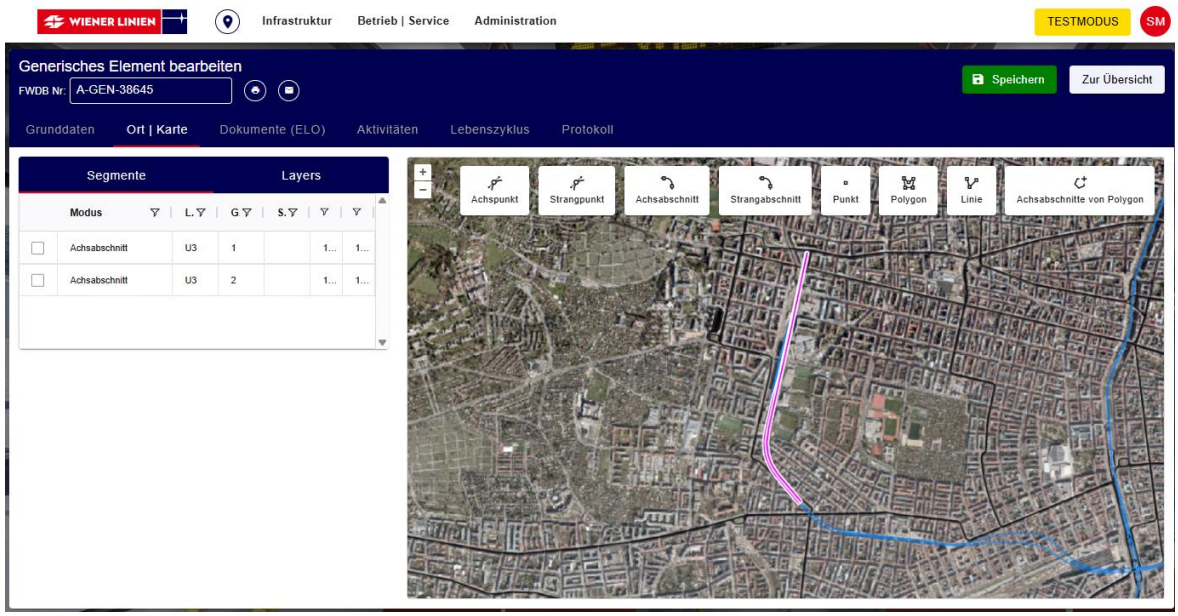


Abbildung 25: FeatureMap – Komponente zur interaktiven Bearbeitung von GIS-Feature-Geometrien

Die Karte agiert als austauschbares UI-Modul, dessen Aktivierung im Edit-Data-Template über Tabs erfolgt. Somit bleibt der Formularzustand unabhängig von der Karteninteraktion. Dieses Design belegt sowohl Modularität als auch die isolierte Erweiterbarkeit einzelner Funktionen.

5.5.3 Layer-Management und Bindung externer Kartendienste

Die Kartenkomponente des ISDB-Frontends integriert verschiedene Layer-Typen, die funktional klar voneinander abgegrenzt sind. Während Basiskarten (z. B. Orthofotos oder thematische Hintergrundkarten) als rasterbasierte *Tile-Layer* eingebunden werden, werden Fachdaten als *Vektor-Layer* über die REST-API im GeoJSON-Format bezogen.

Diese Trennung zwischen Raster- und Vektorebene ermöglicht eine unabhängige Weiterentwicklung der Darstellungslogik und der fachlichen Datenhaltung. Zudem erlaubt sie den parallelen Betrieb unterschiedlicher Datenquellen, ohne dass deren technische Eigenschaften in der UI-Schicht berücksichtigt werden müssen.

Das Layer-Management folgt einem konfigurationsgetriebenen Ansatz, bei dem sämtliche Layer-Eigenschaften – wie Name, Opazität, Zoomgrenzen oder räumliche Ausdehnungen (Extents) – in einer zentralen Metadatenstruktur beschrieben sind. Neue Layer können dadurch hinzugefügt oder bestehende Parameter angepasst werden, ohne dass Quellcodeänderungen in der Kartenkomponente erforderlich sind. Dieses Prinzip unterstützt sowohl Wiederverwendbarkeit als auch Wartbarkeit und stellt sicher, dass UI-spezifische Logik unabhängig von den technischen Details der Datenquelle bleibt.

```
// Konfiguration Layer-Metadaten
export interface MapLayer {
  propertyName: string;
  minZoom: number;
  maxZoom: number;
  opacity: number;
  layerName: string;
  extent: [number, number, number, number];
}

export const mapLayers: MapLayer[] = [
  { propertyName: 'Orthofoto Wien', layerName: 'Cache_Orthofoto', minZoom:
14, maxZoom: 19, opacity: 1, extent: [...] },
];
```

Ein geänderter Layer erfordert nur eine Anpassung in der Konfigurationsdatei, nicht in der Kartenlogik selbst. Dieses Vorgehen minimiert Code-Duplikationen und schafft eine einheitliche Schnittstelle zwischen fachlicher Definition und technischer Instanziierung.

Für die Einbindung externer Kartendienste werden standardisierte Schnittstellen genutzt, was die Interoperabilität mit etablierten GIS-Infrastrukturen sicherstellt. Rasterdaten werden primär über XYZ-Tile-Services eingebunden, deren Kachelkonfiguration

(TileGrid-Definition, Auflösung, Ursprung) an die projektinterne Kartenprojektion angepasst ist. Dadurch können Kacheln Performance optimiert gerendert und Ladevorgänge deterministisch gesteuert werden. Darüber hinaus werden ausgewählte Infrastrukturinformationen über OGC-konforme WMS-Dienste (Web Map Service) angebunden. Die Implementierung der zugehörigen Layer erfolgt generisch über eine Hilfsfunktion, die auf standardisierte Parameter setzt.

```
// Einbindung eines OGC-WMS-Layers
export function getRailLayer(geoserver_url: string) {
  return new TileLayer({
    properties: { name: 'Gleisachsen Wien' },
    source: new TileWMS({
      url: geoserver_url,
      params: { LAYERS: 'fwdb3sys:gleisachsen_neu', VERSION: '1.1.1' },
      serverType: 'geoserver',
    }),
  });
}
```

Durch diese Standardkonformität bleibt der fachliche Layer unabhängig von Implementierungsdetails des Kartendienstes. Änderungen an Server URLs oder Dienstparametern sind unmittelbar über die Konfiguration steuerbar und machen eine Neukompilierung des Frontends entbehrlich.

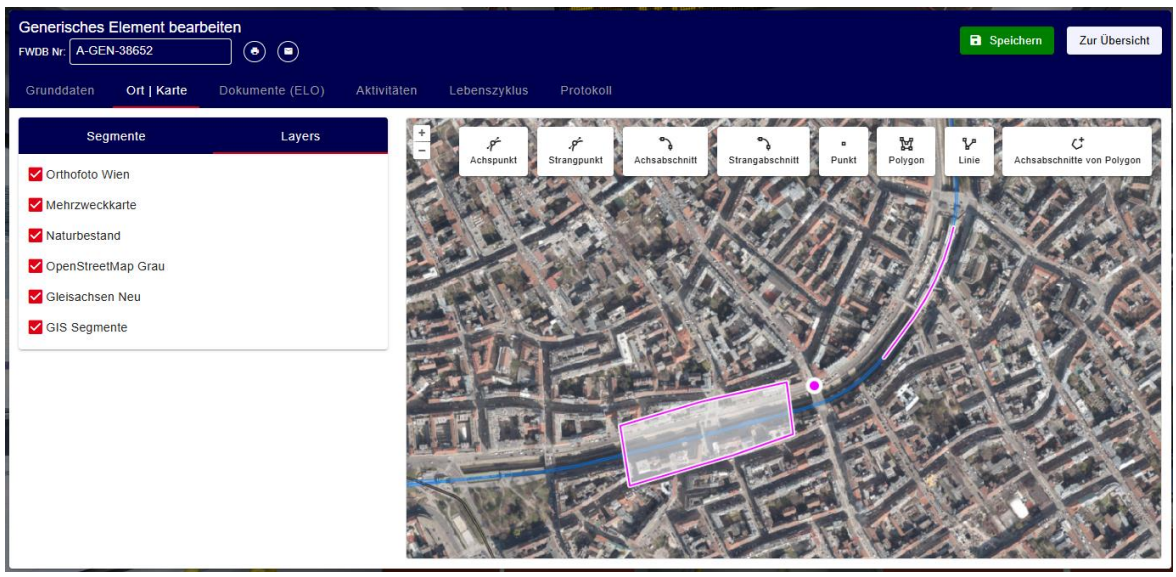


Abbildung 26: Frontend Layer-Konfiguration – Wiederverwendbare XYZ-Tile, WMS und REST-GeoJSON-Integration

Zur Sicherstellung einer stabilen Performance werden räumliche und semantische Filter angewandt. Zoombereiche und Extents begrenzen die Datenabfrage auf relevante Kartenausschnitte, wodurch sowohl Netzwerkbelastung als auch Rendering-Zeit reduziert werden. Insbesondere bei hochauflösenden Basiskarten trägt diese Einschränkung zur deterministischen Steuerung der Darstellungslogik bei.

Zusammenfassend ermöglicht das konfigurationsgetriebene Layer-Management (siehe Abbildung 26) der ISDB-Kartenkomponente eine skalierbare, serviceorientierte Integration mehrerer Kartenquellen. Die konsequente Nutzung offener Standards (XYZ, OGC-WMS, GeoJSON) gewährleistet Interoperabilität, während die Entkopplung von Fach- und Darstellungslogik eine nachhaltige Weiterentwicklung der Geovisualisierung unterstützt.

