

Gr. 1: Heinzelreiter/  
Kurschl*Software Engineering*

Übungen zu SWK5/WEA5 im WS 2021/22

Gr. 2: Pimminger/  
Kurschl

Name \_\_\_\_\_

Aufwand SWK5 in h \_\_\_\_\_

Gr. 3: Sklenitzka/  
NadschlägerAufwand WEA5 in h \_\_\_\_\_

---

## AaaS – Analytics as a Service

Moderne Anwendungen bestehen aus einer Vielzahl von Komponenten und Services, die auf vielfältige Art und Weise miteinander interagieren. Durch die Komplexität der Anwendungsstruktur kann es zu Fehlern kommen, die häufig nur schwer nachvollziehbar sind. Ein probates Mittel zur Fehlererkennung sind Telemetriedaten, die von den Systemkomponenten generiert werden. Um die Fehleranalyse zu vereinfachen bzw. in vielen Fällen überhaupt erst zu ermöglichen, müssen die Telemetriedaten in einem einheitlichen Format zentral gesammelt werden. Dies erlaubt einerseits die Aufbereitung und Visualisierung der Daten, stellt aber andererseits auch die Grundlage für die frühzeitige Identifikation von Fehlersituationen und die automatische Reaktion darauf dar.

Sie werden daher im Rahmen des SWK5/WEA5-Semesterprojekts mit der Implementierung eines Softwaresystems beauftragt, das Telemetriedaten sammeln, aufbereiten und visualisieren kann. Die Software soll auch über ein Warnsystem verfügen, das kritische Systemzustände erkennt und im Bedarfsfall das Servicepersonal darüber informiert.

### Funktionale Anforderungen

Das Softwaresystem erfüllt im Wesentlichen folgende Aufgaben:

- Im Backend werden Telemetriedaten gesammelt, validiert und gespeichert. Es bietet Möglichkeiten zur Abfrage der Daten und versendet bei Bedarf Warnmeldungen.
- Die Visualisierung der Daten erfolgt über eine Web-Anwendung, in welcher Daten grafisch aufbereitet werden. Auch die Konfiguration der Komponenten des Backends gehört zu den Aufgaben dieser Anwendung.
- Als Datenquellen fungieren andere Anwendungen, welche Telemetriedaten an die Kernkomponente weitergeben. Mittels einer .NET-Bibliothek können externe Anwendungen einfach in das Monitoringsystem integriert werden.


Das zu entwickelnde Softwaresystem besteht aus den folgenden Komponenten:

- *AaaS.Core* ist die zentrale Komponente, welche die Geschäftslogik beinhaltet und für die Datenverwaltung zuständig ist. Sie stellt die von *AaaS.Web* benötigte Funktionalität zur Verfügung und kapselt den Zugriff auf die Datenbank. Auch die flexible Verwaltung der Regeln zum Senden von Warnmeldungen und die Anwendung dieser Regeln gehören zu den Aufgaben dieser Komponente.
- *AaaS.ClientSDK* unterstützt die Anbindung einer externen Anwendung an *AaaS.Core*. Diese Komponente kapselt die Implementierungsdetails der Kommunikation mit dem Backend. *AaaS.ClientSDK* ist eine .NET-Bibliothek, die als NuGet-Paket ausgeliefert wird.
- *AaaS.Api* exportiert die von *AaaS.Web* und *AaaS.ClientSDK* benötigte Funktionalität von *AaaS.Core* in Form eines REST-basierten Web-Service.

- *AaaS.Web*: Über diesen Web-Client kann das Backend administriert werden. Die wichtigste Aufgabe dieser Komponente ist aber die grafische Visualisierung der in *AaaS.Core* gesammelten Daten.

### (a) Vorbemerkungen

Zum besseren Verständnis dieser Spezifikation, werden einige der nachfolgend häufig verwendeten Begriffe näher erläutert.

