Projektdokumentation

Auto Daten List Informationen System (ADLIS)

Abschlussarbeit Fakultät 73

3.2.2021

Frömling, Chris
Körner, Rene
VOLKSWAGEN AG

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

ADLIS - Auto Daten List Informationen System

CORS – Cross Origin Resource Sharing

Db – Database

http – Hypertext Transfer Protocol

Java-SE – Java Standard Edition

REST-API - Representational State Transfer - Application Programming Interface

YAML – Yet Another Multicolumn Layout

AUTO DATEN LIST INFORMATIONEN SYSTEM (ADLIS)

1. EINLEITUNG

Der Fortschritt globaler technischer Errungenschaften entwickelt sich seit Mitte des 19. Jahrhunderts in immer schnellerem Ausmaß. Von der Entdeckung der Glühbirne, über die Entwicklung des Flugzeugs bis hin zu den Hochleistungsrechnern in unseren Hosentaschen. Auch die Autoindustrie unterlag seit der Geburtsstunde durch den „Benz Patent-Motorwagen Nummer 1“ einem regen und beeindruckendem Wandel. „Ob wir es mögen oder nicht, langsam, aber sicher übernehmen die Roboter den Job des Autofahrers.“[1]

1.1 PROJEKTDEFINITION

Das System ADLIS dient dazu automatisch generierte Daten eines Kraftfahrzeugs zu empfangen, zu speichern und bei Bedarf und Autorisierung entsprechend zu visualisieren. Diese Daten können sowohl von Privatpersonen, über die Registrierung mit der Fahrzeug-FIN, als auch Werkstätten, Herstellern etc. eingesehen, ausgewertet beziehungsweise für Analysezwecke verwendet werden.

Dieses Abschlussprojekt soll die Möglichkeit bieten Beispieldaten zu generieren, zu speichern, spezifisch auszugeben und entsprechende Accounts zu diesen anzulegen. Das Projekt wird von Grund auf programmiert.

[1] (Popular Science, 1958)

1.2 AUSGANGSLAGE

Zu Beginn des Projekts wurden den Programmierern entsprechende Ressourcen zur Verfügung gestellt.

Dazu gehören ein Git-Repository, ein Docker-Repository, sowie die virtuelle Umgebung von Cloudogu und einer Konfigurationsdatei zur Arbeit mit dem Kubernetes-Cluster. Unter diesen Voraussetzungen ist es möglich, mit Hilfe von Kubernetes und Docker, Microservices unabhängig voneinander aufzubauen, die die geforderte Aufgabenstellung erfüllen.

2. PROJEKT

Dies Projekt ist das Abschlussprojekt der Full-Stack-Development Vertiefung des 1. Jahrganges der Fakultät 73. Neben diversen Programmierkenntnissen, wird ebenso Wissen in der Erstellung von Microservices, dem Aufbau eines Clusters in einer Cloud-Umgebung, sowie im Umgang mit Sicherheitskonzepten, Datenbanken und nötigen API-Rest-Schnittstellen verlangt. Grundlage ist, dass heute Kraftfahrzeuge eine Vielzahl an Daten generieren. Um mit diesen arbeiten zu können und anhand solcher Daten Fehler, Verschleiß und Verbesserungsmöglichkeiten zu erkennen, müssen diese gespeichert und eingesehen werden können. Das zugrundeliegende Projekt stellt einen vereinfachten Aufbau dieses Systems nach, indem es Daten generiert, speichert und einem autorisierten Nutzer zur Verfügung stellt.

2.1 PROJEKTPLANUNG

Das Projektteam agierte in mehreren Planungsphasen. Für die erste wurde sich ein Überblick über die benötigten Microservices verschafft und diese umgesetzt.

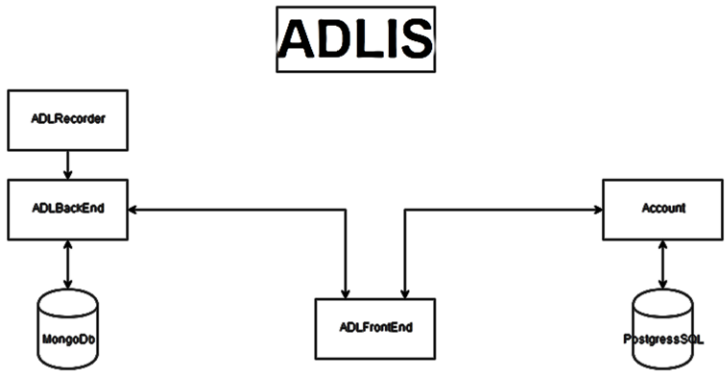

Abbildung 1 - Microservice-Struktur

Zu sehen ist ein Aufbau aus 6 unabhängig laufenden Microservices, welche zuerst im Backend-Bereich mit Java und dem Spring-Boot-Framework und anschließend im Frontend-Bereich mit Angular aufgesetzt wurden. Die Datenbankstrukturen von Postgresql, sowie Mongo wurden in Planungsphase 2 einbezogen.