5.5.4 Trennung von Datenzugriff und GIS-Funktionslogik

Eine weitere strukturelle Entkopplung erfolgt zwischen Datenzugriffs- und GIS-Funktionslogik. Die Datenabfragen werden außerhalb der Kartenkomponente über dedizierte Query- und Service-Klassen umgesetzt. Für REST-API Kommunikation wird die Bibliothek TanStack Vue Query eingesetzt, sodass Caching, Statusverwaltung und Wiederverwendung zentral erfolgen.

```
// Datenzugriff mit erforderlichen Parametern für REST-API
const queryFn = useQuery({
  queryKey: [props.fwdbType, id],
  queryFn: async () => {
    if (id.value) {
      const res = await fetchGsbById(props.fwdbType, Number(id.value));
      return res.data;
    } else return null;
  },
});
```

Die OpenLayers Karte selbst verarbeitet ausschließlich Geometrien und Feature-Zustände, die über diese REST-API-Abfragen bereitgestellt werden. Persistenz, Berechtigungen und Datenmodell-Validierung liegen vollständig in der Service- bzw. Backend-Schicht. Damit wird sichergestellt, dass Änderungen an Persistenz Mechanismen oder Berechtigungslogik keine Anpassungen in der Kartenkomponente erfordern.

5.5.5 Architekturimplikation und Austauschbarkeit

Die ISDB-Kartenkomponente folgt damit einem strikt modularen Ansatz:

- **Austauschbarkeit** durch kapselnde Einbindung als Komponente oder Slot
- **Wiederverwendung** über kontextspezifische Templates (ReadOnly / EditFeature)
- **Separation of Concerns** durch Auslagerung der Daten- und Fachlogik in Service- und Query-Schichten
- **Technologische Offenheit** durch Nutzung standardisierter Technologie-Schnittstellen (OGC, REST / GeoJSON).

Durch diese Softwarearchitektur Prinzipien bleibt die Kartenkomponente unabhängig von Backend-Details und langfristig erweiterbar für neue Datenquellen oder zusätzliche Interaktionskonzepte. Sie bildet damit ein zentrales Beispiel für die erfolgreiche Umsetzung einer API-gestützten, komponentenorientierten GIS-Architektur innerhalb der ISDB-Modernisierung.

5.6 Herausforderungen und Lösungsansätze

Die schrittweise Modernisierung der produktiv eingesetzten WebGIS-Anwendung ISDB stellte ein komplexes Zusammenspiel technischer, organisatorischer und fachlicher Anforderungen dar. Zentrale Herausforderungen ergaben sich aus dem Parallelbetrieb von Legacy- und Neusystem, der Koordination zwischen Fachbereich und Softwareentwicklung, der Gestaltung stabiler REST-Schnittstellen sowie Performance-Aspekten bei großen komplexen Geometrien. Die erfolgreiche Bewältigung dieser Problembereiche stützte sich auf systemarchitektonische Prinzipien, iterative Prozesse und einen strukturierten Projektmanagement Ansatz.

5.6.1 Paralleler Betrieb von Legacy- und Neusystem

Eine grundlegende Herausforderung bestand in der Sicherstellung des kontinuierlichen Produktivbetriebs der ISDB-Anwendung während der parallelen Implementierung neuer Systemkomponenten über REST-API und Vue.js-Frontend. Funktionale Überschneidungen oder divergierende Geschäftslogik hätten die Systemkonsistenz gefährdet.

Zur Umsetzung der Modernisierung wurde die Geschäftslogik aus der monolithischen Struts-Architektur extrahiert und in eine eigenständige Service-Schicht abstrahiert, die über REST-Controller angesprochen wird. Während Struts-Actions und REST-Controller dieselbe Geschäftslogik und Datenbank persistieren, erfolgen die Systemoperationen strikt entkoppelt. Dies ermöglicht den parallelen Betrieb beider Architekturen ohne gegenseitige Interferenzen oder funktionale Seiteneffekte. Ergänzend wurden Feature-Enablers sowie ein modulares Routingkonzept implementiert, wodurch eine kontrollierte Freischaltung neuer Funktionalitäten sukzessiv ermöglicht wird. Der Ansatz gewährleistet rasche Rollback-Mechanismen bei Problemen sowie inkrementelle Modernisierung und Erweiterbarkeit des gesamten ISDB-Systems.

5.6.2 Abstimmung und Validierung im Fachbereich

Die ausgeprägte visuelle und interaktive Charakteristik von WebGIS-Systemen machte einen erhöhten Validierungsaufwand erforderlich, da klassische Unit- oder API-Tests auf Backend-Ebene die fachliche Nutzbarkeit – insbesondere hinsichtlich Layer-Darstellungen oder Feature-Selektionsverhalten – nur unzureichend abbilden konnten.

Zur effektiven Qualitätssicherung wurde daher ein kombiniertes Testkonzept etabliert: Teilautomatisierte End-to-End-Tests im Frontend dienten primär der Erkennung technischer Regressionen, während die fachliche Validierung iterativ durch regelmäßige Reviews mit dem zuständigen Fachbereich erfolgte. Diese wurden auf Basis interaktiver Prototypen und bereitgestellter Testversionen in dedizierten Entwicklungsumgebungen durchgeführt. Die frühzeitige und kontinuierliche Einbindung der Fachanwender ermöglichte eine zeitnahe Feedbacks zu fachlichen Anforderungen, minimierte das Risiko von Fehlentwicklungen und förderte die Akzeptanz der neu entwickelten Benutzeroberfläche nachhaltig.

5.6.3 API-Design und Schnittstellenqualität

Das API-Design erwies sich als zentraler Stabilitätsfaktor im Entwicklungsprozess. Zu generisch definierte Endpunkte führten frühzeitig zu Missverständnissen und inkonsistenten Datenmodellen zwischen Frontend- und Backend-Teams.

Um eine klare und konsistente Schnittstellenstruktur zu gewährleisten, wurde die Unterscheidung verschiedener Datenmodellarten – etwa Attributdatensätze, strukturierte Fachdaten sowie GIS-bezogene Datenmodelle – frühzeitig im Entwicklerteam festgelegt. Auf dieser Basis konnten Endpunkte thematisch gruppiert und die zugrunde liegende Geschäftslogik sauber zwischen Frontend und Backend abgegrenzt werden. Dadurch entstand ein vertraglich klar definiertes API-Design, das die Wartbarkeit erhöhte und Kommunikationsaufwände im weiteren Entwicklungsverlauf deutlich reduzierte.

Ein Design-First-Ansatz mit OpenAPI etablierte die Spezifikation zusätzlich als verbindlichen Vertrag und „*Single Source of Truth*“ zwischen den beteiligten Systemen. Contract-Tests stellten sicher, dass Implementierungen langfristig konform zur Spezifikation blieben. Trotz des initial höheren Abstimmungsaufwands führte dieser Ansatz zu robusteren und wartbareren Schnittstellen sowie zu einer nachhaltig stabileren Systemarchitektur.

5.6.4 ISDB-Projektmanagement als Erfolgsfaktor

Die Bewältigung der komplexen Herausforderungen der ISDB-Modernisierung wurde maßgeblich durch einen strukturierten Projektmanagement-Prozess unterstützt, der in vier übergeordnete Phasen gegliedert ist: *Vorbereitung*, *Definition* und *Planung*, *Implementierung* sowie Test- und Release-Management. Dieses Modell, dargestellt in Abbildung 27, adressiert organisatorische Risiken des Parallelbetriebs durch klare Verantwortlichkeiten, iterative Feedback-Schleifen und definierte Meilensteine. Die horizontale Struktur (grün hervorgehoben) symbolisiert den zeitlichen Ablauf, während vertikale Spalten farblich codierte fachliche Rollen abbilden.

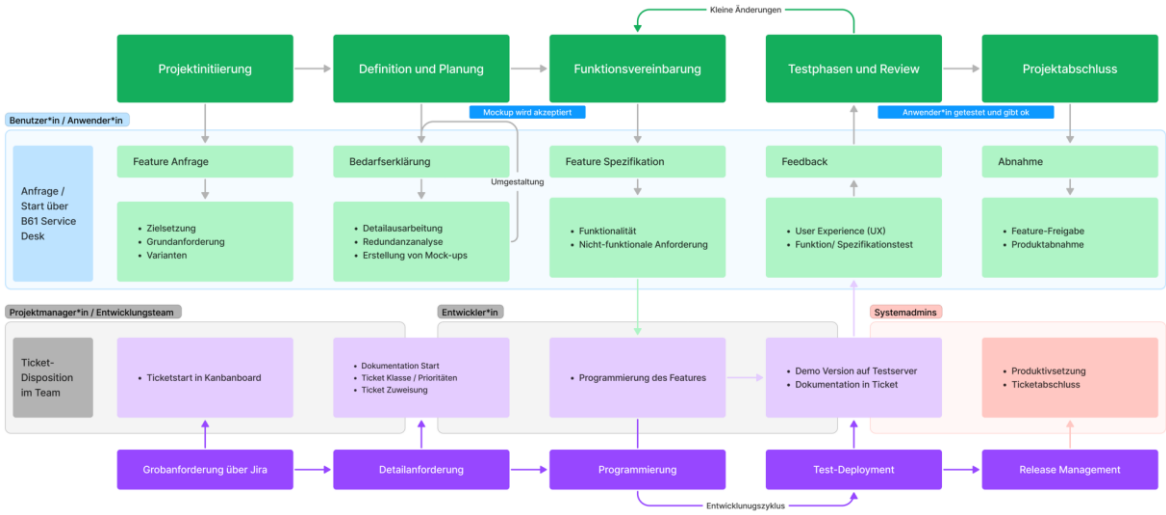


Abbildung 27: Projektmanagement-Prozess der ISDB-Modernisierung

Phasenübersicht und Verantwortlichkeiten:

1. **Vorbereitung**: Initiale Projektvorbereitung (Feature Anfrage) mit Fokus auf der fachlichen Anforderungsdefinition (oberer Bereich) sowie der Klärung organisatorischer und technischer Rahmenbedingungen (unterer Bereich). In dieser Phase werden Zielsetzungen, beteiligte Rollen, Abhängigkeiten und Risiken identifiziert, um eine konsistente Ausgangsbasis für alle weiteren Schritte zu schaffen.

