Hochschule Augsburg
University of Applied Sciences
Praktikantenamt

Deckblatt für den Praxisbericht

Wahyuasmoro, Anzilal Dimas
(Name, Vorname)

Matrikel-Nr.: 2125752

E-Mail-Adresse: anzilal.wahyuasmoro@hs-augsburg.de

Bachelor-studiengang: Informatik

Betreuer/-in *(in der Fakultät)*: Dr.-Ing. Volodymyr Brakov

Bericht Nr. 1 ☒ 2 ☐
(Anzahl der Berichte nach Vorgabe der Fakultät)

Praxissemester

SS/WS: SS 2025

Ausbildungsstelle:
(Firma, Behörde)
digitalXL GmbH & Co. KG

digitalXL GmbH & Co. KG
Trentiner Ring 2 · 86356 Neusäß
+49 821 999 680-00
digitalxl.de

bitte freilassen

Ausbildungsbeauftragte/-r (Firma):

Andreas Sitschow,
(Name, Tel.-Nr. der/des Ausbildungsbeauftragten)

Bericht gesehen:

(Unterschrift der/des Ausbildungsbeauftragten)

Praktikumszeitraum: vom 17.02.2025 bis 18.07.2025

Berichtszeit: vom 17.02.2025 bis 30.06.2025

Ausbildungsgebiet:
(Thema)
Softwareentwicklung/Testautomatisierung & Systemintegration

Der Unterzeichnende versichert, den Bericht selbstständig und nur unter Zuhilfenahme der genannten Hilfsmittel angefertigt zu haben.

Augsburg, den 9.7.2025
(Ort, Datum)

Dimas
(Unterschrift der/des Studierenden)

Praxisseminar

Hochschule
Augsburg University of
Applied Sciences

Fakultät für
Informatik

Praxisbericht

Studienrichtung
Informatik

Anzilal Wahyuasmoro

Matrikelnummer: 2125752

Ausbildungsstelle: digitalXL GmbH & Co. KG

Prüfer: Dr.-Ing. Volodymyr Brovkov

Abgabedatum: 11.07.2025

Hochschule für angewandte
Wissenschaften Augsburg

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info(at)hs-augsburg-de

Fakultät für Informatik
Telefon +49 821 55 86-3450
Fax +49 821 55 86-3499

Verfasser der Arbeit
Anzilal Wahyuasmoro
86165 Augsburg
anzilal.wahyuasmoro
@hs-augsburg.de

Inhaltsverzeichnis

1. Einleitung

Ich studiere Informatik an der Technischen Hochschule Augsburg und absolviere derzeit mein 22-wöchiges Praxissemester bei der **digitalXL GmbH & Co. KG** im Büro Augsburg im Sigma Technopark. **digitalXL** ist ein Unternehmen im Bereich E-Commerce-Softwareentwicklung mit einem klaren Fokus auf digitale Prozessoptimierung für mittelständische Unternehmen [Gmb25b].

Im Rahmen meines Praktikums bin ich als Junior Developer tätig und habe parallel an drei zentralen Projekten gearbeitet:

- **API-Tester:**
  Entwicklung eines automatisierten Systems zur strukturellen Überwachung von Xentral-API-Endpunkten. Ziel war es, Veränderungen wie neu hinzugefügte oder ersetzte Felder frühzeitig zu erkennen, um das Entwicklungsteam schnellstmöglich zu informieren und Integrationsausfälle zu vermeiden (siehe Abschnitt 3.1).
- **POS-App Alpha/Beta Testing:**
  Durchführung von Alpha- und Beta-Tests der Xentral-POS-App auf macOS und Android, um die Funktionsfähigkeit der Live-Druckfunktion sowie die Fehlerbehebung von Logik- und Formatierungsfehlern zu gewährleisten. Ziel war es, die Anwendung zu optimieren, insbesondere die Druckfunktion und die In-App-Validierungen zu testen und sicherzustellen, dass die POS-App reibungslos mit einem Bondrucker zusammenarbeitet (siehe Abschnitt 3.2).
- **Raspberry Pi POS-Integration:**
  Konfiguration und Einrichtung von Raspberry Pi 4 Modellen als POS-Adapter für das Xentral POS-System. Ziel war es, eine stabile Verbindung zwischen dem POS-System und dem Raspberry Pi herzustellen, damit der Kunde das System eigenständig nutzen kann (siehe Abschnitt 3.3).

Die drei beschriebenen Projekte tragen zur Erweiterung und Stabilisierung des cloudbasierten ERP-Systems Xentral bei[Xen25], welches von digitalXL in vielen seiner Kundenprojekte eingesetzt wird[Gmb25b].

Alle drei Projekte befinden sich derzeit in einem fortgeschrittenen, aber noch nicht abgeschlossenen Zustand. In den folgenden Kapiteln werde ich die Details dieser Projekte sowie die Herausforderungen und Lösungen, die während meiner Arbeit aufgetreten sind, näher erläutern.

1.1 Motivation

Ich habe mich für das Praktikum bei **digitalXL** entschieden, da ich zuvor auf 520-Euro-Basis in der Kommission eines Kundenunternehmens tätig war, das Softwarelösungen von digitalXL einsetzt. Zusätzlich habe ich vor meinem aktuellen Praktikum als Werkstudent bei digitalXL gearbeitet, um mich mit den verwendeten Tools, der Software und dem ERP-System *Xentral* vertraut zu machen. Diese praktische Erfahrung ermöglichte es mir, die zugrunde liegenden Systeme besser zu verstehen und motivierte mich, aktiv an deren Weiterentwicklung mitzuwirken. Während meiner Werkstudententätigkeit konnte ich tiefergehende Einblicke in die Arbeitsweise bei digitalXL gewinnen, was mein Interesse an den technischen Aspekten weiter verstärkte. Nun bin ich im Rahmen meines Praktikums bei digitalXL tätig, um meine Kenntnisse in der praktischen Anwendung weiter auszubauen und an innovativen Projekten mitzuarbeiten.

1.2 Projektziele und/oder Aufgabenbereiche

Im Unternehmen bestand bislang kein automatisierter Mechanismus zur Erkennung struktureller Änderungen in Xentral-APIs. Stattdessen mussten API-Responses manuell überprüft werden. Da viele unserer Kunden stark von der Stabilität bestimmter API-Endpunkte abhängen, ist es essenziell, mögliche Änderungen frühzeitig zu erkennen und die betroffenen Integrationen entsprechend anzupassen.

Darüber hinaus bestand die Anforderung, vier Raspberry Pi Geräte für das Xentral POS-System als Adapter zu konfigurieren. Diese Aufgabe wurde mir übertragen, um eine stabile Verbindung zwischen dem POS-System und den Raspberry Pi Geräten zu gewährleisten. Zusätzlich war ich verantwortlich für die Erstellung einer Kurzanleitung, mit der der Kunde eigenständig zwischen Test- und Liveinstanz wechseln kann.

