# Search Engine für TV

21.02.2024 – 07.03.2024

**Kandidat**
Parker Peter
**Betrieb (=Durchführungsort)**
Sunrise GmbH
Thurgauerstrasse 101b, PLZ 8152
T 076 555 55 55 (am besten erreichbar)
G +41587775348
M peter.parker@pk19.ch

**BerufsbildnerIn/ Lehrfirma**
Stammbach Roman
Sunrise GmbH
Thurgauerstrasse 101, 8152 / Glattbrugg
T 0767775348 (am besten erreichbar)
G 0767775348
M roman.stammbach@sunrise.net

**Verantwortliche Fachkraft**
Stammbach Roman
Sunrise Schweiz
Thurgauerstrasse 61, 8152 / Opfikon
T +41 765717198 (am besten erreichbar)
G 0767775348
M roman.stammbach@pk19.ch

**Hauptexperte**

T (am besten erreichbar)
G
M

**Arbeitsbereiche**

- 
- 
- 
- 
- 
- 
- 
- 
- 

# Search Engine für TV

21.02.2024 – 07.03.2024

## Ausgangslage

Der aktuell eingesetzte Search-Service ist bereits mehrere Jahre alt und muss aus diversen Gründen erneuert werden. Es wurde intern im Team evaluiert, ob die verwendete Technologie "MongoDB Atlas Search" noch weiterhin die Richtige ist, oder ob mit OpenSearch eine bessere, moderne Lösung existiert. Der Kandidat und das Team sind sich weitgehend einig, dass OpenSearch eine bessere Alternative bietet. So wird der Kandidat während der IPA diese Technologie verwenden, um die Implementation des Search-Services umzusetzen.
Der Search-Service soll als Microservice erstellt werden und Daten über einen Rest-Endpoint liefern.

## Detaillierte Aufgabenstellung

Im Rahmen der IPA soll eine neuer Search-Service entstehen. Die Implementation wird mit Python mit dem Framework FastAPI geschehen und soll in unsere Infrastruktur als Dockerimage über Kubernetes integriert werden. Es wird mit der Searchengiine "OpenSearch" gearbeitet. Im Verwenden von weiteren Python-Libraries ist der Kandidat weitgehend frei.
Während des Projekts wird stets mit Git und deskriptiven Commit-Messages gearbeitet. Während den Vorarbeiten wird ein Repository auf Bitbucket erstellt, in welchem während der IPA gearbeitet werden kann. Als CI/CD kommt AWS CodePipeline zum Einsatz. Dies ist auch bereits während den Vorarbeiten aufzusetzen.
Der Kandidat hält sich an interne Scrum-Vorgaben und wird uns bei Daily Meetings auf dem aktuellen Stand seiner Arbeiten halten.
Das Erstellen des Projektrahmens, Dateistruktur und leere Python/FastAPI Applikation läuft über ein internes Template bereits während der IPA-Vorbereitung.
Die Implementation soll verhältnismässig getestet werden.

Die Searchengine soll folgende Funktionen enthalten:
* Indexierung der EPG (Electronic Program Guide)-Einträge
* Suche nach EPG-Einträgen
* Filter nach vergangenen, zukünftigen und allen Einträgen

# Hauptfunktionen (MVP)
## Indexierung von OpenSearch
* Job für die Indexierung der Channel-Data
- Die Channel-Data soll über eine bereits bestehende API abgefragt werden. Somit keinen direkten Zugriff auf die Datenbank
- Im Rahmen der IPA gilt dies als erfüllt und ausreichend, wenn die Abfrage der Daten einmal täglich geschieht.
* Job für die Indexierung der EPG (Electronic Program Guide)-Data
- Die EPG-Data soll über eine bereits bestehende API abgefragt werden. Somit keinen direkten Zugriff auf die Datenbank.
- Im Rahmen der IPA gilt dies als erfüllt und ausreichend, wenn die Abfrage der Daten einmal

# Search Engine für TV

21.02.2024 – 07.03.2024

täglich geschieht.