- (a.1) *Telemetriedaten*: Die zentrale Entität von *AaaS* sind Telemetriedaten, die in *Log-Nachrichten* und *Metriken* unterteilt werden können. Beide Kategorien haben gemeinsam, dass mit jedem Datensatz ein *Zeitstempel*, ein *Name* und die Identifikation des Senders (*Client-ID*) verbunden ist. Zusammengehörige Telemetriedaten können über einen gemeinsamen Namen angesprochen werden, z. B. „QueueLength“ oder „RequestCounter“. Der Zeitstempel wird clientseitig erstellt und an das Backend gesendet.
- (a.2) *Log-Nachrichten*: Über Log-Nachrichten können externe Anwendungen textuelle Informationen über den Systemzustand an das Backend übermitteln. Jeder Log-Nachricht muss die Client-Anwendung eine von mindestens drei Kategorien zuordnen: Trace, Warning, Error.
- (a.3) *Metriken*: Mittels Metriken können numerische Werte repräsentiert werden. Man muss zwischen zwei Typen von Metriken unterscheiden: Zähler und Messungen. Zähler werden im Backend inkrementiert, bei Messungen gibt der Client den aktuellen Wert mit.  
 Zweierteams müssen einen weiteren Metriktyp zur Verfügung stellen, mit dem Ergebnisse von Zeitmessung, also Zeitintervalle, repräsentiert werden können.
- (a.4) *Client*: Ein Client ist eine Anwendung, welche Telemetriedaten produziert und an *AaaS* sendet. Da Clients kein direkter Bestandteil von *AaaS* sind, werden sie an manchen Stellen auch als externe Anwendungen bezeichnet.
- (a.5) *App-Key*: Damit ein Client berechtigt ist, *AaaS* zu nutzen, muss er bei jedem Request einen im System hinterlegten App-Key mitsenden. Jedem Client ist ein eindeutiger App-Key zugeordnet, den sich alle Instanzen eines Clients teilen. Auf welche Weise dies geschieht, ist ein Implementierungsdetail, das den Entwicklungsteams überlassen wird.
- (a.6) *Client-ID*: Jeder Instanz eines Clients muss ein eindeutiger Schlüssel zugeordnet sein. Die Erzeugung bzw. Verwaltung dieses Schlüssels ist Aufgabe der Client-Anwendungen.

### (b) AaaS.Core

*AaaS.Core* ist die zentrale Stelle im System, in der die Daten von den verbundenen externen Anwendungen verarbeitet, gespeichert und für die Visualisierungskomponente aufbereitet werden. *AaaS-Core* enthält mehrere Detektoren, welche die Telemetriedaten nach verschiedenen Kriterien analysieren. Werden Anomalien in Daten entdeckt, sorgen Aktionen dafür, dass Warnungen an das Servicepersonal versendet werden.

- (b.1) *Detektoren*: Detektoren haben die Aufgabe, „auffällige“ Werte in den gesammelten Telemetriedaten zu identifizieren. Detektoren werden wiederholt in regelmäßigen Abständen ausgeführt. Dieses Zeitintervall kann für jeden Detektor unterschiedlich sein und muss über den Web-Client konfigurierbar sein. Es soll einfach möglich sein, Detektoren auszutauschen bzw. hinzuzufügen.  
 Es müssen zumindest folgende Detektortypen implementiert werden. Ein *Min/Max-Detektor* überprüft, ob sich die Werte einer Metrik in einem vorgegebenen konfigurierbaren Intervall [min, max] befinden. Überschreitet die Anzahl der Ausreißer innerhalb einer Zeitspanne einen gewissen Schwellwert, wird eine Aktion getriggert. Der *Schiebefenster-Detektor* betrachtet ein Zeitintervall und überprüft, ob der aktuelle Wert, die Summe oder der Durchschnitt der Werte

größer oder kleiner als ein vorgegebener Schwellwert ist. Die Größe des Zeitintervalls, die Aggregationoperation (aktueller Wert, Summe, Durchschnitt), die Vergleichsoperation (kleiner oder größer) und der Schwellwert müssen konfigurierbar sein.