Ein weiterer wichtiger Teil meines Praktikums war das Alpha- und Beta-Testing der Xentral-POS-App auf macOS und Android. Ziel war es, Fehler in der Anwendung zu identifizieren und die App zuverlässig mit dem POS-System und den angeschlossenen Geräten, insbesondere dem Bondrucker, zu verbinden. Dabei dokumentierte ich alle auftretenden Fehler und Herausforderungen im Support-Ticketsystem und arbeitete mit dem Entwicklerteam an deren Lösung.

1.3 Inhaltsangabe

Im folgenden Bericht wird zunächst das Unternehmen digitalXL näher vorgestellt. Anschließend werden die drei Projekte detailliert beschrieben, an denen ich im Rahmen meines Praxissemesters mitgewirkt habe. Dabei wird auf die jeweiligen technischen Herausforderungen, Lösungsansätze sowie den Entwicklungsstand eingegangen.

Abschließend erfolgt eine persönliche Reflexion des Praxissemesters hinsichtlich der erlernten Kompetenzen, der Arbeitsweise im Team sowie des Mehrwerts für mein weiteres Studium und meine berufliche Zukunft.

2. Unternehmen

Die **digitalXL GmbH & Co. KG** ist eine junge, dynamische Agentur mit Sitz in Neusäß bei Augsburg, die sich auf E-Commerce- und ERP-Consulting spezialisiert hat. Seit ihrer Gründung im April 2021 begleitet digitalXL Kunden bei der Implementierung, Konfiguration und Anpassung des ERP-Systems *Xentral* sowie bei der Entwicklung individueller Reports, Schnittstellen und Multichannel-Strategien. Als Premium-Partner von Xentral bietet das Unternehmen neben Software- und Hardware-Lösungen auch Workshops und Schulungen an, um voll funktionsfähige Gesamtsysteme aus einer Hand zu gewährleisten [Gmb25b].

Xentral-ERP

Xentral ist eine cloudbasierte Enterprise-Resource-Planning-Lösung (ERP), die speziell für kleine und mittelständische E-Commerce-Unternehmen entwickelt wurde. Die Plattform vereint alle Geschäftsbereiche, von Warenwirtschaft und Bestellabwicklung über Lagerverwaltung bis hin zu Buchhaltung und Versand, in einer einzigen Oberfläche. Dank offener APIs lassen sich externe Systeme nahtlos integrieren, und durch modulare Erweiterungen kann jeder Kunde die Software passgenau an seine Prozesse anpassen. Xentral unterstützt sowohl On-Premise- als auch Cloud-Installationen und bietet Standard-Reports sowie flexible BI-Tools für individuelle Auswertungen [Xen25].

2.1 Unternehmensdaten

digitalXL ist als mittelständisches Unternehmen in der Region Augsburg verankert und beschäftigt aktuell über 30 festangestellte Mitarbeitende. Mit mehr als 400 betreuten Kunden aus verschiedenen Branchen konnte digitalXL bereits zahlreiche ERP-Projekte realisieren und setzt dabei auf bewährte Methoden sowie agile Entwicklungsprozesse. Die kumulierte Erfahrung des Teams umfasst insgesamt rund 100 Jahre Xentral-Expertise, eine Zahl, die sich aus den individuellen Berufsjahren aller Entwickler und Consultants zusammensetzt und nicht aus dem Alter des Produktes selbst. Darüber hinaus arbeitet digitalXL eng mit über 20 Partnerfirmen zusammen, um ein umfangreiches Netzwerk für EDI-Anbindungen und Multichannel-Integrationen zu garantieren [Gmb25b].

Geschäftsfelder[Gmb25b]

- **Xentral-Implementierung & Customizing**: Begleitetes ERP-Setup, individuelle Konfiguration, Schnittstellenentwicklung
- **Xentral Reports & BI**: Entwicklung maßgeschneiderter SQL-Berichte und Dashboards
- **Multichannel-Integration**: Anbindung von Marktplätzen und Vertriebskanälen
- **EDI & Systemanbindungen**: Elektronischer Datenaustausch und Automatisierung
- **Schulungen & Workshops**: Onboarding, Training und Best-Practice-Workshops
- **E-Commerce Consulting**: Prozessanalyse, Optimierung und kontinuierliche Betreuung

2.2 Abteilungs-/Teamdaten[Gmb25c]

Innerhalb von digitalXL bin ich dem *Entwicklerteam* zugeordnet, das als zentrales Kompetenzzentrum der Agentur fungiert. Unsere Abteilung ist verantwortlich für die Planung, Umsetzung und Wartung sämtlicher Software- und Prozesslösungen, die digitalXL seinen Kunden anbietet.

Die Teamstruktur ist bewusst flach gehalten, um schnelle Entscheidungswege und intensiven Wissensaustausch zu ermöglichen. Aktuell setzt sich das Entwicklerteam wie folgt zusammen:

- **CTO & Co-Founder**: Strategische Leitung, Governance und finale Qualitätsfreigaben
- **Technical Consultant**: Hauptverantwortung für EDI-Anbindungen, Kunden-Workshops und technischen Support
- **2 Senior Developer**: Betreuung und Weiterentwicklung großer Kernprojekte (ein Entwickler im Büro, ein Entwickler remote)
- **2 Werkstudierende**: Unterstützung bei der Xentral-Integration, Automatisierung von Reports, Datenanalyse, SQL-Reporting und Dashboard-Entwicklung
- **Praktikant (meine Position)**: Unterstützung bei den Projekten API-Tester und POS-App-Anbindung

Das Entwicklerteam arbeitet eng mit den Abteilungen Projektmanagement und Consulting zusammen, um Anforderungen frühzeitig abzustimmen und Umsetzungstermine zuverlässig einzuhalten. Durch die flache Hierarchie hat jede:r Mitarbeitende die Möglichkeit, Verbesserungsvorschläge einzubringen und direkt an Entscheidungsprozessen teilzunehmen.

3. Projekte

Im Rahmen meines 22-wöchigen Praxissemesters bei der **digitalXL GmbH & Co. KG** habe ich parallel an drei zentralen Projekten gearbeitet:

- **API-Tester**: Entwicklung eines automatisierten Monitoring-Tools für API-Antworten. Der Tester liegt in zwei Versionen vor und überwacht strukturverändernde Änderungen, um das Dev-Team frühzeitig zu alarmieren.
- **POS-App Alpha/Beta Testing**: Funktionaler Test einer Xentral-POS-App auf Android und macOS, insbesondere Bon-Druck und Usability.
- **Raspberry Pi POS-Integration**: Einrichtung von Raspberry Pi 4 als POS-Adapter für Bondrucker und Kassensysteme.

Alle drei Projekte befinden sich derzeit in einem fortgeschrittenen, aber noch nicht abgeschlossenen Zustand.

3.1 API-Tester

3.1.1 Projektüberblick und -beschreibung

Im Februar 2025 begann ich im Rahmen meines Praxissemesters bei **digitalXL** mit der Entwicklung eines automatisierten API-Testers für das interne Dev-Team. Ziel war es, das Team durch frühzeitige Benachrichtigungen über strukturverändernde Änderungen in API-Antworten zu entlasten und Integrationsausfälle proaktiv zu vermeiden.