2. **Definition und Planung**: Detaillierte Ausarbeitung der Feature-Spezifikation sowie der OpenAPI verbindliche REST-API. Ergänzend erfolgen grundlegende Entscheidung zur Umsetzung und die Erstellung erster Mock-ups. Diese Phase etabliert einen stabilen, abgestimmten Referenzrahmen für Backend- und Frontend-Entwicklung.

3. **Implementierung**: Parallele Umsetzung der spezifizierten Anforderungen in Backend- und Frontend-Komponenten. Der Einsatz von Feature-Enabler ermöglicht eine kontrollierte, schrittweise Aktivierung neuer Funktionen, während die parallele Entwicklung eine effiziente Nutzung der Entwicklungsressourcen unterstützt.

4. **Test- und Releasemanagement**: Bereitstellung fachlicher und technischer Tests auf dedizierten Testumgebungen sowie Vorbereitung des Deployments. Die finale Freigabe erfolgt feature basiert nach erfolgreicher Validierung von Fachbereich, bevor die Endversion in den Produktivsystem erfolgt.

Querschnittliche Elemente und Feedback-Mechanismen

Die fachliche Validierung durch den Fachbereich begleitet alle Phasen des Prozesses und stellt sicher, dass Anforderungen kontinuierlich überprüft und angepasst werden. Iterative Feedback-Schleifen zwischen Entwickler-Team und Fachanwender*innen ermöglichen frühzeitige Korrekturen und reduzieren das Risiko von Fehlentwicklungen. Dieses

strukturierte Vorgehen erhöhte die Transparenz im Projektverlauf, minimierte Abstimmungsaufwände und trug wesentlich zu einer stabilen und termingerechten Einführung der modernisierten ISDB-Anwendung bei.

5.7 Bewertung der Umsetzung und Zielarchitektur

Dieses Kapitel bewertet die umgesetzte Modernisierung der ISDB-Anwendung sowie die daraus resultierende Zielarchitektur systematisch. Die Analyse erfolgt qualitativ und quantitativ anhand definierter *Key Performance Indicators* (KPIs) nach SMART-Kriterien (spezifisch, messbar, akzeptabel, realistisch, terminiert) und fokussiert die architektonische Wirksamkeit der gewählten Strategie hinsichtlich Wartbarkeit, Integrationsfähigkeit, Migrationsfähigkeit und Performance.

Der Fokus liegt nicht auf technischen Implementierungsdetails, sondern auf der langfristigen Tragfähigkeit des Architekturkonzepts.

5.7.1 Erreichte architektonische Zielsetzungen

Die Umsetzung bestätigt die grundsätzliche Eignung eines API-zentrierten, schichtweise entkoppelten Architekturansatzes (vgl. Kapitel 4.2.1).

Durch eigenständige REST-API Implementierung auf Basis OpenAPI 3.x (vgl. Kapitel 5.2.2) wurde eine klare Trennung zwischen Backend (Geschäftslogik inkl. Persistenz Schicht) und Frontend (Benutzeroberfläche mit Präsentationslogik) erreicht (vgl. Kapitel 4.3.1). Abbildung 28 veranschaulicht, dass Spring-basierte Backend nun als serviceorientierte Plattform fungiert, während das Vue.js Frontend ausschließlich als REST-API Konsument agiert (vgl. Kapitel 5.3). Implizite Framework-Abhängigkeiten wurden dadurch systematisch eliminiert.

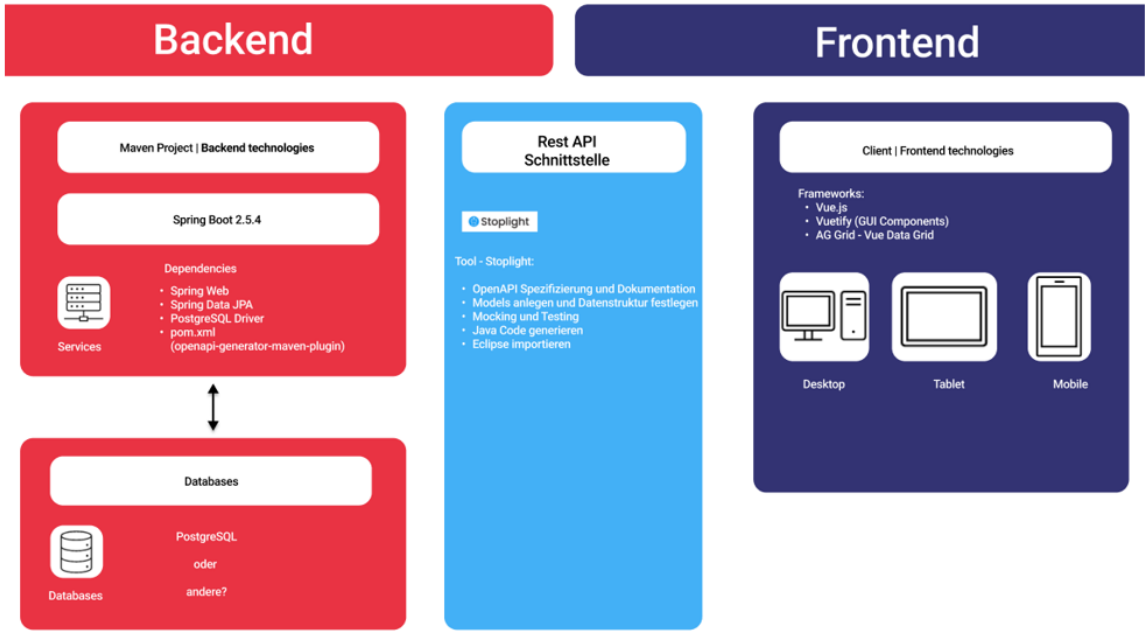


Abbildung 28: ISDB-Zielarchitektur – API-First-Design mit entkopplter Spring Boot / Vue.js Architektur und paralleler Struts-Migration

Der parallele Betrieb von Struts-basierter Legacy-Anwendungsbereich und neuer REST-API (vgl. Kapitel 5.2.1) sowie *RestController* ermöglichte eine schrittweise Migration ohne Unterbrechung des produktiven ISDB-Betriebs. Die Einführung einer zentralen Service-Schicht im Backend verhinderte fachliche Datensatz-Duplikationen und stellte die Konsistenz der Geschäftslogik sicher (vgl. Kapitel 4.3.1). Dieses Vorgehen bestätigt die Eignung der Modernisierungsstrategie für produktionskritische WebGIS-Systeme (vgl. Kapitel 5.6).

Das konsequent verfolgte API-First-Design etablierte die OpenAPI-Spezifikation als vertraglich verbindliche Schnittstelle zwischen Backend, Frontend und weiteren Konsumenten (vgl. Kapitel 5.2.3). Automatisierte Dokumentation mittels Swagger UI, typsichere Client-Generierung sowie Contract-Tests verbesserten sowohl die Testbarkeit als auch die teamübergreifende Koordination signifikant (vgl. Kapitel 4.5).

Das gewählte Frontend-Framework Vue.js erwies sich als geeignete Technologie für den Einsatz im Enterprise-WebGIS-Kontext (vgl. Kapitel 5.4). Die komponentenbasierte Architektur in Kombination mit UI-Framework (Vuetify) und State-Management asynchroner Datenverwaltung inkl. Caching-Mechanismus (TanStack Query) unterstützt die Entwicklung wartbarer und erweiterbarer Kartenanwendungen (vgl. Kapitel 5.5).

5.7.2 Quantitative Bewertung anhand KPIs

Zur Validierung der architektonischen Qualität wurden unten genannten KPIs herangezogen, die Performance, Wartbarkeit und Entwicklungsproduktivität abbilden.