Ab der Planungsphase 2 erfolgte die Umsetzung in der Cloud-Umgebung, wofür zunächst eine erweiterte Übersicht des Microservice-Aufbaus nötig war.

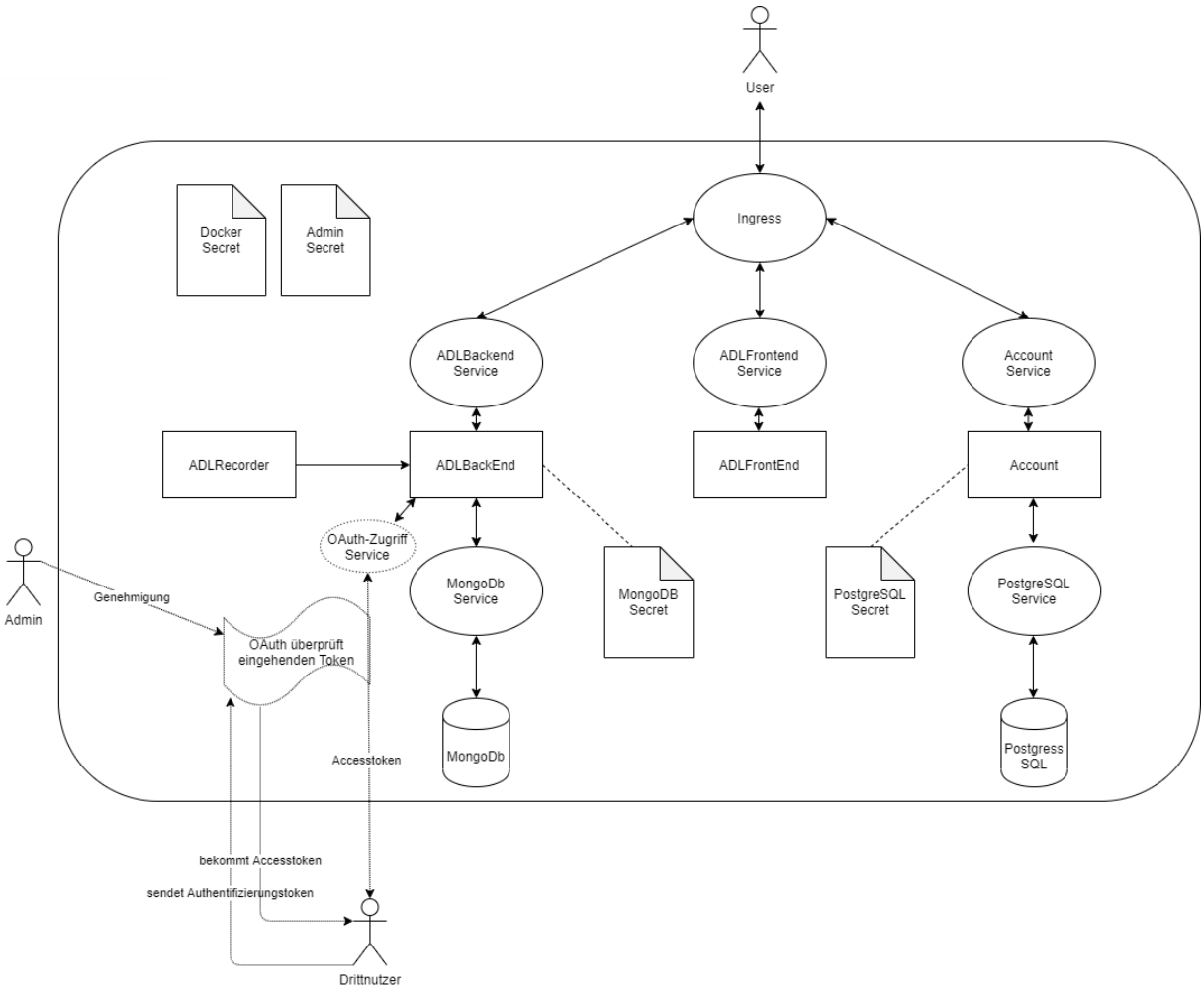

Abbildung 2 - Microservice-Architektur in der Cloud

Planungsphase 2 befasste sich mit dem Aufbau der grundlegenden Strukturen in der Cloud-Umgebung. Jede Anwendung musste containerisiert und mit Hilfe von Docker-Images und jeweils zugehörigen Services zur Kommunikation untereinander programmiert werden. Außerdem wurden die Datenbankstrukturen implementiert.