Von jedem Detektortyp kann es mehrere Instanzen geben, die über deren Namen einer Metrik zugeordnet sind.

- ✓ Sind die Detektoren lose an die Anwendung gekoppelt und so einfach austauschbar?
- ✓ Gibt es automatisierte Tests für die Logik, die einfach (ohne Datenbank) ausführbar sind?

(b.2) *Aktionen*: Detektoren triggern Aktionen, wenn sie Ausreißer erkennen. Mittels Aktionen werden Warnmeldungen generiert, die über verschiedene Kanäle an das Servicepersonal weitergegeben werden. Beispielsweise könnten E-Mails bzw. SMS versendet, eine Warnlampe eingeschaltet oder ein Eintrag in einer Datenbank erstellt werden. Einem Detektor ist genau eine Aktion zugeordnet. Auch bei Aktionen ist auf darauf zu achten, dass diese einfach austauschbar sind.

Es ist zumindest ein Aktionstyp zu realisieren, der einen Web-Hook triggert. Auf diese Weise können Workflow-Automation-Tools wie Zapier oder IFTTT in *AaaS* eingebunden werden, welche die weitere Verarbeitung der Warnmeldungen übernehmen. Die Integration dieser Tools ist aber nicht Teil dieses Projekts. Allerdings ist eine Möglichkeit vorzusehen, die Web-Hooks zu testen (z. B. mittels Request Bin).

👤 Zweierteams müssen eine weitere Aktion implementieren, mit der E-Mails an eine konfigurierbare Adresse versenden kann. Die E-Mail enthält detaillierte Informationen zur entdeckten Anomalie.

- ✓ Achten Sie auf eine saubere Trennung von Detektoren und Aktionen, sodass diese beliebig kombiniert werden können.

(b.3) *Autorisierung*: Für die eingehenden Nachrichten ist zu überprüfen, ob Sie einen gültigen App-Key enthalten. Ist dies nicht der Fall, ist die Nachricht zu verwerfen und ein entsprechender Statuscode an den Sender zurückzuschicken.

(b.4) *Speichern von Log-Nachrichten*: Nach der Autorisierung der eingehenden Nachrichten werden sie mit dem mitgelieferten Zeitstempel in der Datenbank persistiert.

- ✓ Der Zugriff auf die Datenbank ist von *AaaS-Core* entkoppelt und erfolgt über eigenständige Komponenten.

(b.5) *Abfragen*: Eine wesentliche Aufgabe von *AaaS-Web* ist die Visualisierung der erfassten Nachrichten und Metriken. Über Abfragen erhält diese Systemkomponente Zugang zu den erforderlichen Daten. Welche Daten benötigt werden und in welcher Weise diese aufbereitet werden müssen, ist aus den Anforderungen an *AaaS-Web* abzuleiten.

(b.6) 👤 *Heartbeats*: Clients senden in regelmäßigen Abständen Nachrichten, die bestätigen, dass der Client noch existiert bzw. dass zum Client eine Verbindung besteht. Fallen mehrere Heartbeats aus, ohne dass sich der Client vorher abgemeldet hat, ist einmalig eine entsprechende Warnmeldung zu generieren.

## (c) AaaS.ClientSDK

Clients kommunizieren über einen REST-Endpunkt mit dem Backend. Um die Einbindung von Clients möglichst einfach zu gestalten, wird der für die Kommunikation mit dem Backend erforderliche Infrastrukturcode zu einer Bibliothek zusammengefasst und als NuGet-Paket zur Verfügung gestellt. 👤 Die Bibliothek muss eine komfortable Möglichkeit bieten, Zeitmessungen durchzuführen und die Ergebnisse an das Backend zu senden.