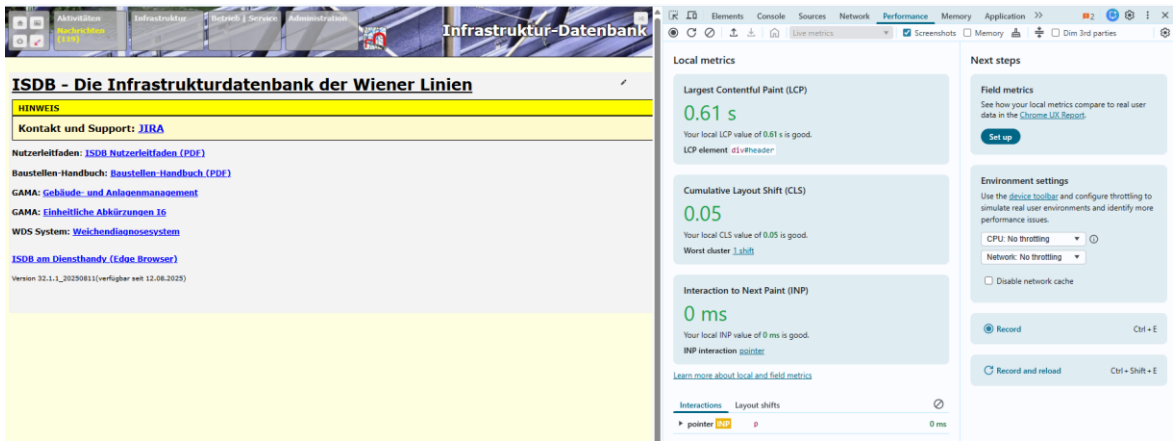


Abbildung 29: Struts Startseite - Performance

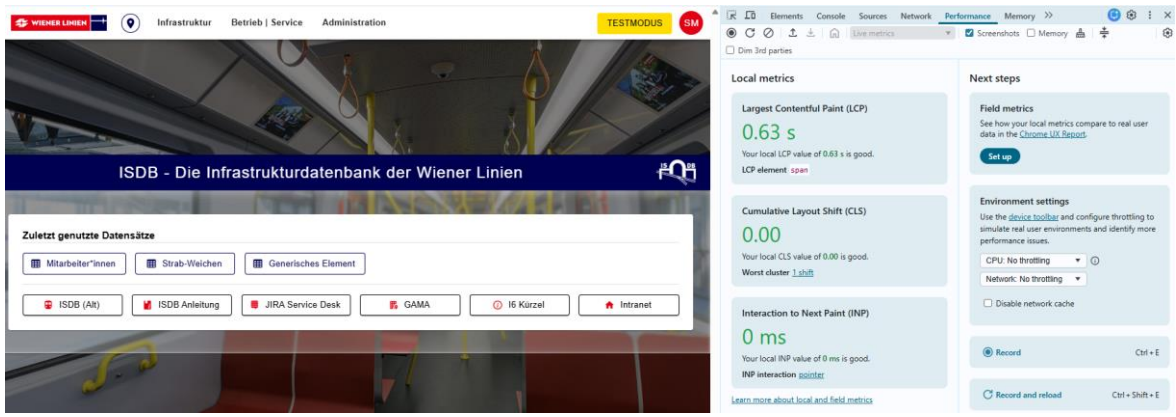


Abbildung 30: Vue.js Startseite – Performance

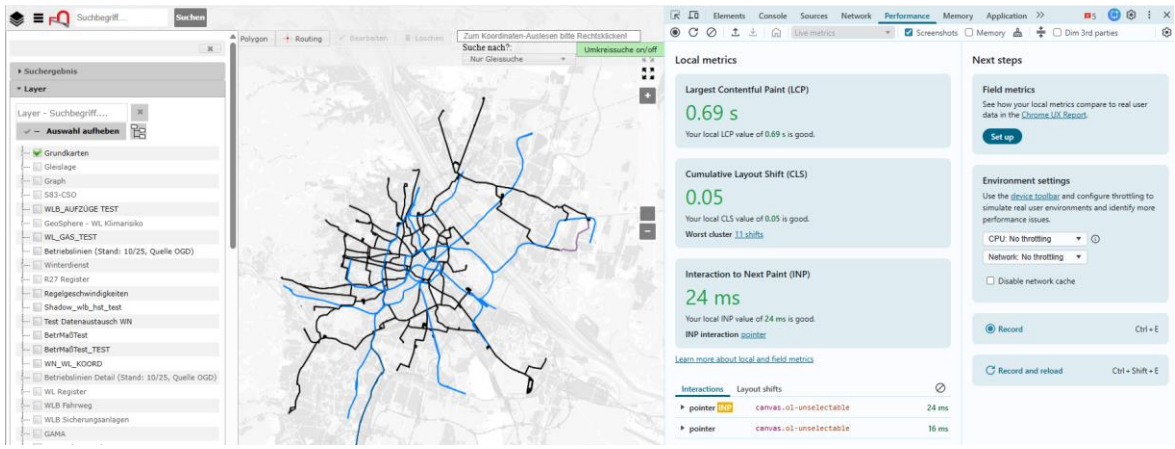


Abbildung 31: Struts Kartenladezeit

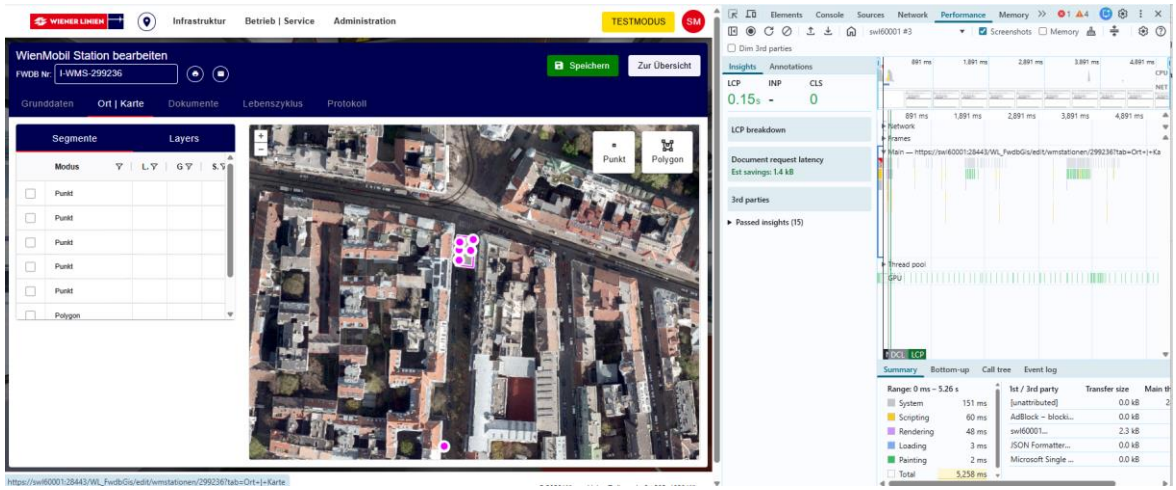


Abbildung 32: Vue.js Kartenladezeit inkl. FeatureCollections

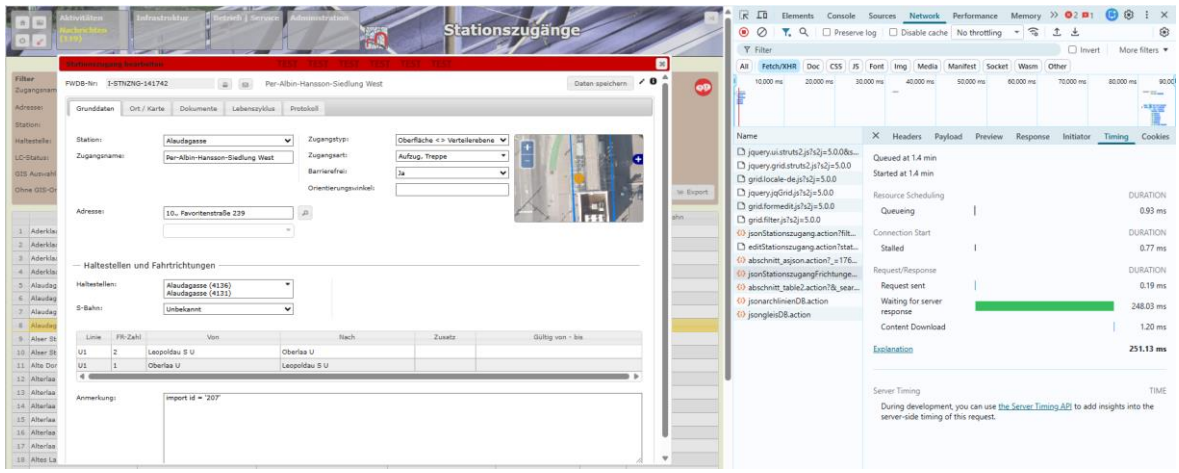


Abbildung 33: Struts API-Response von Stationszugang

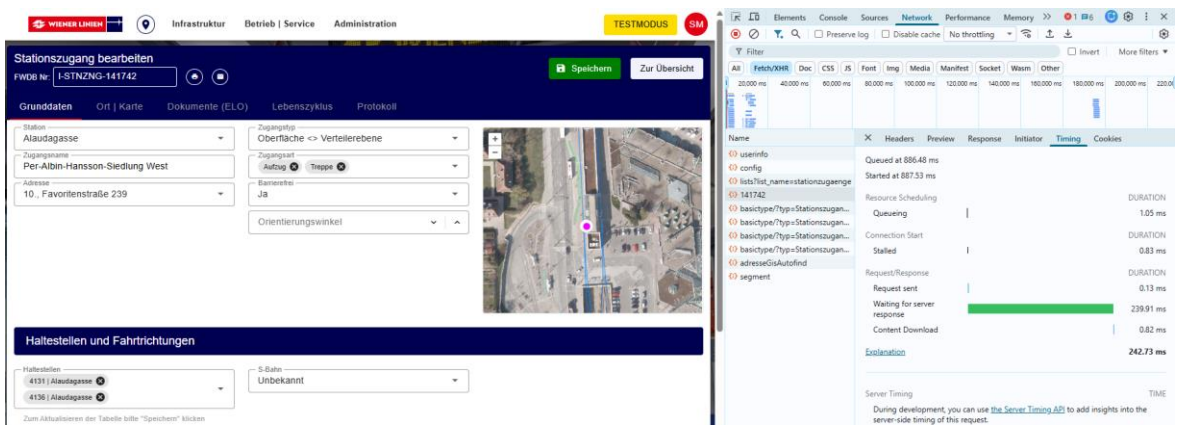


Abbildung 34: Vue.js API-Response von Stationszugang

Tabelle 15: KPI-Entwicklung der ISDB-Modernisierung

KPIs	Struts Legacy (2012)	Zielarchitektur (ab 2023)	Differenz	Beleg
Ladezeit Startseite	0,66s	0,63s	-4,5%	Abb. 17-18
Ladezeit Karten	0,69s	0,15s	-78%	Abb. 19-20
API-Response	248,03 ms	239,91 ms	-3%	Abb. 21-22
Deployments pro Sprint (4W)	o.B. 0,5 – 1	1,5 – 2	+200-300%	Jira Release
Testabdeckung	k.A	~12%	•	
Gepl. PI-Tickets	k.A	~ 100 - 150	•	Jira Board

Die Tabelle 14 dokumentiert die empirische Validierung der Zielarchitektur durch konkrete Performance- und Prozessmetriken. Die Ergebnisse zeigen eine heterogene Wirkung der Modernisierung: Während technische Performance teilweise nur marginal verbessert wurde, ergeben sich signifikante strukturelle Gewinne in Entwicklungsgeschwindigkeit und Modularität.