Zusätzlich wurden eine Authentifizierung und diverse Secrets benötigt, die empfindliche Daten enthalten und den entsprechenden Microservices übergeben werden. Dies wurde allerdings erst in Phase 3 umgesetzt.

Phase 3 war die zeitintensivste, da hier sowohl die Funktionstüchtigkeit der Programme, inklusive der REST-API-Schnittstellen als auch der Cloudumgebung umgesetzt wurde. Auch die Erstellung der Dokumentation fiel in diesen Zeitraum.

2.2 ZEITPLANUNG

Die folgende Tabelle enthält die Zeitplanung des vorliegenden Projektes. Dabei wurden je Phase entsprechende Analysen, Entwürfe und Implementierungen umgesetzt.

Die Fakten für die Dokumentation wurden während des gesamten Zeitraums zusammengetragen.

Phase 1	Planung und Erstellung der Grundkonstrukte der einzelnen Anwendungen ➔ Java + Spring-Boot x2 ➔ Angular x1 ➔ JavaSE x1	22.01.2021
Phase 2	Planung und Erstellung der Grundkonstruktion in der Cloud ➔ Docker ➔ Kubernetes ➔ MongoDB ➔ PostgreSQL	25.01.2021
Phase 3	Funktionsfähigkeit der Microservices herstellen Funktionsfähigkeit des Clusters herstellen	26.01.2021- 03.02.2021
	Dokumentation	01.02.2021- 03.02.2021

Abbildung 3 – Zeitplanung

3. DURCHFÜHRUNG

Die folgenden Punkte beschreiben die Herangehensweise und Umsetzung der geforderten Punkte, die das Projekt erfüllen soll. Dabei wird anhand der gegebenen Aufgaben-Priorisierung schrittweise vorgegangen.

3.1 ARCHITEKTUR-DESIGN

Abbildung 2 bildet das Gesamtkonzept der geforderten Architektur ab inklusive einer möglichen OAuth-Authentifizierung ab. Als in einzelne Pods containerisierte Microservices sollen die in Abbildung 4 erläuterten zur Verfügung gestellt werden:

MicroService	Funktion	API
ADLRecorder	generiert Beispieldaten	POST ans ADLBackend
ADLBackend	Speichert generierte Recorder-Daten und stellt diese zur Verfügung	POST vom ADLRecorder GET ans ADLFrontend * MongoDb
ADLFrontend	User Interface zum Registrieren, Anmelden und Abrufen der Daten	GET vom ADLBackend GET/POST ans Account
Account	Anlegen neuer Accounts Login Funktion	GET/POST vom ADLFrontend * PostgreSQL
Mongo-Database	Speicher für generierte ADLRecords	*ADLBackend
PostgreSQL-Database	Speicher für Accounts	*Account

Tabelle 1 - Microservice Anforderungen

In der Kubernetes Umgebung werden weiterhin diverse Services und Secrets benötigt, die eine Kommunikation der einzelnen Container untereinander bzw. extern ermöglichen.

Das Projektteam setzt für die Kommunikation innerhalb des Clusters für alle Programme NodePort-Services auf. Damit Das Frontend nach außen kommunizieren kann und zusätzlich ein Zugriff auf den Account und das ADLBackend möglich ist, wird vor das ADLFrontend, sowie die REST-API-Schnittstellen, die zur Kommunikation des ADLFrontends mit weiteren Services dienen, ein sogenannter Ingress-Service gesetzt, der als Schnittstelle dient und eine externe Kommunikation ermöglicht.

Die sicherheitsrelevanten Daten für die Mongo-Datenbank, die PostgreSQL-Datenbank, das Docker-Repository und die Basic-Authentifizierung werden in jeweils eigens dafür angelegte Secrets hinterlegt. Dadurch entstehen zentrale Zugriffspunkte, die gesichert die Daten für den Zugriff gewünschter Services bereitstellen.

3.2 DATENMODELL

In diesem Projekt werden lediglich 2 Arten von Objekten/Entitäten angelegt, um die Funktionsweise zu gewährleisten. Diese sind wie folgt aufgebaut:

ADLRecord	Type	Account	Type
{id} fin	String	{id} fin	String
gps	String	password	String
electromotiveBeltTensionings	int		
vehicleIlluminationHours	int		
km	int		
fuelPercentage	int		
tirePressure	double		
refrigerantPercentage	int		
brakefluidPercentage	int		
screenWashPercentage	int		
kmHighway	int		
kmRoad	int		
kmCity	int		
temperatureCelsius	int		
elektricSeatAdjustments	int		
cdSwap	int		
chargeCycles	int		
creationDateString	String		

Tabelle 2 – Datenmodelle

3.3 IMPLEMENTIERUNG

Die ersten beiden Phasen dienten dem Team dazu die Grundplanung durchzugehen und entsprechend nötige Architekturen in den Grundstrukturen aufzubauen.