Der API-Tester liegt in zwei Ausführungen vor:

- *Lokale Version:* Diese läuft auf einem lokalen Server mit ngrok[ngr25] und dient als Backup. Sie ist vollständig in JavaScript implementiert und ermöglicht schnelle, interne Tests[dim25c].
- *Globale Version:* Diese wurde mit Deno Fresh[Inc25] vollständig für den globalen Zugriff deployed. Im Zuge der Migration wurde der Code von JavaScript auf TypeScript portiert, um Typensicherheit und Wartbarkeit zu erhöhen[dim25d].

Beide Varianten rufen JSON-Responses verschiedener Endpunkte ab, transformieren sie in ein strukturorientiertes Format und vergleichen sie rekursiv mit vordefinierten Schemata im Verzeichnis `expected/`[1].

[1] Die JSON-Schemata liegen im GitHub-Repository des API-Testers: https://github.com/dimasdigitalXL/ApiTester/tree/main/expected, Zugriff 27.06.2025.

Zur Umsetzung dieser Abläufe kommen mehrere zentrale Technologien zum Einsatz:

Der Tester verwendet **Slack Interactivity**[Sla25a], um interaktive Benachrichtigungen über **Slash-Commands**[Sla25c] und PIN-Modal-Freigaben direkt im Chat zu ermöglichen. Für die persistente Speicherung von Konfigurationen und Testergebnissen wird `Deno KV`[Lan25b] genutzt. Wiederkehrende Testläufe werden automatisiert über `Deno.cron()`-[Lan25a] angestoßen. Die CI/CD-Prozesse einschließlich Deployment erfolgen über **GitHub Actions**[Git25], wodurch jede Änderung sofort getestet und bereitgestellt werden kann. Ergänzt wird das System durch ein Fresh-basiertes Web-Dashboard[Inc25], das Testergebnisse live visualisiert und manuelle Steuerungsmöglichkeiten bietet.

3.1.2 Technologische Grundlagen

Die folgenden Kernkomponenten ermöglichen den reibungslosen Betrieb des API-Testers:

Slack Interactivity: Bietet interaktive Elemente wie Buttons und Modals direkt im Slack-Chat mittels Block Kit. Callback-Payloads werden serverseitig verifiziert und sensible Aktionen (z. B. Freigaben) per PIN-Modal abgesichert[Sla25a].

Slash-Commands: Nutzer steuern den API-Tester direkt aus Slack über drei Commands[Sla25c]:

- **/datastruc <Endpoint>:**
  Zeigt die gespeicherte JSON-Struktur des übergebenen Endpunkts (z. B. „/datastruc Get View Address“).
- **/start:**
  Löst einen Testlauf aus und sendet eine Slack-Zusammenfassung.
- **/endpoints:**
  Listet alle Endpunkte aus `config.json`.

Deno KV: Transaktionaler Key-Value-Store mit atomaren Transaktionen, TTL und Namespaces. Testergebnisse und Konfigurationen werden versioniert per CRUD-API (`kv.get()`, `kv.set()`, `kv.delete()`) gespeichert[Lan25b].

Deno.cron(): Definiert Cron-ähnliche Zeitpläne (z. B. `"0 10,20 * * *"`) in Deno-Skripten. Automatisierte Testläufe oder Health-Checks werden per SSE live ins Dashboard gestreamt[Lan25a].

GitHub Actions: Automatisiert CI/CD via YAML-Workflows: Checkout, Deno-Installation, Testlauf (`deno test`), Linting und Deployment des Fresh-Frontends. Secrets werden sicher in den Repo-Einstellungen verwaltet[Git25].

Fresh Framework: Serverseitiges Rendering mit Preact-Islands und minimalem Client-JS. JSON-Diffs werden per SSE ins Web-Dashboard gestreamt, Tailwind CSS sorgt für responsives Styling[Inc25].

Rekursiver Prüfalgorithmus

Ein besonders wichtiger Bestandteil ist der rekursive Abgleich zwischen den tatsächlichen API-Antworten und den zuvor definierten JSON-Schemata im Verzeichnis `expected/`. Der Algorithmus prüft tief verschachtelte Objekte und Arrays, ohne dass für jede Ebene zusätzlicher Code notwendig ist.

Die folgende Seite zeigt den vollständigen Code des Prüfalgorithmus, der fehlende, zusätzliche und typabweichende Felder erkennt und sammelt.

Listing 3.1: Kernfunktion zum Vergleich erwarteter und tatsächlicher JSON-Strukturen[2]

```
function compareStructures(expected, actual, path = "") {
  const missingFields = [];
  const extraFields = [];
  const typeMismatches = [];

  // Sonderfall: Arrays → vergleiche nur erstes Element
  if (Array.isArray(expected) && Array.isArray(actual)) {
    if (expected.length > 0 && actual.length > 0) {
      return compareStructures(expected[0], actual[0], path);
    }
    return { missingFields, extraFields, typeMismatches };
  }

  // Wenn kein Objekt → Abbruch
  if (typeof expected !== "object" || typeof actual !== "object" || !
   expected || !actual) {
    return { missingFields, extraFields, typeMismatches };
  }

  // 1. Fehlende Felder und Typabweichungen prüfen
  for (const key in expected) {
    const currentPath = path ? ${path}.${key} : key;
    if (!(key in actual)) {
      missingFields.push(currentPath);
    } else if (
      typeof expected[key] === "object" &&
      expected[key] !== null &&
      typeof actual[key] === "object" &&
      actual[key] !== null
    ) {
      const subResult = compareStructures(expected[key], actual[key],
   currentPath);
      missingFields.push(...subResult.missingFields);
      extraFields.push(...subResult.extraFields);
      typeMismatches.push(...subResult.typeMismatches);
    } else if (typeof expected[key] !== typeof actual[key]) {
      typeMismatches.push({
        path: currentPath,
        expected: typeof expected[key],
        actual: typeof actual[key]
      });
    }
  }

  // 2. Zusätzliche Felder erkennen
  for (const key in actual) {
    if (!(key in expected)) {
      const currentPath = path ? ${path}.${key} : key;
      extraFields.push(currentPath);
    }
  }

  return { missingFields, extraFields, typeMismatches };
}
```

[2]Quellcode auf GitHub: https://github.com/dimasdigitalXL/ApiTester/blob/main/core/compareStructures.js, Zugriff 19.06.2025.

Erklärung:

- **Fehlende Felder:**
  Der erste `for`-Block iteriert alle Schlüssel des erwarteten Objekts und sammelt Pfade, die im `actual`-Objekt fehlen. Dies zeigt direkt an, wenn ein API-Update wichtige Daten nicht mehr liefert.
- **Rekursive Prüfung:**
  Bei Objekt-Werten ruft sich der Algorithmus selbst auf, um verschachtelte Strukturen vollständig zu prüfen. So entfällt manueller Mehraufwand für beliebige Tiefe von JSON-Objekten.
- **Typabweichungen:**
  Unterscheidet der Datentyp zwischen `expected` und `actual`, wird dies in `typeMismatches` dokumentiert. Das verhindert Laufzeitfehler, wenn der Konsumenten-Code einen festen Typ erwartet.
- **Zusätzliche Felder:**
  Felder im `actual`-Objekt, die im Schema nicht definiert sind, werden in `extraFields` gesammelt. Dadurch lassen sich neue oder unerwartete Daten sofort identifizieren.