* Job für die Indexierung der Aufnahmen

- Indexierung direkt nach Erstellung der Aufnahme

- Wenn die Aufnahme gelöscht wird, soll diese auch aus dem Index gelöscht werden und nicht mehr auffindbar sein (Nice to have, wird nicht gewertet)

Die Indexierung wird dank OpenSearch-API möglich und soll damit implementiert werden.

## Implementation der Such-Logik

Wenn ein Benutzer ein Such-Term eingibt, soll ein Asset nach Titel und Subtitel gesucht werden.

* Sobald ein Benutzer eine Aufnahme erstellt, soll er diese mit einem Keyword finden können (eine kleine Verspätung von ein paar Sekunden ist möglich)

* EPG-Data und Channels sollen gefunden werden

- Diese sollten immer verfügbar sein, durch den Daily Job.

* Beispiel:

- Wenn ein Benutzer eine Big Bang Theory-Folge sucht, sollte er Big Bang Theory finden können via Titel, oder Episodentitel

* Die Suche sollte einen gewissen Spielraum für Rechtschreibfehler bieten. Zum Beispiel sollte ein Prompt wie “Big bang Teory” auch die richtigen Resultate liefern.

## Implementation des API-Endpoints

Der API-Endpoint ist als REST-Service verfügbar und gibt auf ein Keyword die wichtigen Daten via JSON zurück.

* Es werden nur die nötigen Felder für eine Suche zurückgegeben

* Die Antwort soll einen Diskriminator beinhalten, welcher einen spezifischen Typ von Response kategorisieren kann (Recordings/EPG-Data/Channels)

Grundsätzlich stehen diese primären Hauptfunktionen (MVP) im Vordergrund. Weitere Funktionen werden nach der IPA vom Team kontinuierlich in den neuen Service übertragen.

## Mittel und Methoden

Der Kandidat arbeitet mit seinem Apple MacBook und in der IDE seiner Wahl. Die Entwicklung läuft lokal ab, der Kandidat erstellt nach eigenem Ermessen Code-Commits und pusht diese regelmässig in das Bitbucket-Repository. Der Search-Service wird in der Programmiersprache Python mit dem Framework FastAPI implementiert. Die Datenbank läuft bereits auf MongoDB. CI/CD läuft über AWS CodePipeline.

Die verschiedenen Tasks werden mittels Scrum-Prozess organisiert und werden für den Kandidaten in Jira bereitgestellt. Er wird nach diesen Tasks arbeiten.

# Search Engine für TV

21.02.2024 – 07.03.2024

Welche Projektmanagement-Methode verwendet wird, um sich selbst und die Dokumentation zu organisieren, wird nicht vorgegeben. Hier ist der Kandidat frei und kann zum Beispiel mittels Wasserfall- oder IPERKA-Methode vorgehen.

Für die Arbeit werden die internen Code-Guidelines angewendet.

## Vorkenntnisse

Der Kandidat kennt die Technologien bereits sehr gut. Er hat gute Vorkenntnisse in Python und FastAPI und kennt sich mit Pydantic aus.

In der Thematik "Search-Engine" hat der Kandidat noch wenig Erfahrung und wird hier auf Neues treffen. Besonders OpenSearch ist für den Kandidaten und für das gesamte Team neu.

## Vorarbeiten

Der Kandidat wird die grundlegenden Rahmenarbeiten des Search-Services bereits in der IPA-Vorbereitung erledigen. Dazu gehört die Erstellung des Projektrahmens, das Einrichten des Git-Repositorys, Boilerplate FastAPI Applikation, CI/CD-Konfiguration.

## Neue Lerninhalte

OpenSearch, bzw. MongoDB Atlas Search ist neu für den Kandidaten, er wird während den Vorarbeiten bereits Skills für die beiden Infrastrukturen erarbeiten.

## Arbeiten in den letzten 6 Monaten

Der Kandidat konnte in den letzten 6 Monaten bei verschiedenen Python-Microservices für das TV-Backend arbeiten und hat schon viele Fähigkeiten erlangt mit dem Einsetzen von FastAPI.

