

XmasWishes Projekt - Ergebnisbericht

Cyril Wagner
Universität Trier

January 21, 2025

Einleitung

In diesem Dokument wird das Projekt **XmasWishes** vorgestellt, das im Rahmen des zweiten Übungsblatts umgesetzt wurde. Ziel des Projekts ist die Konzeption und (teilweise) Implementierung eines Wunschverwaltungssystems für den Weihnachtsmann, das sowohl *digitale* als auch *postalische* Eingänge aus aller Welt verarbeiten kann. Die folgenden Themenbereiche werden behandelt:

- **Architektur und Modellierung (Teilaufgabe 1)**
- **Konkretisierung möglicher Softwaretechnologien (Teilaufgabe 2)**
- **Implementierung eines Prototyps (Teilaufgabe 3)**
- **Integration von Papierbriefen via Apache Camel (Teilaufgabe 4)**

Dieser Bericht enthält Links bzw. Verweise auf die Code-Repositories sowie Diagramme zu den relevanten Architekturkonzepten.

1 Teilaufgabe 1: Architektur- und Modellierungsvorschlag

Das XmasWishes-System soll geografisch verteilte Datenhaltung bieten, um die saisonal hochschnellenden Anfragen im letzten Quartal bewältigen zu können.

Abbildung 1 zeigt ein mögliches Architekturdiagramm mit Microservices, KI-Service, Gift-/Logistik-Service sowie einer globalen Datenbank-Lösung:

1.1 Wichtige Aspekte

- **Microservices-Ansatz:** Entkopplung und bessere Skalierbarkeit
- **Datenschutz (DSGVO):** Pseudonymisierung, Verschlüsselung, Rollen- und Rechte-management
- **Status-Workflow:** Jeder Wunsch durchläuft verschiedene Bearbeitungsstufen (z. B. *nur formuliert* → *in Bearbeitung* → *in Auslieferung* → *unter dem Weihnachtsbaum*)

Weitere Details, wie *Datenbankschema* oder *Nachrichtenbus*, sind im Architekturenwurf berücksichtigt.

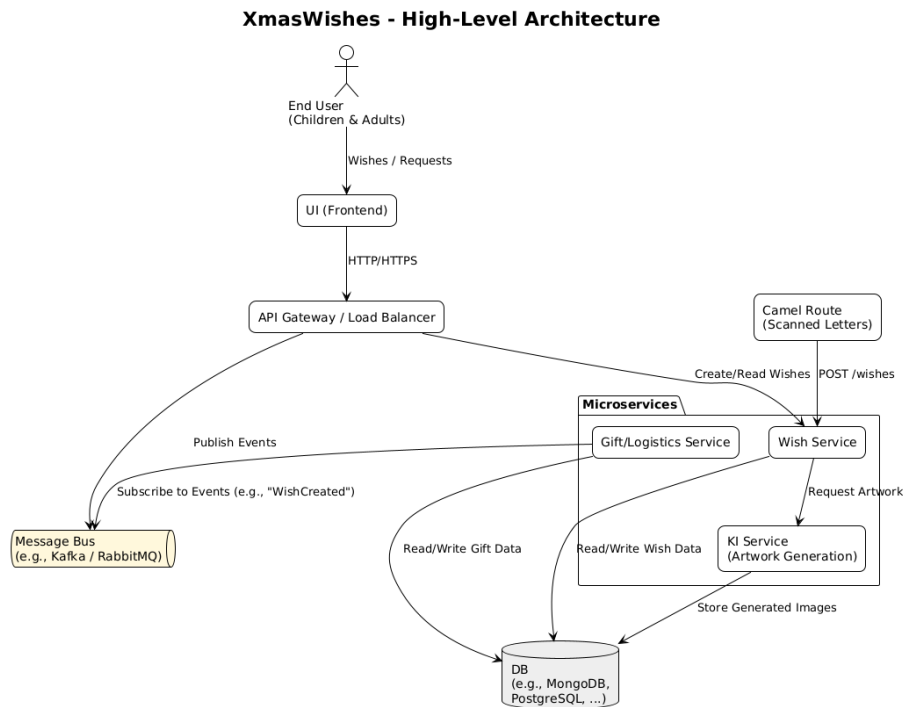


Figure 1: Architekturübersicht von XmasWishes

2 Teilaufgabe 2: Auswahl von Softwaretechnologien

In Teilaufgabe 2 wurden konkrete Technologien vorgeschlagen, um die oben genannte Architektur in die Praxis umzusetzen. Beispiele:

- **Backend:** Node.js (Express), Java (Spring Boot) oder Python (Flask/FastAPI)
- **Datenbank:** MongoDB (Atlas), PostgreSQL (AWS Aurora), DynamoDB (NoSQL) etc.
- **Nachrichtenbus:** Apache Kafka, RabbitMQ, AWS SQS
- **KI-Service:** Containerisierte KI-Modelle (z. B. Stable Diffusion, DALL·E-API etc.)