Fazit: Dieser rekursive Prüfalgorithmus stellt sicher, dass alle strukturellen Änderungen in API-Antworten frühzeitig erkannt werden und liefert dem Team detaillierte Berichte, um Probleme schnell zu beheben.

3.1.3 Architektur & logische Komponenten

Die modulare Architektur des API-Testers gliedert sich in vier Schichten, die jeweils klar abgegrenzte Verantwortlichkeiten übernehmen und so eine hohe Skalierbarkeit sowie Wartbarkeit gewährleisten.

Core Layer: Diese Schicht implementiert die HTTP-Request-Logik in `apiCaller.js` und den rekursiven JSON-Vergleich in `compareStructures.js`. Sie erkennt strukturelle Abweichungen und protokolliert alle Ergebnisse für eine lückenlose Nachverfolgbarkeit (siehe Abschnitt 3.1.2)[dim25b]. Änderungen werden in standardisierten Datenobjekten abgelegt, sodass nachfolgende Schichten diese einheitlich verarbeiten können.

Integration Layer: Der Integration Layer bindet externe Dienste ein und orchestriert die Abläufe zwischen ihnen:

- *Slack Interactivity* über Block Kit, Slash-Commands und PIN-Modals mit serverseitiger HMAC-Verifikation[Sla25a],
- *Cron-Jobs* zur zeitgesteuerten Ausführung der Tests mit `Deno.cron()`[Lan25a],
- optionale API-Gateways oder Webhook-Endpoints zur Erweiterung um weitere Benachrichtigungskanäle.

Fehler und Ausnahmen werden hier abgefangen und an ein zentrales Logging weitergereicht, bevor sie an den UI Layer oder das Persistenz-Subsystem gemeldet werden.

UI Layer: Die Web-Oberfläche basiert auf dem Fresh-Framework[Inc25] und Preact[Aut25]. Testergebnisse und Inline-Diffs werden per SSE-Stream an die Benutzeroberfläche übermittelt und nur die interaktiven Preact-Islands werden clientseitig hydriert[Fre25], was eine schlanke, performante UI ermöglicht. Ein Redux-ähnliches State-Management garantiert, dass View-Komponenten stets mit den aktuellen Testergebnissen und Konfigurationsdaten versorgt werden.

Persistenz Layer: Alle Konfigurationen, Testergebnisse und Freigabe-States werden in `Deno KV` gespeichert, einem transaktionalen Key-Value-Store mit Versionierung und TTL-Unterstützung[Lan25b]. Zusätzlich bietet dieser Layer ein Repository-Interface, das atomare Updates und Rollbacks ermöglicht, um bei fehlerhaften Vergleichen die Datenkonsistenz zu gewährleisten.

Der Testlauf wird über Slack oder die UI gestartet, der Core Layer prüft die Antwort und speichert Abweichungen im Persistenz Layer. Anschließend liefern SSE und Slack-Interactivity die Ergebnisse an UI und Chat zurück. Dank der klar definierten Schichten lässt sich jede Komponente unabhängig weiterentwickeln (siehe GitHub-Repository).

3.1.4 Datenmodell

Erwartete API-Antworten werden durch JSON-Schemas im Verzeichnis `expected/` definiert und geladen[dim25a]. Anschließend findet ein **rekursiver Vergleich** mit der aktuellen Antwort statt (siehe Abschnitt 3.1.2).

Datenmodell-Transformation

Vor dem Abgleich werden alle Werte in den JSON-Antworten durch Platzhalter ersetzt, um sich ausschließlich auf die Struktur zu konzentrieren:

- `Strings` → `"string"`
- `Null-Werte` → `null`
- `Boolean` → `true` / `false`
- `Zahlen` → `0`

Diese Platzhalter sichern eine rein strukturelle Prüfung: Änderungen an konkreten Feldwerten werden ignoriert, nur Abweichungen in der Key-Architektur fallen auf.

Ein typisches Schema (Beispiel `Get_View_Address.json`)[3] sieht so aus:

```
{
  "data": {
    "id": "string",
    "type": "string",
    "name": "string",
    "street": "string",
    "zip": "string",
    "city": "string",
    "country": "string",
    "state": "string",
    "addressSupplement": "string",
    "department": "string",
    "subDepartment": "string",
    "contactDetails": {
      "email": "string",
      "phone": "string",
      "gln": null
    },
    "deliveryDetails": null
  },
  "links": {
    "self": "string",
    "customer": "string"
  }
}
```

[3]Beispiel-Schema: https://github.com/dimasdigitalXL/ApiTester/blob/main/expected/Get_View_Address.json, Zugriff 27.06.2025

Vergleichsprozess

Im Anschluss an die Transformation führt der Tester einen rekursiven Abgleich durch, um drei Klassen von Abweichungen zu ermitteln:

- **Fehlende Felder**: Erwartete Keys, die in der Antwort fehlen.
- **Zusätzliche Felder**: Unerwartete Keys in der Antwort.
- **Typabweichungen**: Wenn der tatsächliche Datentyp nicht dem Platzhalter-Typ entspricht.

Verschachtelte Objekte und Arrays werden dabei bis in die tiefsten Ebenen mit dem ersten Element verglichen. Alle Differenzen werden protokolliert und in einer neuen Datei mit Suffix `_updated.json` abgelegt. Nach Freigabe per Slack-Interactivity (inklusive PIN-Verifikation) wird `config.json` entsprechend angepasst, sodass künftige Vergleiche gegen das aktualisierte Schema laufen[4].

3.1.5 Arbeitsablauf

Februar–März 2025: Aufbau eines JavaScript-Prototyps zur strukturierten Ausführung von HTTP-Requests gegen die Xentral-API und Vergleich der JSON-Antworten mit lokalen Schemata. Ein ngrok-Tunnel[ngr25] stellte die Erreichbarkeit sicher, und erste Slack-Interactivity-Benachrichtigungen ermöglichten Feedback direkt im Entwickler-Channel[Sla25a].

April–Mai 2025: Erweiterung der Slack-Interactivity um Slash-Commands (`/start`, `/endpoints`, `/datastruc`) und PIN-geschützte Modals. Parallel erfolgte die Migration auf Deno, da die offizielle Dokumentation derzeit hauptsächlich auf TypeScript ausgerichtet ist[Den25], wodurch Typensicherheit und Modularität weiter verbessert wurden. Zudem wurde ein Mechanismus zur Erkennung neuer API-Versionen implementiert und die Unterstützung mehrerer Slack-Workspaces integriert, um die Skalierbarkeit zu erhöhen[Sla25a].