Zum Testen von Backends ist ein Client zu implementieren, der mithilfe von *AaaS.ClientSDK* Telemetriedaten generiert und an das Backend sendet. Dieser Client kann sehr einfach gehalten werden, sollte aber folgende Mindestanforderungen erfüllen:

- Es werden Log-Nachrichten, Metrikdaten und Zeitmessungen (👤) generiert.
- Die Frequenz, in welcher Telemetriedaten erzeugt werden, ist steuerbar.

Beispielsweise könnte man im Client auf Events wie Mausbewegungen oder Tastatureingaben reagieren und daraus Telemetriedaten ableiten. Die konkrete Funktionalität und auch die UI-Technologie (Konsolen, Web-, Desktop-Anwendung etc.) ist völlig Ihnen überlassen. Der Fokus dieser Clients liegt darauf, die Funktionsweise des Gesamtsystems zu testen und zu demonstrieren. Die Qualität der softwaretechnischen Umsetzung des Testclients ist sekundär.

- ✓ Bietet das SDK eine saubere und möglichst einfache API?
- ✓ Ist es durch die Interaktion mit dem Client möglich, die Funktionsweise von Features wie der Anomalieerkennung zu demonstrieren?

#### (d) AaaS.Api

Die für *AaaS.Web* erforderliche Funktionalität ist in Form eines REST-basierten Web-Service zu exportieren. Externe Anwendungen werden ebenfalls über eine REST-Schnittstelle an *AaaS-Core* angebunden und können über diesem Weg Telemetriedaten an das Backend übermitteln.

Die Web-Service-Schnittstellen werden maschinenlesbar als OpenAPI-Dokument ausgeliefert. Client-Entwickler erhalten eine übersichtliche Dokumentation in Form von HTML-Seiten.

- ✓ API-Dokumentation, Uniform Interface, schmale API-Schicht (keine Geschäftslogik im Web-Service)
- ✓ Sind die beiden APIs für den Web-Client und die externen Anwendungen sauber voneinander getrennt und als zwei unabhängige Controller implementiert?

#### (e) AaaS.Web

Der Web-Client muss die nachfolgend näher ausgeführten Anforderungen erfüllen:

- (e.1) *Authentifizierung mit OAuth2*. Benutzer müssen sich für die Nutzung des Clients anmelden. Verwenden Sie hier bspw. den in der Übung eingesetzten Identity-Server, der von Manfred Steyer in der Azure-Cloud gehostet wird, oder Sie können auch andere Dienste wie Auth0 oder Open-Source-Lösungen wie z.B. Keycloak dafür nutzen.

- ✓ Guards, OAuth2

- (e.2) *Verwaltung der Detektoren*. In einer eigenen Administrationsseite können Detektoren verwaltet und konfiguriert werden. Es sollte zumindest möglich sein, vorhandene Detektoren ein-/auszuschalten, bzw. diese zu konfigurieren (bspw. *Min/Max*, *Sliding Window*).

- ✓ Formulare, Validierung, Kommunikation mit dem Backend (Services), Observables

- (e.3) *Visualisierung von Log-Nachrichten*. Stellen Sie die Telemetriedaten vom Typ „Log-Nachricht“ in Form einer übersichtlichen Liste dar, welche auch durchsucht werden kann. Sie können sich hier z.B. von den Produkten rund um <https://www.elastic.co/de/elasticsearch/> inspirieren lassen. Überlegen Sie sich vor allem ein Konzept um die Anzeige übersichtlich zu halten.

- ✓ Formulare, Kommunikation mit dem Backend (Services), Observables, RxJS debounceTime, eigene Komponente für die Suche, Pipes

(e.4) *Statisch konfigurierte Visualisierung von Metriken*. Metriken können visuell ansprechender z. B. in Form von Charts dargestellt werden. Versuchen Sie dazu geeignete Bibliotheken einzubinden. Unter einer statisch konfigurierten Visualisierung verstehen wir, dass die Charts/Darstellungen schon fix hinterlegt sind (also „hard coded“ bspw. für die CPU-Last).