Die Wahl fiel beispielhaft auf Node.js mit Express, MongoDB, Kafka (als Event-Bus) sowie Docker/Kubernetes für die Orchestrierung.

3 Teilaufgabe 3: Prototyp-Implementierung

Ein einfacher Prototyp wurde erstellt, der folgende Kernfunktionalität demonstriert:

- **REST-Endpoint /wishes (POST, GET)** zum Anlegen und Auslesen von Wünschen
- **In-Memory- oder DB-gestützte** Speicherung der Wünsche
- **Lasttest-Skript** (Python) zum Messen der Requests/sec

3.1 Ausschnitt aus dem Quellcode (Asynchroner Python-Test)

Nachfolgend ein Auszug aus dem asynchronen Python-Skript, das mit Hilfe von `aiohttp` und `asyncio` Lasttests gegen das XmasWishes-System durchführt. Hier werden in exponentiell steigender Anzahl gleichzeitige Requests gesendet, bis eine Obergrenze erreicht wird. Das Skript misst unter anderem die Dauer sowie die *Requests pro Sekunde (RPS)* und schreibt die Ergebnisse in eine Excel Datei. Die Obergrenze wurde gewählt, da eine größere Anzahl an Requests eine sehr lange Exekutionszeit gebraucht hätten. Dies lässt sich aus den Ergebnissen auslesen. Wir haben den Schnitt aus unseren Resultaten bis 1.5 Millionen genommen und die Anzahl der Requests auf

Listing 1: Asynchroner Loadtest mit aiohttp

```
1 import asyncio
2 import aiohttp
3 import time
4 import random
5 import math
6 import xlswriter
7
8 BASE_URL = "http://localhost:3000"
9
10 async def create_wish(session):
11     """
12     Asynchronous function to send a POST request using aiohttp session.
13     """
14     url = f"{BASE_URL}/wishes"
15     data = {"text": f"Ich_w nsche_mir_ein_Geschenk_Nr.{random.randint(1,9999)}"}
16     async with session.post(url, json=data) as response:
17         return response.status
18
19 async def test_load(num_requests):
20     """
21     Runs num_requests asynchronous tasks in parallel to create wishes.
22     """
23     async with aiohttp.ClientSession() as session:
24         tasks = [create_wish(session) for _ in range(num_requests)]
25         results = await asyncio.gather(*tasks, return_exceptions=True)
26     return results
27
28 async def main():
29     max_requests = 1.5e6
30     exponent = 1.1
31     number = 100
32
33     results_for_excel = []
34
35     while True:
36         num_requests = math.ceil(number ** exponent)
37         number = num_requests
38         if num_requests > max_requests:
39             break
40
41         print(f"\nStarting_load_test_with_{num_requests}_requests...")
42         start_time = time.time()
43
44         results = await test_load(num_requests)
45
```

```

46     duration = time.time() - start_time
47     rps = len(results) / duration if duration else 0
48     status_codes = set(results)
49
50     print(f"Done! {len(results)} requests in {duration:.2f} seconds.")
51
52     print(f"Requests/sec: {rps:.2f}")
53     print(f"Status codes: {status_codes}")
54
55     results_for_excel.append([num_requests, duration, rps])
56
57     # Write results to Excel
58     workbook_name = "load_test_results_async.xlsx"
59     workbook = xlswriter.Workbook(workbook_name)
60     worksheet = workbook.add_worksheet("Results")
61
62     worksheet.write(0, 0, "Number of Requests")
63     worksheet.write(0, 1, "Duration (seconds)")
64     worksheet.write(0, 2, "Requests/Second")
65
66     for i, row_data in enumerate(results_for_excel, start=1):
67         worksheet.write(i, 0, row_data[0])
68         worksheet.write(i, 1, row_data[1])
69         worksheet.write(i, 2, row_data[2])
70
71     workbook.close()
72     print(f"\nAll results have been saved to '{workbook_name}'.")
73
74 # If you run this script directly:
75 if __name__ == "__main__":
76     asyncio.run(main())

```

3.2 Messung der Ergebnisse

Bei einer lokalen Ausführung wurden ca. **3000 Requests/sec** erreicht, primär limitiert durch das lokale System und den einfachen Node.js/Express-Server. Figure 2 zeigt die Ergebnisse, 1.5 Milliarden Requests hätten mehr als 6 Tage Laufzeit benötigt.