Juni 2025: Automatisierung durch zwei Cron-Jobs für tägliche Testläufe um 10 Uhr und 20 Uhr, Aufbau einer CI/CD-Pipeline sowie das Deployment und die Live-Visualisierung über das Fresh-basierte Web-Dashboard (siehe Abschnitt 3.1.7).

Beide Versionen setzen Cron-Jobs ein: Die lokale Version läuft auf dem Entwickler-Laptop und benötigt kontinuierliche Stromversorgung, selbst im gesperrten Bildschirmmodus bleibt der Cron-Job aktiv, solange das Gerät nicht ausgeschaltet wird. Im Gegensatz dazu werden die Cron-Jobs der globalen Deno-Version auf einem Server ausgeführt und laufen unabhängig vom lokalen Rechner weiter.

[4] Aktualisierung in https://github.com/dimasdigitalXL/ApiTester/blob/main/config.json, Zugriff 27.06.2025

3.1.6 Beispielhafte Slack-Ausgabe

In Abbildung 3.1 ist zu sehen, wie ein täglicher Cron-Job um 10 Uhr ausgeführt wurde und eine Nachricht in unserem Entwickler-Workspace auf der Slack-Plattform erzeugt wurde[Sla25b]. Die Nachricht listet fehlende, zusätzliche und typabweichende Felder auf und bietet die Buttons **„Einverstanden“** und **„Warten“** zur Freigabe oder Verzögerung der Anpassung. Am unteren Rand befindet sich eine Gesamtstatistik mit der Anzahl erfolgreicher Tests, Warnungen und kritischen Fehlern.

Abbildung 3.1: Beispielhafte Slack-Benachrichtigung eines Cron-Jobs mit Interaktions-Buttons und Teststatistik

Nach dem Klicken auf den **„Einverstanden“**-Button öffnet sich ein PIN-Modal, um die Freigabe der Änderungen zu bestätigen.Nach erfolgreicher Eingabe aktualisiert der Bot das erwartete Schema in `config.json`, sodass künftige Vergleiche gegen die neue Struktur erfolgen. Gleichzeitig ersetzt er die ursprüngliche Nachricht durch eine Bestätigung, die den freigebenden Nutzer nennt (siehe Abbildung 3.2).

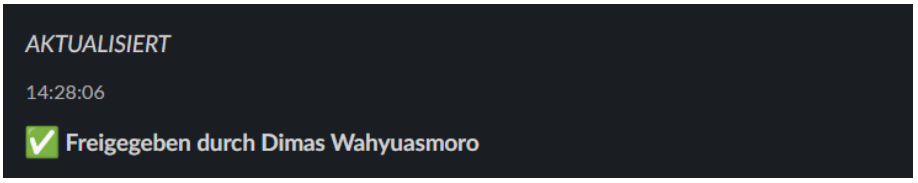

Abbildung 3.2: Bestätigung der Freigabe durch den Nutzer

Anschließend wird der Testlauf sofort wiederholt. Dabei tauchen die zuvor fehlenden Felder nicht mehr auf, da die Änderungen nun als gültig gespeichert sind, alle Tests schlagen erfolgreich an und die Slack-Nachricht spiegelt den neuen Status wider.

Wählt der Nutzer hingegen **„Warten“**, bleibt der Endpunkt im Wartestatus und wird beim nächsten Lauf erneut geprüft. Die ursprüngliche Nachricht bleibt unverändert, bis eine Freigabe erfolgt, sodass Änderungen erst nach abschließender Prüfung übernommen werden können.

3.1.7 Web-Dashboard mit Deno Fresh

Seit Juni 2025 steht der API-Tester zusätzlich als interaktive Web-Applikation auf Basis des Fresh-Frameworks zur Verfügung (siehe Abbildung 3.3). Beim ersten Aufruf erscheinen in der Mitte vier Buttons: Der zentrale, besonders hervorgehobene Button **„Tests starten“**, darunter die Buttons **„Endpunkte“** und **„Verfügbare Routen“**[5], sowie ganz rechts der Button **„Vergleich“**, der zunächst deaktiviert ist.

Betätigt der Nutzer den Button *„Tests starten“*, so wird direkt unterhalb desselben der Hinweis eingeblendet, dass die Tests erfolgreich ausgeführt und eine Slack-Nachricht gesendet wurde. Klickt man anschließend auf *„Endpunkte“*, öffnet sich in der linken Spalte eine Übersicht aller in `config.json` konfigurierten Endpunkte. Wählt man hier einen Eintrag aus, erscheint in der rechten Spalte die zu diesem Endpunkt gespeicherte JSON-Datenstruktur (aus dem Ordner `expected`), und der Button *„Vergleich“* wird aktiviert.

Ein Klick auf *„Vergleich“* zeigt dann unten in der Mitte eine Inline-Diff-Ansicht, die die alte und die neue Strukturversion direkt gegenüberstellt. Alle Unterschiede sind farblich markiert:

- **Fehlende Felder:**
  Einträge, die in der alten Struktur vorhanden, in der neuen aber nicht mehr enthalten sind. Dies signalisiert, dass möglicherweise API-Antworten nicht mehr alle erwarteten Daten liefern, was Integrationsfehler verursachen kann.
- **Neue Felder:**
  Einträge, die in der neuen Antwort hinzugekommen sind und zuvor nicht existierten. So erkennt man schnell Erweiterungen oder unerwartete Zusatzinformationen, die das Datenmodell verändern.
- **Typabweichungen:**
  Felder, bei denen sich der Datentyp geändert hat (z. B. von String auf Number). Solche Abweichungen können zu Laufzeitfehlern führen, wenn der Verbrauchercode auf einen bestimmten Typ angewiesen ist.

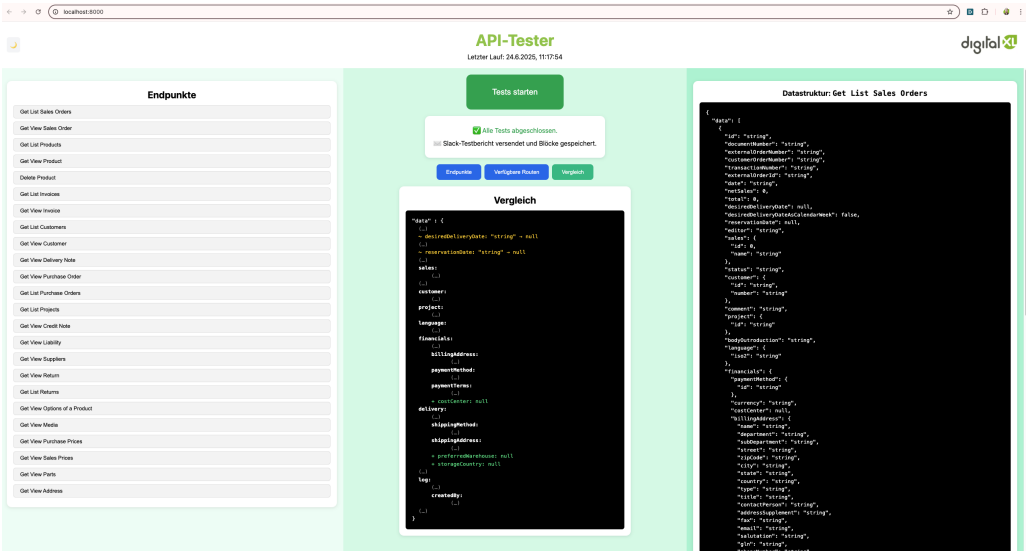

Abbildung 3.3: Web-Dashboard des API-Testers (Fresh Framework, Stand 18. 06. 2025)

Unter https://digitalxl-fresh-api-tester.deno.dev/ kann die Anwendung direkt getestet werden.