Startseite-Ladezeit (-4,5%): Die minimale Verbesserung von 0,66s auf 0,63s (Abbildung. 29-30) unterstreicht, dass Struts bei einfachen Seiten trotz Legacy-Charakter performant bleibt, die Server-Side-Rending für Anwendungsinitial gleichgewichtig ist. SPA-Framework (Frist Initial Bundle) kompensiert hier clientseitige Vorteile nicht vollständig.

Kartenladezeit (-78%): Die dramatische Reduktion von 0,69s auf 0,15s (Abbildung. 31-32) belegt die Kernstärke der SPA-Architektur für GIS-Anwendungen. Lazy Loading, TanStack Query Caching und OpenLayers-Optimierung (vgl. Kapitel 5.5) eliminieren synchrone Server-Roundtrips vollständig.

API-Response (-3%): Die marginale Verbesserung (248ms → 240ms, Abbildung. 33-34) zeigt, dass Legacy-Services bereits gut optimiert sind. REST-API und JSON-Serialisierung kompensieren PostGIS-Gewinne nicht signifikant.

Entwicklungs-KPIs: Strukturelle Durchbrüche

Deployments pro Sprint (+200-300%): Der Sprung von 0,5-1 (rückwirkende Validierung durch ehemalige Teammitglieder) auf 1,5-2 Releases (Jira-Daten) zeigt die Wirkung der Modularisierung. Komponentenarchitektur (vgl. Kapitel 5.4) und Feature Flags (vgl. Kapitel 5.6) ermöglichen parallele Entwicklung ohne Legacy-Kopplung.

Testabdeckung (~12%): Erste automatisierte E2E-Tests im Frontend schaffen aktuell Baseline. Ziel: min. 85% (Backend + Frontend)

Geplante PI-Tickets im Durchschnitt 100-150: Die Skalierung der Entwicklungskapazität (Abbildung 35) zeigt Vertrauen in die neue Architektur. Parallelisierung Frontend/Backend (vgl. Kapitel 5.1) ermöglicht höhere Feature-Velocity.

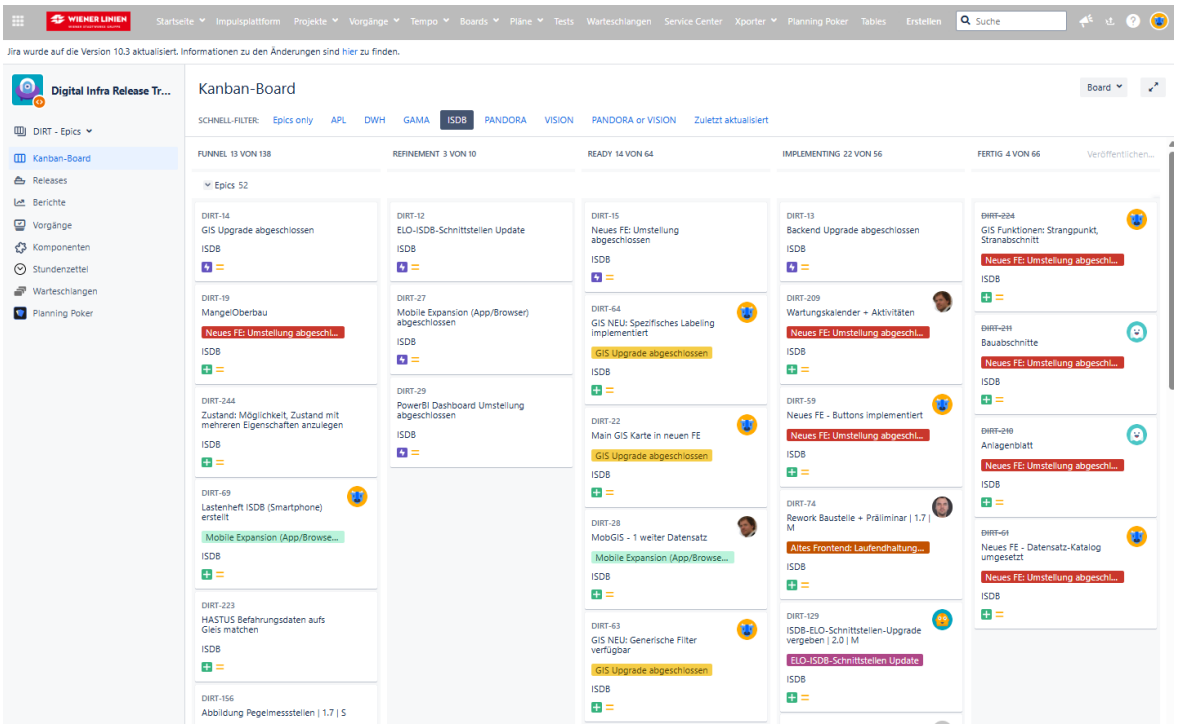


Abbildung 35: Product Increment (PI) Planning – Digital Infra Release Train

Die Modernisierung der ISDB-Anwendung bewirkt keine generische Performance-Steigerung, sondern architektonisch gezielte Verbesserungen, die die spezifischen Anforderungen von WebGIS-Systemen adressieren. Während WebGIS-Kernfunktionen wie Karten- und Geodatenverarbeitung massive Gewinne verzeichnen (Ladezeit -78%), zeigen generische Seiten die bekannte Stärke des Legacy-SSR-Systems gegenüber SPA-Framework. Gleichzeitig werden Entwicklungsprozesse deutlich effizienter, da Deployment-Frequenz um 200-300% steigt, strukturelle Engpässe bei Wartbarkeit und Fehlerbehebungen werden laufend behoben.

Strategische Korrektheit der Transformation: Die Modernisierung adressiert gezielt GIS-spezifische Performance- und Interaktionsanforderungen und eliminiert fundamentale Legacy-Architekturdefizite (Wartbarkeit, Kopplung), ohne generische Funktionalitäten zu verschlechtern. Die hybride Zielarchitektur erweist sich als optimal für produktionskritische WebGIS-Migrationen: Sie bewahrt bewährte Legacy-Stärken und integriert selektiv moderne SPA-Framework-Vorteile, wodurch die ISDB-Systemlandschaft auf State-of-the-Art-Niveau transformiert wird.

Dennoch handelt es sich um eine Übergangslösung, da langfristig der vollständige Ausbau des Struts-Legacy-Systems empfehlenswert ist. Das Backend sollte ausschließlich auf REST-Controller und standardisierte REST-API-Schnittstellen (OpenAPI 3.x) konsolidiert werden, da diese mittlerweile als branchenüblicher Standard gelten und zukunftssicher sind.

6 Zusammenfassung und Ausblick

Dieses Kapitel fasst die zentralen Inhalte und Ergebnisse der vorliegenden Masterarbeit zusammen und ordnet sie in Bezug auf die eingangs formulierte Fragestellung ein. Auf Basis der Zielarchitektur, der Umsetzung und der Bewertung werden die wesentlichen Erkenntnisse zusammengeführt sowie Schlussfolgerungen, Empfehlungen und ein Ausblick abgeleitet.

6.1 Fragestellung und Zielsetzung der Arbeit

Ziel dieser Masterarbeit war es, zu untersuchen, wie eine produktionskritische Legacy-WebGIS-Anwendung schrittweise in eine moderne, wartbare und performante Webarchitektur überführt werden kann, ohne den laufenden Betrieb zu gefährden. Als Fallbeispiel diente die Infrastruktur-Datenbank der Wiener Linien (ISDB), eine historisch gewachsene WebGIS-Anwendung auf Basis einer monolithischen Apache-Struts-Architektur.

Die zentrale Forschungsfrage (vgl. Kapitel 1.4) lautete:

„Wie kann eine Legacy-WebGIS-Anwendung durch eine entkoppelte Zielarchitektur nachhaltig modernisiert werden?“

Zur Konkretisierung wurden drei Arbeitsfragen formuliert, die im Rahmen der Konzeption, technischen Umsetzung und Bewertung systematisch untersucht wurden.

6.2 Fazit der Forschungsfragen

Arbeitsfrage 1:

„Welche Kriterien sind bei der Auswahl eines geeigneten Frontend-Frameworks für WebGIS-Anwendungen entscheidend?“

Die Arbeit zeigt, dass für datenintensive WebGIS-Systeme insbesondere folgende Kriterien ausschlaggebend sind (vgl. Kapitel 2.4 und 5.3):

- geringe Entwickler-Einstiegshürde und niedriger SPA-Framework-Overhead
- komponentenbasierte Architektur zur inkrementellen Migration
- gute Integration mit GIS-Bibliotheken (z. B. OpenLayers)
- effiziente Verarbeitung von GeoJSON-Datenstrukturen für GIS-Grundfunktionen
- optionale, aber konsistente TypeScript-Unterstützung

Vue.js erfüllte diese Anforderungen im Kontext der ISDB-Migration am besten. Die empirischen Ergebnisse (Kapitel 5.7) zeigen signifikante Verbesserungen bei Kartenperformance und Entwicklungsdurchsatz, wodurch Vue.js als geeignetes Enterprise-WebGIS-Frontend validiert werden konnte.