# Individuelle Kriterien

Auf den folgenden Seiten werden die individuellen Kriterien aufgeführt, welche durch die verantwortliche Fachkraft für diese IPA festgelegt wurden.

# Individuelle Kriterien

| **Leitfrage**<br>1 | **Implementation der Search Engine**<br><br>Es werden die wichtigen Hauptfunktionen implementiert (10 Teilanforderungen, aus der Aufgabenstellung):<br>* Indexierung von OpenSearch<br>* Implementation der Such-Logik<br>* Implementation des API-Endpoints |
|---|---|
| Gütestufe 3 | Es werden 9 - 10 Anforderungen erfüllt |
| Gütestufe 2 | Es werden 6 - 8 Anforderungen erfüllt |
| Gütestufe 1 | Es werden 3 - 5 Anforderungen erfüllt |
| Gütestufe 0 | Es werden < 3 Anforderungen erfüllt |

Notizen

# Individuelle Kriterien

| **Leitfrage**<br>2 | **Testabdeckung**<br><br>Die Implementation wird ausreichend und sinnvoll getestet. |
|---|---|
| Gütestufe 3 | Die Lösung wird umfassend und sinnvoll getestet. Testfälle und Szenarien sind klar definiert. Abdeckungslücken, falls vorhanden, werden identifiziert und dokumentiert. |
| Gütestufe 2 | Einige Funktionalitäten werden weniger detailliert getestet. Mögliche Lücken in der Testabdeckung sind identifiziert, aber die Bewertung ist nicht vollständig. |
| Gütestufe 1 | Testabdeckung ist oberflächlich, wichtige Funktionalitäten der implementierten Lösung wurden nicht ausreichend getestet. |
| Gütestufe 0 | Die Testabdeckung ist unvollständig. |

Notizen

# Individuelle Kriterien

| **Leitfrage**<br>3 | **Code-Qualität**<br><br>Der Code ist verständlich strukturiert. Es werden sinnvolle Namen für Funktionen/Variablen verwendet. Kommentare sind vorhanden, wo sinnvoll. |
|---|---|
| Gütestufe 3 | Die Code-Struktur ist konsistent und lesbar im Einklang mit internen Code-Standards. Werkzeuge wie flake8 und/oder black werden verwendet zur Identifizierung von Code-Stilproblemen. Klare und aussagekräftige Kommentare, insbesondere bei komplexen Abschnitten oder Algorithmen. |
| Gütestufe 2 | Einige Teile des Codes entsprechen den internen Code-Standards, während andere möglicherweise nicht einheitlich formatiert sind. Einige Code-Stilprobleme werden durch flake8 und/oder black identifiziert. Kommentare sind nicht sehr aussagekräftig oder nicht sinnvoll eingesetzt. |
| Gütestufe 1 | Wenige Teile des Codes entsprechen den internen Code-Standards. Es werden viele Stilprobleme durch flake8 oder und/oder black identifiziert. Kommentare sind wenig/nicht vorhanden, wo sinnvoll. |
| Gütestufe 0 | Inkonsistente Code-Struktur ohne Beachtung interner Code-Standards. Wesentliche Code-Stilprobleme wurden von flake8 und/oder black identifiziert. Fehlende oder unklare Kommentare, die die Code-Abschnitte nicht ausreichend erläutern. |

Notizen

# Individuelle Kriterien

# Individuelle Kriterien

| **Leitfrage**<br>4 | **Architektur**<br><br>Skalierbarkeit, Architektur der Implementation. |
|---|---|
| Gütestufe 3 | Die Implementierung ermöglicht eine Skalierbarkeit durch effiziente Nutzung von Ressourcen. Architektonische Entscheidungen unterstützen eine einfache Integration neuer Funktionen ohne umfangreiche Code-Änderungen. |
| Gütestufe 2 | Skalierbarkeitsaspekte sind teilweise berücksichtigt, aber es gibt Bereiche, die noch nicht optimal auf zukünftiges Wachstum vorbereitet sind. Einige Teile des Codes könnten besser modularisiert werden, um die einfache Hinzufügung neuer Funktionen zu erleichtern. |
| Gütestufe 1 | Die Implementierung ist nicht gut auf Skalierbarkeit vorbereitet und könnte Schwierigkeiten bei der Bewältigung steigender Anforderungen haben. |
| Gütestufe 0 | Die Implementierung ist nicht auf Skalierbarkeit vorbereitet. Architektonische Entscheidungen behindern oder verhindern die einfache Integration neuer Funktionen. |