[5] Nur für Debugging; listet alle in der API registrierten Routen auf.

3.1.8 Eigene Arbeitspakete

Im Verlauf dieses Projekts habe ich das gesamte System von Grund auf neu entwickelt. Zu Beginn erstellte ich die Core-Komponenten, wobei ich die HTTP-Request-Logik in `apiCaller.js` und den JSON-Vergleich in `compareStructures.js` implementierte (siehe Abschnitt 3.1.2).

Anschließend entwickelte ich die Slack-Integration, einschließlich interaktiver Slash -Commands und PIN-geschützter Modals (vgl. Abschnitt 3.1.2). Um den Testablauf zu automatisieren, richtete ich zwei Cron-Jobs mit `Deno.cron()` ein und entwickelte eine GitHub Actions-Pipeline, die bei jedem Push die Tests ausführt und das Fresh-Frontend automatisch deployt (vgl. Abschnitt 3.1.2).

Schließlich implementierte ich das Fresh/Preact-basierte Web-Dashboard, erstellte SSE-Streams für Live-Updates (siehe Abschnitt 3.1.7) und sorgte für die Persistenz aller Konfigurations- und Testergebnisdaten in Deno KV (vgl. Abschnitt 3.1.4). Alle Komponenten wurden so konzipiert, dass sie nahtlos miteinander kommunizieren und einen stabilen und skalierbaren API-Testprozess ermöglichen.

Während der Entwicklung erhielt ich kontinuierlich Erweiterungen und Feedback von den Mitgliedern des Entwicklerteams (vgl. Abschnitt **??**). Diese Zusammenarbeit führte dazu, dass der API-Tester fortlaufend optimiert und erweitert wurde.

3.1.9 Ausblick & Weiterentwicklung

Für die Zukunft sind mehrere Erweiterungen geplant: Erstens soll die Slack-Integration direkt ins Web-Dashboard verlagert werden, sodass Freigaben und Interaktionen ohne Kanalwechsel möglich sind. Zweitens ist die Absicherung über ein OAuth 2.0-Modul vorgesehen, um den Zugriff auf den Tester und das Dashboard zu schützen[OAu25]. Drittens werden automatisierte Unit- und Integrationstests eingeführt, um Kernmodule systematisch zu prüfen und Regressionen frühzeitig zu erkennen[Atl25]. Parallel dazu ist eine Code-Bereinigung geplant, um redundante Komponenten zu entfernen und die Projektstruktur zu straffen. Schließlich soll ein Alert-System realisiert werden, das bei kritischen Testergebnissen automatische Benachrichtigungen per E-Mail oder SMS versendet, um das Team umgehend zu informieren.

3.1.10 Herausforderungen

Eine zentrale Herausforderung stellte der anfängliche Verzicht auf automatisierte Unit- und Integrationstests dar. Ohne diese Tests musste jede Änderung per Hand validiert und mit Postman[Pos25] gegen die Xentral-API abgeglichen werden, was erheblichen Mehraufwand und intensive Analyse der Konsolenausgaben erforderte.

Zudem war die Slack-Integration aufwendiger als erwartet: Da zur lokalen Entwicklung ngrok-Tunnel genutzt wurden, änderte sich die weitergeleitete URL bei jedem Neustart, sodass in den Slack-App-Einstellungen für Interactivity und Slash-Commands regelmäßig die Request-URL angepasst werden musste. Die Implementierung interaktiver Slash-Commands und PIN-Modals erforderte mehrere Iterationsschleifen, bis eine nahtlose Nutzererfahrung gewährleistet war. Trotz dieser Hürden konnte durch kontinuierliche Verbesserungen sichergestellt werden, dass der API-Tester zuverlässig läuft und die Anforderungen des Entwicklerteams erfüllt.

3.2 POS-App Alpha/Beta Testing

3.2.1 Projektüberblick

Das Projekt begann im April und wurde parallel zu weiteren internen Initiativen durchgeführt. Die Xentral-POS-App wurde auf macOS und Android getestet[Gmb25a], um die Integration mit dem vorhandenen EPSON TM-m30II (Model M362B) Bondrucker zu prüfen und Logik- sowie Formatierungsfehler frühzeitig zu erkennen und zu beheben.

3.2.2 Durchgeführte Tests und Bug-Reporting

Zunächst wurde die App auf einem Firmen-Mac installiert und erste Testdrucke mit dem Bondrucker durchgeführt. Anschließend wurden diese Tests auf ein Samsung Android-Tablet übertragen und sichergestellt, dass sowohl das Tablet als auch der Drucker im selben Netzwerksegment (Haupt-WLAN) betrieben wurden, da sonst die IP-Adresse des Druckers nicht gefunden werden konnte. Alle auftretenden Fehler und Schwierigkeiten wurden fortlaufend in einem einzigen Ticket im Support-Ticketsystem dokumentiert. Die Einträge wurden mit Screenshots und gelegentlich mit kurzen Videos (z. B. zur Veranschaulichung von Verzögerungen oder fehlender Anmeldeberechtigung) ergänzt. Nach jeder Rückmeldung des Entwicklerteams wurden die Fehler reproduziert, die Korrekturen verifiziert und die Ergebnisse im Ticket vermerkt, bevor mit dem nächsten Testzyklus fortgefahren wurde.

3.2.3 Ergebnisse

Ab Juni funktioniert die Live-Druckfunktion wie erwartet. Die App zeigt nun In-App-Notifications für Druckstart und erfolgreichen Druckabschluss an. Durch die Inline-Validierung werden negative Rabattwerte verhindert, und die Bon-Formatierung entspricht einem herkömmlichen Kassenbon.