✓ Kommunikation mit dem Backend (Services), Observables, Bibliotheken (Module)

(e.5) *Dynamisch konfigurierte Visualisierung von Metriken (nur für ein Sehr Gut notwendig!)*. Darunter verstehen wir, dass erst zur Laufzeit ausgelesen wird, welche Metriken im Backend verfügbar sind. Es kann also dynamisch konfiguriert werden, wie das Chart diese Metriken anzeigt (Linie, Balken, etc.) und ob mehrere Charts angezeigt werden etc.

✓ Kommunikation mit dem Backend (Services), Observables, Bibliotheken (Module), Lösungsansatz

## (f) 👤 Allgemeine Anforderungen für Zweierteams

Überlegen Sie sich geeignete Maßnahmen, die eine professionelle Anwendungsentwicklung in einem Team unterstützen, setzen Sie diese um und nehmen Sie diese auch in die Dokumentation auf. Dazu zählen beispielsweise der Einsatz eines Issue-Tracking-Systems (z. B. jenes von GitHub), automatisierte Builds und Vorkehrungen, die eine einfache Installation und Inbetriebnahme Ihres Softwaresystems ermöglichen.

*Hinweis:* GitHub Classroom stellt nur sehr eingeschränkt Ressourcen für GitHub Actions zur Verfügung, die sich alle Mitglieder eines Classrooms, also alle Studierenden, teilen müssen.

## Ergebnisse

Die in der Anforderungsdefinition festgelegten Funktionen sind in Einer- oder Zweierteams zu realisieren. Anforderungen, die mit dem Symbol 👤 gekennzeichnet sind, müssen nur von Zweierteams ausgearbeitet werden. Alle anderen Anforderungen gelten für beide Gruppen. Die WEA5 zuzuordnende Funktionalität ist in Form von Einzelarbeiten umzusetzen.

Die Entwicklung der Projektarbeit ist auf GitHub durchzuführen. Alle Teams bekommen ein Repository zugewiesen, auf welches das Team und die Übungsleiter Zugriff haben.

Erstellen Sie für alle Dokumente und Quelltext-Dateien eine übersichtliche Verzeichnisstruktur und packen Sie diese in eine ZIP-Datei. Dieses Archiv stellen Sie Ihrem Übungsleiter auf der E-Learning-Plattform in den Kursen zur SWK5-Übung bzw. zu WEA5 zur Verfügung. Achten Sie darauf, dass Sie nur die relevanten (keine generierten) Dateien in das Archiv geben. Große Archive mit unnötigen Dateien vergeuden Speicherplatz und können daher zu Punkteabzügen führen.

Große Binärdateien, wie Komponenten von Drittherstellern oder Datenbanken dürfen keinesfalls im Repository abgelegt werden. Checken Sie anstatt dessen Scripte zur Erzeugung des Datenbankschemas und zum Befüllen der Datenbank ein. Dokumentieren Sie, wie Ihr System in Betrieb genommen werden kann.

Konzentrieren Sie sich bei der Dokumentation dieser Projektarbeit auf die Darstellung der Architektur und des Designs Ihrer Anwendung. Legen Sie die Dokumentation so an, dass Personen, die Ihr Projekt weiterführen möchten, die Struktur Ihres Softwaresystems schnell erfassen und Designentscheidungen nachvollziehen können. Durch das unreflektierte Einfügen von generierten Diagrammen wird diese Anforderung nicht erfüllt. Quelltexte sollen nur dann in die Dokumentation aufzunehmen, wenn Sie damit zentrale Aspekte Ihrer Implementierung erläutern wollen. Die Dokumente sind ausschließlich in Form von PDF-Dateien abzugeben.