Arbeitsfrage 2:

„Wie lässt sich eine performante und flexible REST-API-Struktur für GIS-Daten gestalten?“

Eine performante und flexible REST-API für GIS-Daten erfordert die Berücksichtigung struktureller Heterogenität (Punkt/Linie/Polygon) und datenintensiver Vektor Geometrien (vgl. Kapitel 5.2.2.1). In der ISDB-Implementierung wurde dies in folgenden Schritten gelöst.

- **Contract-first OpenAPI 3.x** als „Single Source of Truth":

Die API wird zuerst spezifiziert, dann via Code-Generierung in typsichere Spring-REST-Controller und DTOs überführt. Dies erzwingt Versionierung, ermöglicht Validierung und sichert Konsistenz bei mehreren Konsumenten (SPA-Frontend, Fachabteilungen).

- **Polymorphes Geometry-Modell** mit Discriminator:

```
GeometryDTO (type: Point|LineString|Polygon) → { PointDTO | LineStringDTO |
PolygonDTO }
```

Clients verarbeiten GeoJSON-konforme Features deterministisch ohne Sonderfall-Logik. Spezifische Erweiterungen bleiben somit jederzeit Schema-konform möglich.

- **Performance durch selektive Abfragen:**

```
# GET /api/v1/segmente?bbox=...&limit=100 → FeatureCollection
```

Die Performance wird primär durch bedarfsorientierte Datenbereitstellung erreicht: Endpunkte unterstützen selektive Abfragen (z. B. nach räumlichem Ausschnitt) und liefern fachlich zugeschnittene Ressourcen (z. B. Segmente inkl. Feature), um Datenvolumen, Round-Trips und clientseitige Verarbeitung zu reduzieren. Ergänzend werden räumliche Operationen möglichst datenbanknah über PostGIS realisiert, sodass Filterung und Selektion auf Basis räumlicher Indizes erfolgen und nur tatsächlich benötigte Geometrien über die API übertragen werden.

Die REST-API fungiert damit nicht als Implementierungsdetail, sondern als strategisches Architekturartefakt, das Integrationsfähigkeit und Wartbarkeit langfristig sicherstellt.

Arbeitsfrage 3:

„Welche Herausforderungen ergeben sich bei der Migration von Legacy-WebGIS-Systemen hinsichtlich Architektur, Datenstruktur und Systemintegration?“

Die technisch praktische Umsetzung der ISDB-Modernisierung zeigt, dass die zentralen Herausforderungen weniger in einzelnen Technologien, sondern in der kontrollierten Transformation eines produktionskritischen Gesamtsystems liegen. Insbesondere folgenden Problemfelder erwiesen sich als maßgeblich.

- **Sicherstellung Parallelbetrieb von Legacy- und Neusystem*:***

Die Struts-Anwendung musste während der gesamten Modernisierung produktiv bleiben. Die Herausforderung bestand in der gleichzeitigen Aufrechterhaltung des Betriebs bei paralleler Einführung neuer Komponenten, wobei eine zentralisierte Service-Schicht fachliche Konsistenz zwischen Alt- und Neusystem sicherstellen musste.

- **Hoher Validierungsaufwand:**

Parallelbetrieb erforderte kontinuierliche Abstimmung neuer und bestehender Funktionalitäten. Iterative Reviews mit lauffähigen Prototypen in Testumgebungen ermöglichten frühzeitige Identifikation fachlicher Abweichungen vor produktiver Übernahme.

- **Stabile REST-API:**

Änderungen an Datenstrukturen hätten Frontend und Konsumenten beeinträchtigt. Die Herausforderung lag in der Sicherstellung einer stabilen, konsistenten und versionierbaren Schnittstelle über den gesamten Migrationsprozess hinweg.

- **Organisatorische Steuerung:**

Migration der Systemmodernisierung erfolgte strukturierte Feature-Priorisierung. Ein phasenbasiertes Vorgehensmodell in ISDB-Projektmanagement (vgl. Kapitel 5.6.4) mit Feature-Enablern ermöglichte kontrollierte Legacy-Ablösung und schuf Transparenz für alle Teams.

- **Die erfolgreiche Migration:**

Die Kombinationen aus technischen und organisatorischen Maßnahmen ermöglichten am Ende risikominimierte Legacy-WebGIS-Transformation.

Zusammenfassend zeigt die Umsetzung Modernisierung der ISDB, dass die Migration von Legacy-WebGIS-Systemen erfolgreich bewältigt werden kann, wenn technische Maßnahmen (Service-Abstraktion, API-First, Ziel-Architektur-Muster) konsequent mit organisatorischer Disziplin (iterative Validierung, Phasenmodell) kombiniert werden. Im Fokus stehen dabei die Beherrschung von Komplexität, Systemabhängigkeiten und Übergangsrisiken, die eine schrittweise Transformation erforderlich machen.

Arbeitsfrage 4:

„Welche Lösungsansätze sowie schrittweisen Implementierungsstrategien ermöglichen die erfolgreiche Migration zu einer entkoppelten WebGIS-Architektur?“

Im Rahmen dieser Masterarbeit erfolgte die ISDB-Transformation anhand folgenden schrittweisen Migrationsansatz.

- **Zielarchitektur:**

Einführung einer entkoppelten Architektur mit SPA-Frontend (Vue.js) und Backend Upgrade inkl. Implementierung RestAPI nach OpenAPI Standard bei klarer Trennung von Präsentations- und Geschäftslogik.

- **Übergangsarchitektur:**

Die koexistierende Architektur kombiniert Struts-Actions (Legacy) und REST-Controller (neue API), die beide eine beide eine gemeinsame zentrale Service- / Business-Logic-Layer nutzen. Dadurch wird fachliche Konsistenz gewährleistet und beide greifen gemeinsam auf die ISDB-Datenbank zu, obwohl beide Controller parallel produktiv laufen.

- **Contract-First API-Entwicklung:**

Der Contract-First-Ansatz auf Basis der OpenAPI-Spezifikation ermöglicht durch den Einsatz des OpenAPI-Generator-Plugins die automatisierte Generierung von Spring-Controllern im Backend sowie typsicheren Clients im Frontend. Dabei werden aus den definierten Modellen und Endpunkten (Paths) konsistente Klassenstrukturen für beide Systemseiten erzeugt.

- **GIS-Integration:**

Einsatz von OpenLayers als eigenständige Vue.js-Komponente zur Umsetzung zentraler Kartenfunktionalitäten, einschließlich Layer-Management und Feature-Editing, bei gleichzeitiger vollständiger Erhaltung der bestehenden GIS-Funktionalitäten.

- **Migrationsstrategie:**

Schrittweise, inkrementelle Transformation durch parallelen Betrieb, iterative Validierung und phasenweise Ablösung bestehender Funktionalitäten.

6.2.1 Empfehlungen für vergleichbare WebGIS-Systeme

Für die Modernisierung vergleichbarer produktionskritischer WebGIS-Systeme wird empfohlen, API-First-Ansätze bereits in der Planungsphase zu etablieren und den Hybridbetrieb explizit als befristete Übergangsphase zu definieren. Inkrementelle Migration nach Zielarchitekturkonzept der Software-Anwendung sollte gegenüber risikoreichen Big-Bang-Ansätzen klar bevorzugt werden, da sie technische Stabilität mit organisatorischer Machbarkeit vereint.

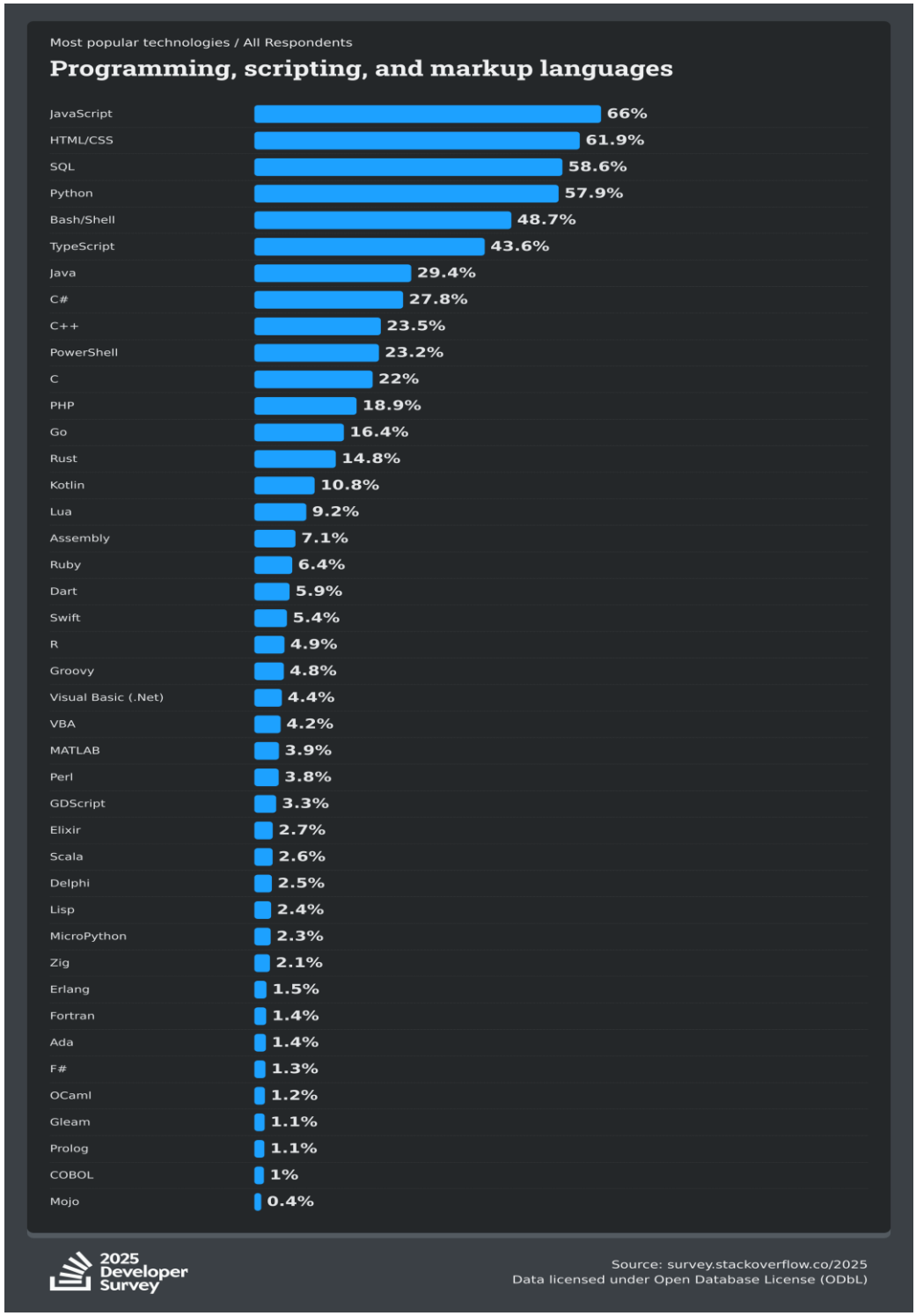
6.2.2 Ausblick und weiterführender Forschungsbedarf