3.2.4 Herausforderungen

Während der Testläufe zeigte sich eine Verzögerung von etwa fünf bis acht Sekunden zwischen dem Knopfdruck in der App und dem tatsächlichen Bondruck, was ohne visuelles Feedback zu unbeabsichtigten Mehrfachdrucken führte. Zudem war es essenziell, dass Tablet und Drucker im selben Netzwerksegment lagen, da ansonsten keine Verbindung aufgebaut werden konnte. Hinzu kam, dass zu Beginn nicht über die notwendigen Berechtigungen verfügt wurde, um die POS-App zu installieren und sich anzumelden, was die Testvorbereitung erheblich verzögerte. Über längere Zeit blieb unklar, warum der Verkaufs-Button in der App keinen Bon auslöste, obwohl bei den Tests des Support- und Entwicklerteams keine Probleme auftraten. Diese Diskrepanz führte zu wiederholten, langwierigen Abstimmungen im Support-Ticket-Chat und verlängerte den gesamten Testzyklus. Außerdem wurde dem EPSON-Setup-Guide[Eps25] gefolgt und die nötige Software installiert, um WLAN zu aktivieren. Erst später stellte sich heraus, dass das Modell diese Funktion nicht unterstützt, weshalb der Bondrucker per LAN-Kabel in das Hauptnetzwerk integriert werden musste.

3.2.5 Meine Arbeitspakete

Ich war verantwortlich für die Installation und Konfiguration der Xentral-POS-App auf macOS und Android sowie für die systematischen Alpha- und Beta-Tests inklusive ausführlichem Bug-Reporting im Support-Ticketsystem. Darüber hinaus habe ich dem Support-Team per Ticket konkrete Verbesserungsvorschläge übermittelt, zum Beispiel für die Einführung von In-App-Notifications zur Anzeige von Druckstart und Druckende, für die Inline-Validierung negativer Rabattwerte und für die Anpassung der Bon-Formatierung. Anschließend habe ich die von den Entwicklern umgesetzten Änderungen überprüft und validiert.

3.3 Raspberry Pi POS-Integration

3.3.1 Projektüberblick

Ab April wurde im Rahmen eines Kundenprojekts fünf Raspberry Pi 4 Model B als POS-Adapter eingerichtet. Ein Gerät dient dem internen Test, vier wurden für den Kunden vorkonfiguriert und gemeinsam mit einer Kurzanleitung zum Instanzwechsel ausgeliefert. Ziel war es, automatisierte Drucktests zu ermöglichen und dem Kunden das selbstständige Wechseln zwischen Xentral-Testinstanz und Liveinstanz zu erleichtern.

3.3.2 Konfiguration und Kundenanleitung

Die internen Einrichtungsschritte umfassten das Flashen von Raspberry Pi OS auf eine microSD-Karte [Ras25b], die Installation aller notwendigen Pakete (unter anderem PHP 7.4 und Xentral-Adapterbox) sowie das Auslesen der Seriennummern mit `cat /proc-/cpuinfo`. Die Seriennummern wurden zusätzlich auf die Gehäuse geklebt, sodass sie von außen sichtbar sind. Intern wurden diese Details dokumentiert. Für die Einrichtung des WLAN und bei Bedarf der LAN-Verbindung wurde die offizielle Raspberry Pi OS Netzwerkkonfiguration [Ras25a] genutzt. Die dem Kunden übergebene Kurzanleitung beschränkt sich auf den Instanzwechsel: Sie erklärt, wie man im Konfigurationsfile die IP-Adresse des Raspberry Pi und die Seriennummer anpasst, um zwischen Test- und Liveinstanz zu wechseln.

3.3.3 Arbeitsablauf

Zunächst studierte ich die offizielle Raspberry Pi OS Dokumentation [Ras25c] und führte die dort beschriebenen Grundkonfigurationen durch. Anschließend prüfte ich über das Webinterface der Xentral POS Instanz, ob jeder Raspberry Pi korrekt registriert war.

3.3.4 Ergebnisse

Vier vorkonfigurierte Raspberry Pi sind als Adapterboxen in der Xentral POS Instanz registriert und stabil mit dem System verbunden. Die kompakte Kurzanleitung zum Instanzwechsel ermöglicht es dem Kunden, eigenständig zwischen Test- und Liveinstanz zu wechseln.

3.3.5 Herausforderungen

Die SSH-Verbindung vom Mac war instabil, sodass auf eine lokale Konfiguration per HDMI umgestiegen werden musste. Das Gast-WLAN blockierte den Druckerzugriff, sodass eine LAN-Verbindung im Hauptnetzwerk erforderlich war. Außerdem mussten die fehlenden Anweisungen in der Xentral-Dokumentation dem Betreuer gemeldet werden, und am folgenden Tag wurde eine Lösung bereitgestellt.

3.3.6 Meine Arbeitspakete

Ich habe fünf Raspberry Pi 4 Model B mit Raspberry Pi OS geflasht und eingerichtet. Auf Anweisung meines Entwicklungschefs installierte ich alle Systempakete und richtete die Xentral-Adapterbox ein. Die Seriennummern las ich aus, hinterlegte sie in der Konfiguration und klebte sie auf die Gehäuse, sodass sie von außen sichtbar sind. Für den Kunden erstellte ich eine Kurzanleitung, die erklärt, wie man anhand der Raspberry Pi IP-Adresse und Seriennummer in der Konfigurationsdatei zwischen Test- und Liveinstanz wechselt. Abschließend stellte ich sicher, dass alle vier Geräte in der Xentral POS Instanz korrekt registriert sind und Druckaufträge zuverlässig ausführen.

4. Persönliche Stellungnahme

In diesem Kapitel möchte ich meine Eindrücke und Erfahrungen während meines Praktikums bei digitalXL teilen. Dabei geht es um meinen Arbeitsalltag, den Arbeitsumfang, meine Highlights sowie Verbesserungsvorschläge.

4.1 Wie sieht Ihr Arbeitsalltag aus?

Mein Arbeitsalltag war abwechslungsreich. Ich begann zwischen 8:00 und 8:30 Uhr und arbeitete bis 17:00 Uhr mit einer 45-minütigen Mittagspause. Täglich hatten wir um 9:00 Uhr einen Stand-Up, in dem wir die Aufgaben des Vortages besprachen und uns für den Tag abstimmten. Außerdem gab es dienstags und donnerstags regelmäßige Team-Meetings, bei denen wir uns über den Fortschritt austauschten und neue Anforderungen besprachen. Die restliche Zeit verbrachte ich mit der Arbeit an meinen Projekten, wobei ich viel selbstständig und fokussiert arbeiten konnte.

Zusätzlich gab es regelmäßige firmeninterne Events, die den Arbeitsalltag auflockerten und das Teamgefühl stärkten. Gemeinsame Mittagessen, Projektvorstellungen oder informelle Treffen sorgten dafür, dass die Stimmung stets gut blieb und ich Kolleg:innen auch persönlich besser kennenlernen konnte.

4.2 Arbeitsumfang

Während meines Praktikums war ich an drei Projekten beteiligt:

- **API-Tester:**
  Entwicklung eines Systems zur Überwachung der Xentral-API (siehe Abschnitt 3.1).
- **POS-App Alpha/Beta Testing:**
  Systematische Alpha- und Beta-Tests der Xentral-POS-App auf macOS und Android, inklusive Testdrucke mit dem EPSON TM-m30II und detailliertem Bug-Reporting (siehe Abschnitt 3.2).
- **Raspberry Pi POS-Integration:**
  Konfiguration des Raspberry Pi als Adapterbox für das Xentral-POS (siehe Abschnitt 3.3).