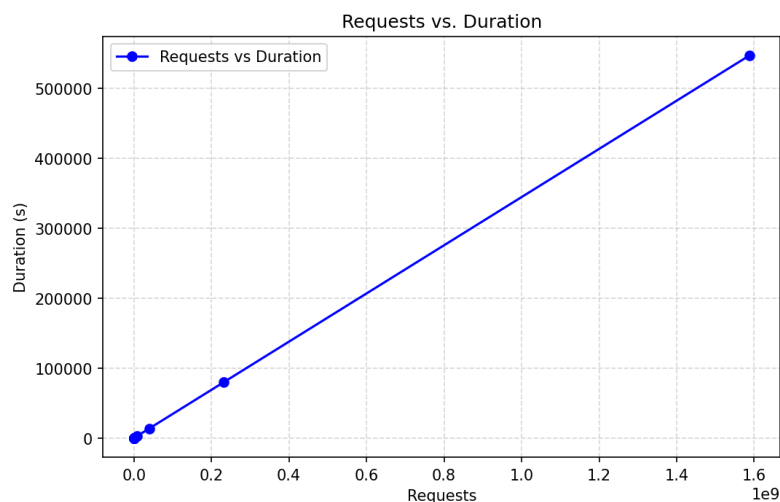


Figure 2: Duration of Requests with around 3000 Requests/Second

4 Teilaufgabe 4: Apache Camel-Lösung

Ein Teil der Wünsche kommt weiterhin in Papierform. Diese Briefe werden am Nordpol eingescannt und automatisch via **Apache Camel** eingespielt. In einer Java-Anwendung wurde folgende Camel-Route definiert:

Listing 2: Camel-Route zum Einlesen gescannter Briefe

```
1 from("file:data/scanned-letters?noop=true")
2   .log("New scanned letter file: ${header.CamelFileName}")
3   .process(exchange -> {
4       // Extract file content, e.g. via OCR or simple text read
5       String fileContent = exchange.getIn().getBody(String.class);
6       // Build a JSON body for XmasWishes
7       String requestBody = "{\n\"text\":\n\" + fileContent.replace("\n"
8           , "\n") + "\n}";
9       exchange.getIn().setBody(requestBody);
10  })
11  .to("http://localhost:3000/wishes?httpMethod=POST")
   .log("Letter sent to XmasWishes. Response: ${body}");
```

Sobald eine Datei im Verzeichnis `data/scanned-letters` landet, wird sie:

1. eingelesen,
2. (ggf. per OCR) ausgewertet,
3. als POST an `/wishes` gesendet.

4.1 Integrationstest

Wir konnten durch einfaches Platzieren einer Testdatei im `data/scanned-letters`-Verzeichnis eine neue Wunsch-Erfassung im XmasWishes-System anstoßen. Dies stellt sicher, dass auch die *Offline*-Briefe digitalisiert werden.

5 Fazit und Ausblick

Mit den beschriebenen Architekturansätzen, Technologieentscheidungen und einem ersten Prototyp demonstriert das XmasWishes-Projekt, wie ein hochskalierbares, fehlertolerantes und DSGVO-konformes System für Weihnachtswünsche aufgebaut werden kann.

Künftige Schritte:

- **Ausbau der KI-Funktionalität** zur künstlerischen Generierung der Wunsch-Bilder
- **Weitere Lasttests** mit professionellen Tools (z.B. JMeter, k6)
- **Automatisiertes Deployment in Kubernetes** (CI/CD)

Abgabehinweise:

Dieser Bericht enthält alle relevanten Diagramme (Abbildung 1), die wichtigsten Codeauszüge (Listings 1 und 2) und die Links/Verweise auf unsere Repositories: https://github.com/s4cywagn/SA4E_PF1.

Deadline: 26.1.2025.