Die Ergebnisse dieser Arbeit eröffnen vielversprechende Perspektiven für die weitere Evolution der ISDB-Architektur sowie vergleichbare WebGIS-Systeme. Technisch ergibt sich Potenzial in der Zerlegung monolithischer Backends in domänenspezifische Microservices, dem Einsatz ereignisgetriebener (Event-Driven) Architekturen für Echtzeit-GIS-Daten sowie Cloud-nativen Deployments mit skalierbaren GIS-Pipelines nach CI/CD Konfiguration Disziplin.

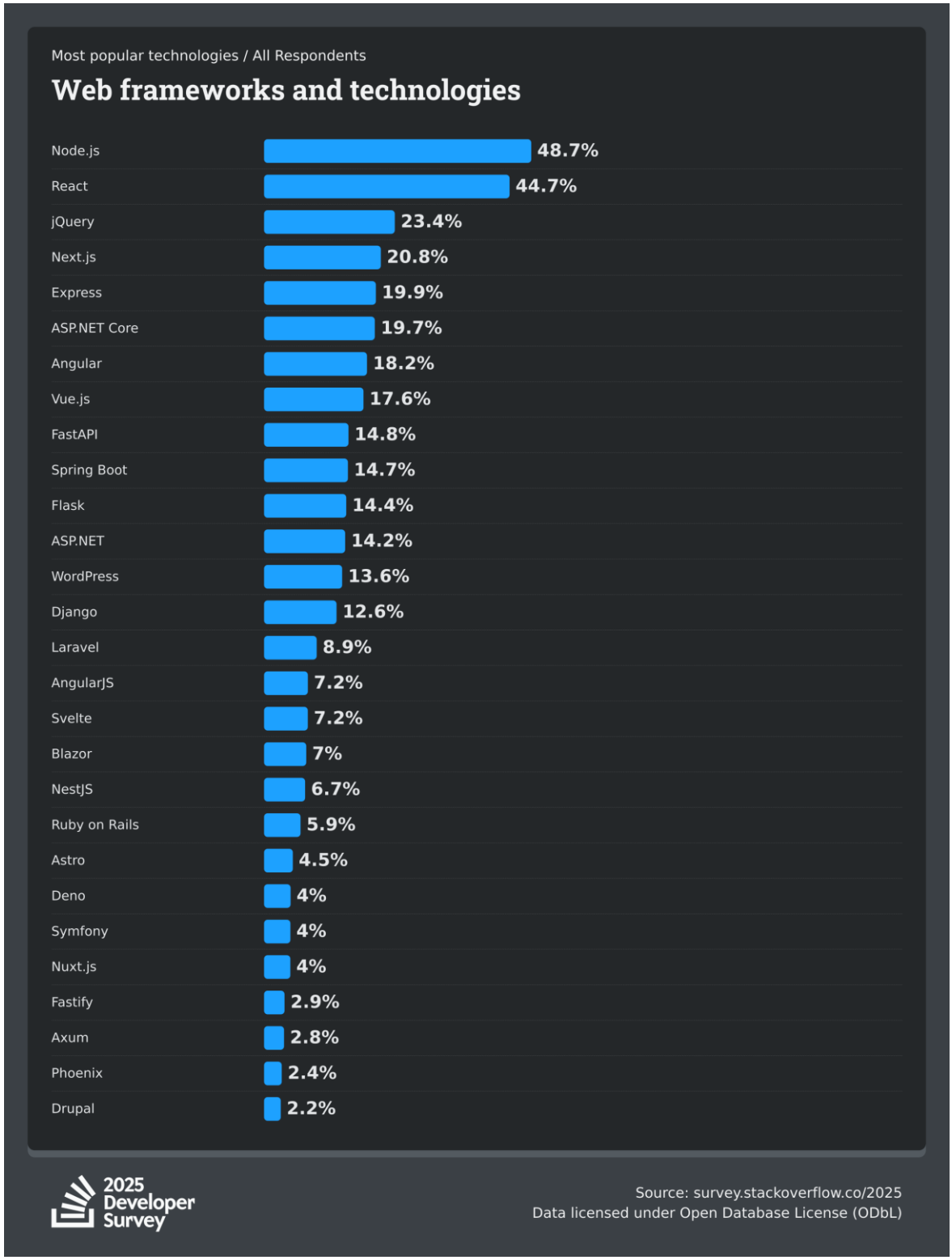
Forschungsbedarf besteht in vergleichenden Studien zu alternativen Frontend-Technologien (Angular vs. React Frameworks) sowie der systematischen Untersuchung organisatorischer Erfolgsfaktoren bei Legacy-Transformationen.

Die ISDB-Modernisierung liefert ein praxisvalidiertes Referenzmodell für die risikominimierte, schrittweise Transformation komplexer WebGIS-Anwendungen im Infrastrukturmanagement – bewährt unter Produktionsbedingungen, zukunftssicher konzipiert.

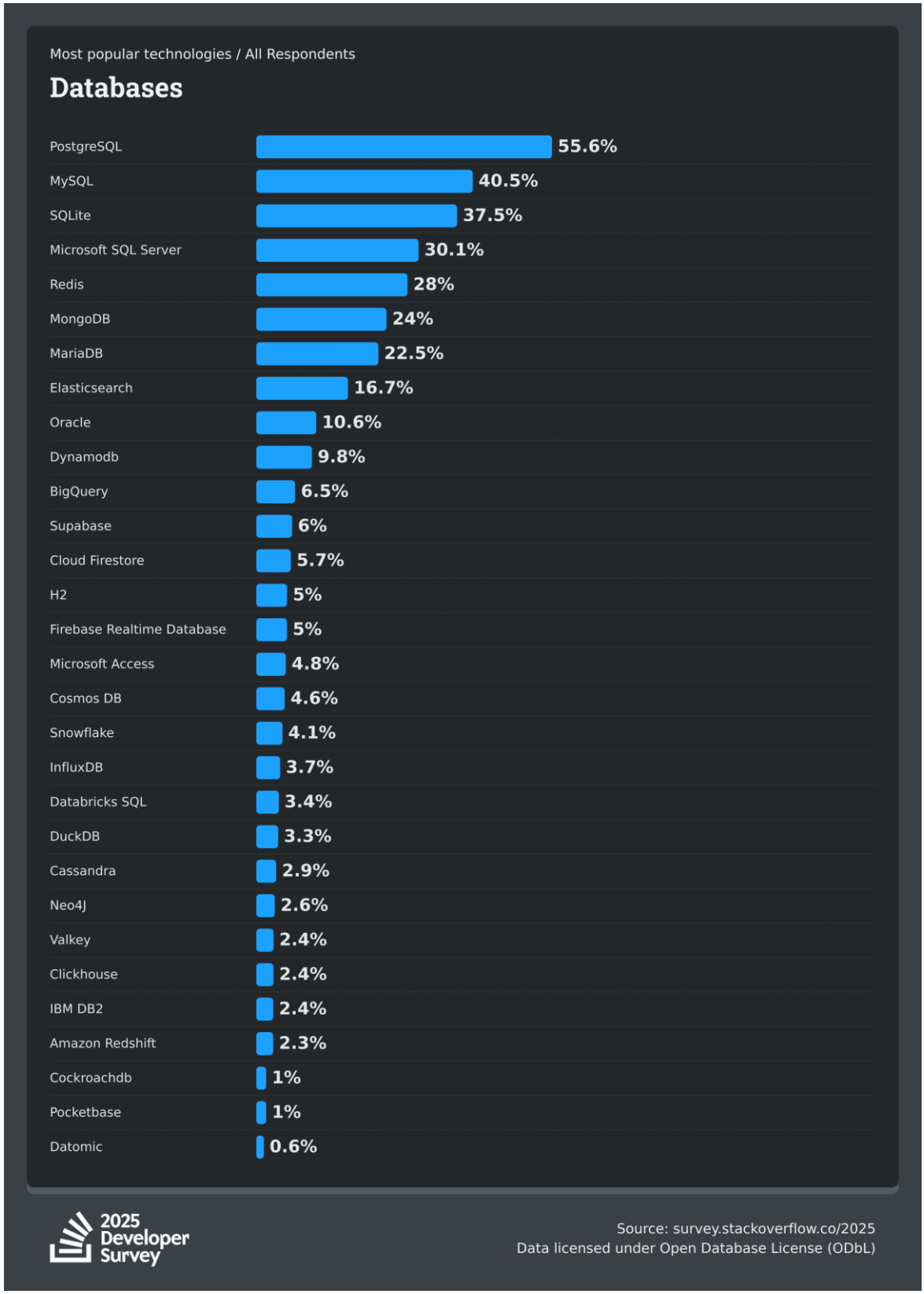
7 Anhang



Anhang 1: Nutzungsanteile von Programmiersprachen, Skript- und Markup-Sprachen (Stack Overflow Developer Survey 2025) – Quelle: https://survey.stackoverflow.co/2025/technology, Zugriff am 25.12.2025