Ich war für die gesamte Umsetzung verantwortlich und konnte meine Fähigkeiten in der Backend-Entwicklung und der Arbeit mit Embedded-Systemen deutlich ausbauen.

4.3 Was hat Ihnen gefallen?

Die Arbeit bei digitalXL hat mir sehr gut gefallen. Besonders spannend fand ich die Kombination aus verschiedenen Technologien, vom API-Tester über die Raspberry Pi-Integration bis hin zur POS-App. Die offene und freundliche Teamatmosphäre war eine große Motivation, und ich konnte meine Arbeit weitgehend selbstständig gestalten. Besonders gefreut hat mich, dass ich die Möglichkeit hatte, meine Arbeitszeiten flexibel zu gestalten und den Code gründlich zu dokumentieren. Der lockere Austausch im Team und die schnelle Hilfe bei Unklarheiten habe ich sehr geschätzt.

4.4 Haben Sie Verbesserungsvorschläge?

Ein Verbesserungsvorschlag für das API-Tester-Projekt wäre, das Projekt nicht allein, sondern im Tandem mit einem weiteren Praktikanten oder Kollegen durchzuführen. Besonders bei komplexen Aufgaben wie dem globalen Deployment wäre Unterstützung hilfreich, um Ideen zu diskutieren und Fehler schneller zu identifizieren. Eine kontinuierliche Zusammenarbeit von Anfang an würde den Entwicklungsprozess effizienter gestalten.

4.5 Was haben Sie gelernt?

Besonders wertvoll für mich war, wie verschiedene Komponenten, Backend, Embedded-Hardware und Web-UI, zusammenspielen, um ein vollständiges System zu schaffen. Dabei konnte ich nicht nur meine technischen Fähigkeiten erweitern, sondern auch ein tieferes Verständnis für die Gesamtsystem-Architektur entwickeln (vgl. Abschnitt 3.1.3).

Abschließend würde ich ein Praktikum oder einen Job bei digitalXL jedem empfehlen:
Die vielseitige Arbeit, das angenehme Team und die Freiheit bei der Gestaltung der Arbeitszeit machen es zu einer sehr positiven Erfahrung.

Literatur

[Atl25] Atlassian. *The different types of software testing*. Zugriff am 27.06.2025. 2025. URL: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing.

[Aut25] Preact Authors. *Preact – Fast 3kB Alternative to React*. Zugriff am 27.06.2025. 2025. URL: https://preactjs.com/.

[Den25] Deno Land. *Web Development Fundamentals – TypeScript in Deno*. Zugriff am 27.06.2025. 2025. URL: https://docs.deno.com/runtime/fundamentals/web_dev/.

[dim25a] dimasdigitalXL. *API-Tester (lokale Version) – expected JSON-Schemata*. Zugriff am 27.06.2025. 2025. URL: https://github.com/dimasdigitalXL/ApiTester/tree/main/expected.

[dim25b] dimasdigitalXL. *API-Tester Core Components (apiCaller.js, compareStructures.js)*. Zugriff am 27.06.2025. 2025. URL: https://github.com/dimasdigitalXL/ApiTester/tree/main/core.

[dim25c] dimasdigitalXL. *ApiTester – lokale Version (JavaScript) Source Code*. Zugriff am 27.06.2025. 2025. URL: https://github.com/dimasdigitalXL/ApiTester.

[dim25d] dimasdigitalXL. *fresh-api-tester – globale Version (TypeScript) Source Code*. Zugriff am 27.06.2025. 2025. URL: https://github.com/dimasdigitalXL/fresh-api-tester.

[Eps25] Epson. *Epson TM-m30II (Model M362B) – WLAN-Einrichtung*. Zugriff am 27.06.2025. 2025. URL: https://download4.epson.biz/sec_pubs/bs/html/m001274/de/chap04_4.html.

[Fre25] Fresh Contributors. *Concepts – Islands Architecture*. Zugriff am 27.06.2025. 2025. URL: https://fresh.deno.dev/docs/concepts/islands.

[Git25] Inc. GitHub. *GitHub Actions Documentation*. Zugriff am 27.06.2025. 2025. URL: https://docs.github.com/actions.

[Gmb25a] Xentral GmbH. *Einrichtung des POS*. Zugriff am 27.06.2025. 2025. URL: https://help.xentral.com/hc/de/articles/360016758359-Einrichtung-des-POS.

[Gmb25b] digitalXL GmbH & Co. KG. *Über digitalXL*. Zugriff am 13.06.2025. 2025. URL: https://www.digitalxl.de.

[Gmb25c] digitalXL GmbH & Co. KG. *Unser Team*. Zugriff am 19.06.2025. 2025. URL: https://www.digitalxl.de/team/.

[Inc25] Deno Land Inc. *Fresh – The next-gen web framework for Deno*. Zugriff am 27.06.2025. 2025. URL: https://fresh.deno.dev/.

[Lan25a] Deno Land. *Deno Cron Documentation*. Zugriff am 27.06.2025. 2025. URL: https://docs.deno.com/deploy/kv/manual/cron/.

[Lan25b] Deno Land. *Deno KV Documentation*. Zugriff am 27.06.2025. 2025. URL: https://deno.land/manual@v1.37.4/runtime/kv.

[ngr25] ngrok, Inc. *ngrok – Secure introspectable tunnels to localhost*. Zugriff am 27.06.2025. 2025. URL: https://ngrok.com/.

[OAu25] OAuth Community Initiative. *OAuth 2.0*. Zugriff am 27.06.2025. 2025. URL: https://oauth.net/2/.

[Pos25] Postman, Inc. *Postman API Platform*. Zugriff am 27.06.2025. 2025. URL: https://www.postman.com/.

[Ras25a] Raspberry Pi Documentation. *Configuration – Host a wireless network*. Zugriff am 27.06.2025. 2025. URL: https://www.raspberrypi.com/documentation/computers/configuration.html#host-a-wireless-network-from-your-raspberry-pi.

[Ras25b] Raspberry Pi Documentation. *Raspberry Pi Imager and Software*. Zugriff am 27.06.2025. 2025. URL: https://www.raspberrypi.com/software/.

[Ras25c] Raspberry Pi Documentation. *Raspberry Pi OS Documentation*. Zugriff am 27.06.2025. 2025. URL: https://www.raspberrypi.com/documentation/.

[Sla25a] Inc. Slack Technologies. *Interactivity and Shortcuts in Slack apps*. Zugriff am 27.06.2025. 2025. URL: https://api.slack.com/interactivity.

[Sla25b] Inc. Slack Technologies. *Slack Platform – API Overview*. Zugriff am 27.06.2025. 2025. URL: https://api.slack.com/.

[Sla25c] Slack Technologies, Inc. *Slash Commands*. Zugriff am 27.06.2025. 2025. URL: https://api.slack.com/interactivity/slash-commands.

[Xen25] Xentral. *ERP-System Xentral*. Zugriff am 19.06.2025. 2025. URL: https://www.xentral.com/.