Notizen

# Individuelle Kriterien

| **Leitfrage**<br>5 | **API-Dokumentation**<br><br>Die API wird durch Einsatz von Pydantic und Swagger automatisch dokumentiert. |
|---|---|
| Gütestufe 3 | Pydantic Models werden konsequent für die Modellierung von Datenobjekten und Validierung verwendet. Gut strukturierte und präzise OpenAPI (Swagger) Dokumentation, die automatisch aus den Pydantic Models generiert werden. Klar definierte Endpunkte und Parameter in der Dokumentation, die die Verwendung der API für Entwickler erleichtern. |
| Gütestufe 2 | Pydantic Models werden teilweise verwendet, einige Datenobjekte könnten jedoch durchgängiger modelliert sein. Die OpenAPI Dokumentation ist teilweise präzise, es gibt jedoch einige Bereiche, die verbessert werden könnten, um Entwicklern eine klare Orientierung zu bieten. |
| Gütestufe 1 | Pydantic Models werden unzureichend verwendet, was zu einer geringeren Datenvalidierung und Strukturierung führt. Die OpenAPI Dokumentation ist unklar oder nicht gut strukturiert, was die Verwendung der API für Entwickler erschwert. |
| Gütestufe 0 | Es werden keine Pydantic Models verwendet, die Daten werden nicht validiert. Die OpenAPI-Dokumentation ist unvollständig. |

Notizen

# Individuelle Kriterien

# Individuelle Kriterien

| **Leitfrage**<br>6 | **Git-Workflow**<br><br>Git wird gekonnt eingesetzt. |
|---|---|
| Gütestufe 3 | Commits erfolgen nach eigenem Ermessen, wenn sinnvoll. Die Commithistory ist klar strukturiert, Entstehung des Produkts ist nachvollziehbar. Es wird mit Branches gearbeitet, wenn sinnvoll. |
| Gütestufe 2 | Commits erfolgen sporadisch. Branches werden nicht sinnvoll eingesetzt. Die Git-Struktur ist weniger klar. |
| Gütestufe 1 | Commits erfolgen selten oder werden nicht sinnvoll eingesetzt. |
| Gütestufe 0 | Commits erfolgen entweder gar nicht oder nur sehr selten. Es ist keine klare Git-Struktur vorhanden. |

Notizen

# Individuelle Kriterien

| **Leitfrage**<br>7 | **Fehlerbehandlung**<br><br>Behandlung von Fehlern (Error-/Exception-Handling). Geworfene Exceptions werden in der Applikation behandelt und eine entsprechende klare Antwort mit entsprechendem HTTP Code zum Client geliefert. Eigene Exceptions werden genutzt um Eingaben zu validieren und erwartete Fehler zu behandeln. |
|---|---|
| Gütestufe 3 | Fehler werden behandelt, soweit möglich und sinnvoll. Das Error-Handling ist weitgehend abdeckend und wird Python-Idiomatisch eingesetzt (keine Catch-All-Statements). Requests werden mithilfe von FastAPI/Pydantic validiert. Die HTTP-Statuscodes werden sinnvoll gewählt. |
| Gütestufe 2 | Es treten teilweise Fehler auf, welche nicht behandelt wurden, Requests werden nicht ausreichend validiert. |
| Gütestufe 1 | Error-Handling wird nicht sinnvoll/nicht abdeckend angewendet. Das Error-Handling ist nicht Python-Idiomatisch umgesetzt. |
| Gütestufe 0 | Es treten viele Fehler auf, welche nicht behandelt wurden.<br>Requests werden nicht validiert. |

Notizen