Anhang 2: Nutzungsanteile ausgewählter Web-Frameworks und -Technologien (Stack Overflow Developer Survey 2025) – Quelle: https://survey.stackoverflow.co/2025/technology, Zugriff am 25.12.2025



Anhang 3: Nutzungsanteile von Datenbanksystemen (Stack Overflow Developer Survey 2025) – Quelle: https://survey.stackoverflow.co/2025/technology, Zugriff am 25.12.2025

8 Literaturverzeichnis

AlmaBetter. (2024). Top Front-end Frameworks. https://www.almabetter.com/bytes/articles/top-front-end-frameworks (Zugriff am 29.12.2025)

Annett, R. (2019). Working with Legacy Systems: A practical guide to looking after and maintaining the systems we inherit. Birmingham: Packt Publishing.

Assunção, W.K.G., Marchezan, L., Arkoh, L., Egyed, A. und Ramler, R. (2025). Contemporary Software Modernization: Strategies, Driving Forces, and Research Opportunities, in: ACM Transactions on Software Engineering and Methodology 34, H. 5, Art. 142, 1–35. DOI: https://doi.org/10.1145/3708527

Behncke, K., Hoffmann, K., de Lange, N. & Plass, C. (2009). Web-Mapping, Web-GIS und Internet-GIS – ein Ansatz zur Begriffsklärung. *Kartographische Nachrichten*, 6, 303–308.

Betriebsvereinbarung B6 Digitale Systemarchitektur. (2022). Interne Unternehmensdokumentation, Wien.

Bill, R. (2016). Grundlagen der Geo-Informationssysteme. Wichmann.

Donvir, A. (2024). A comparative analysis of JavaScript unit and end-to-end testing frameworks: Enhancing quality assurance in modern web applications. *Proceedings of the International Conference on Computational Intelligence and Communication Networks*, 1289–1296. https://doi.org/10.1109/CICN63059.2024.10847365

Fayad, M. & Schmidt, D. C. (1997). Object-Oriented Application Frameworks. https://www.dre.vanderbilt.edu/~schmidt/CACM-frameworks.html (Zugriff am 26.12.2025)

Fiby, J. (2004). Fahrwegdatenbank der Wiener Linien: Spezifikation der Bestandsdaten. Wien: Rosinak & Partner Ziviltechniker GmbH. Betriebsinternes Dokument

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Dissertation). University of California, Irvine. https://roy.gbiv.com/pubs/dissertation/ (Zugriff am 18.12.2025)

Flanagan, D. (2006). JavaScript: The Definitive Guide (5. Aufl.). O'Reilly Media.

GeeksforGeeks. (2025). Top Most Popular Java Frameworks for Web Development. https://www.geeksforgeeks.org/blogs/top-most-popular-java-frameworks-for-web-development/ (Zugriff am 18.03.2026)

Haghwerdi-Poor, G. (2010). GIS-Konzept und Konturen eines IT-Master-Plans. Vieweg+Teubner.

Hermans, K. (2023). Mastering Agile: A comprehensive guide to learn agile.

Irani, Z., Abril, R. M., Weerakkody, V., Omar, A. & Sivarajah, U. (2023). The impact of legacy systems on digital transformation in European public administration. *Government Information Quarterly*, 40(1), 101784. https://doi.org/10.1016/j.giq.2022.101784

Jones, D. (2017). JavaScript: Novice to Ninja (2. Aufl.). SitePoint.

Kail, R. (2009-2012). B6 Datenmanagement: Strategisches Programmmanagement zum Komponentenmodell. Kail Consulting (2009), Wien. Projektstatus Infrastruktur-Datenbank (April 2012).

Kaluža, M. & Vukelić, B. (2018). Comparison of front-end frameworks for web applications development. *Zbornik Veleučilišta u Rijeci*, 6(1), 261–282. https://doi.org/10.31784/zvr.6.1.19

Knoche, H. & Hasselbring, W. (2018). Using microservices for legacy software modernization. *IEEE Software*, 35(3), 44–49. https://doi.org/10.1109/MS.2018.2141035

Korduan, P. (2004). Metainformationsystem for Precision Agriculture (Dissertation). Universität Rostock.

Lazar, K. et al. (2025). Generating OpenAPI specifications from online API documentation with large language models. In *Proceedings of the 63rd Annual Meeting of the ACL* (pp. 237–253). https://doi.org/10.18653/v1/2025.acl-industry.18

Mallisetty, M. (2023). RAML vs. OAS: Which is the best API specification for your project? *DZone*. https://dzone.com/articles/raml-vs-oas-which-is-the-best-api-specification-fo (Zugriff am 18.01.2026)

MAPID. (2025). A new era of spatial intelligence. https://www.linkedin.com/pulse/geospatial-trends-2025-in-depth-analysis-how-webgis-ai-cloud-ueroc/ (Zugriff am 01.12.2025)

OpenAPI Initiative. (2024). Learn OpenAPI. https://learn.openapis.org/ (Zugriff am 31.12.2025)

Ott, C., Kral, U. & Brunner, P. H. (2012). Ressourcen- und Umweltmanagement am Beispiel der Wiener Linien. http://hdl.handle.net/20.500.12708/37377

Preim, B. & Dachselt, R. (2010). Interaktive Systeme (2. Aufl.). Springer.

Rahn, H. (2015). WebGIS und WebMapping für Anfänger. Diplomica.

Reddy, M. (2011). API Design for C++. Morgan Kaufmann.

Rudert, F. & Pundt, H. (2008). Standardisierte Geodienste auf mobilen Endgeräten. In *Angewandte Geoinformatik 2008* (S. 305–312). Wichmann.

Seip, C., Korduan, P. & Zehner, M. (2017). Web-GIS: Grundlagen, Anwendungen und Implementierungsbeispiele. Wichmann.

Studer, R. (2013). Konzepte für eine verteilte wissensbasierte Softwareproduktionsumgebung. Springer.

Tyson, M. & Maier, F. (2025). Die besten JavaScript-Frameworks im Vergleich. *Computerwoche*. https://www.computerwoche.de/article/2833386/die-besten-javascript-frameworks-im-vergleich.html (Zugriff am 18.01.2026)

Veenendaal, B., Brovelli, M. A. & Li, S. (2017). Review of Web Mapping: Eras, trends and directions. *ISPRS International Journal of Geo-Information*, 6(10), 317. https://doi.org/10.3390/ijgi6100317

Wagner, J. (2021). API-First, API Design-First, or Code-First. *Stoplight Blog*. https://blog.stoplight.io/api-first-api-design-first-or-code-first-which-should-you-choose (Zugriff am 18.01.2026)

Washizaki, H. (2024). Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) (Version 4.0). IEEE Computer Society.

Wegner, P. (1996). Interoperability. *ACM Computing Surveys*. https://dl.acm.org/doi/pdf/10.1145/234313.234424 (Zugriff am 21.12.2025)

Woehe, J. M. & Kurz, E. (2021). Krisen in Digitalprojekten erfolgreich managen. Hanser.

Wolfart, D. et al. (2021). Modernizing legacy systems with microservices: A roadmap. https://www.researchgate.net/publication/352541897

9 Hilfsmittelverzeichnis – Einsatz von KI-Tools

KI-Hilfsmittel	**Einsatz**	**Betroffene Teile der Arbeit**	**Beleg**
perplexity.ai	Übersetzung von Artikel Textpassagen	Kap. 1.3	Die Veröffentlichungen zum Thema „Legacy System Modernizing“ wurden maschinell übersetzt und anschließend inhaltlich überprüft sowie zusammenfassend in die Arbeit integriert.
perplexity.ai	Generierung von Code Fragmente	1. Kap. 2.4.7.3 2. Kap. 2.4.7.4 3. Kap. 5.2.2	1. Beispiel YAML File wurde generiert. 2. Beispiel Befehle zum Ausführen von OpenAPI-Generator in Frontend und Backend 3. WebSecurityConfig.java wurde zur Darstellung anonymisiert
OpenAPI ChatGPT Instant 5.3	Unterstützung bei sprachlicher Überarbeitung	Kap. 3 - 6	Die Vorschläge wurden ausschließlich zur sprachlichen Optimierung (Grammatik, Stil und Struktur) verwendet. Der fachliche Inhalt wurde eigenständig erstellt und überprüft.
OpenAPI ChatGPT Instant 5.3	Literaturlist sortiert	Kap. 8	Alphabetische Einordnung