Der ADLRecorder sollte als JAVA-SE Programm ohne zusätzliche Frameworks aufgesetzt werden. Nach Absprache mit dem Trainer wurde sich auf eine äußerst schlanke Spring-Anwendung geeinigt.

Die beiden Microservices bezüglich des ALDBackends und des Accounts wurden ebenso um das Spring-Framework erweitert und mit Hilfe von Spring Initializr aufgesetzt. Zum Testen wurde bei beiden eine H2-Datenbank integriert. Das ADLBackend sollte im Laufe des Projektes an eine Mongo-Datenbank (für eingehende ADLRecord-Daten) und der Account an eine PostgreSQL-Datenbank (für angelegte Accounts) angebunden werden. Für das Frontend als User Interface wurde das Angular Framework als Routing-Module eingerichtet.

In Phase 2 wurde in der Cloudogu Umgebung das Grundgerüst hochgezogen. Aus den Grundservices Docker Images gebaut und diese über .yaml Dateien in Pods gestartet und entsprechende NodePort-Services (zum Testen anfangs als LoadBalancer) angehängt. Für den Zugriff auf das Docker-Repository wurde ein Secret mit entsprechenden Daten in der Kommandozeile kreiert. Weitere Secrets für die Datenbanken wurden über .yaml Dateien aufgesetzt und an den nötigen Stellen eingebunden. Weiterhin wurden mithilfe von Helm die Ressourcen für die Datenbanken gepullt, angepasst und installiert/ gestartet. Für die Basic-Authentifizierung entstand ein weiteres Secret.

Zu Beginn von Phase 3 wurden die Microservices um die gewünschten Dependencies, Funktionen erweitert und aufeinander abgestimmt. Erste Prioritäten lagen aufeinanderfolgend bei dem ADLBackend, dem ADLRecorder und dem ADLFrontend. Den einzelnen Pods wurden LoadBalancer-Services vorgesetzt, um durch den dadurch möglichen externen Zugriff mit Hilfe von Postman die REST-API-Schnittstellen zu testen und Swagger einzubinden.

Als HTTP-UserInterface wurde mit Angular zunächst die visuelle Ausgabe der Daten in einem Webfrontend umgesetzt und angepasst und später um gewünschte Funktionen wie die zeitliche Auswahl, Registrierung und Login erweitert. Die Besonderheit hier ist die Verwendung des VW eigenen Group-Ui für das Styling der Webpage.

Anschließend wurden die Programme für die Basic-Authentifizierung vorbereitet, SecurityConfiguration hinzugefügt, API-headers eingefügt und die empfindlichen Authentifizierungsdaten aus dem Secret über das Deployment an die Microservices

gegeben. Für das Angular Framework besteht diese Möglichkeit leider nicht, ist aber zentral umgesetzt, um auch hier eine pflegeleichte Wartbarkeit zu ermöglichen.

Damit die Services nicht einzeln von außen erreichbar sind, wurde ein Ingress-Service erstellt und die Pfade der anzusprechenden Services verknüpft als zentrale externe Zugriffstelle.

Nach den umgesetzten Prio1 und Prio2 Anforderungen wurden die Funktionalität des Account-Microservices und das Login- und Registrierungs-feature im Frontend inklusive Basic-Authentifizierung und postgreSQL-Datenbank hinzugefügt.

3.4 QUALITÄTSSICHERUNG/ TEST

Um eine besser Codequalität zu erreichen und mögliche Fehlerquellen zu verringern, arbeitete das Team ausschließlich in branches, welche erst bei voller Funktion in den master-Branch gemergt wurden. Weiterhin werden sämtliche Aktivitäten mit entsprechenden Hinweisen geloggt. Das ADLBackend wurde um einige Testklassen erweitert, um deren korrekte Funktionsweise darzustellen. Der Großteil der API-Tests erfolgte über Postman und die korrekte Funktionsweise des UserInterfaces beim Senden und Empfangen von Daten. Das Projektteam bevorzugt üblicherweise ein TestDrivenDevelopment, welches aufgrund der zeitlichen Umstände nicht umgesetzt wurde.

4. PROJEKTABSCHLUSS

Im Folgenden geht das Team auf Probleme ein, schneidet das Konzept der OAuth kurz an und stellt die Meinung im Fazit und Ausblick des Projektes dar.

4.1 PROBLEME

Im Laufe des Projekts gab es einige Schwierigkeiten mit CORS, also Cross-Origin zu bewältigen. Dies war besonders der Fall, als die Ressourcen der Umgebung ausgeschöpft waren, das Team den bereits aufgesetzten Ingress-Service nicht mehr

nutzen konnte und daher, um eine lauffähige Version garantieren zu können, das Projekt zwischenzeitlich auf eine LoadBalancer-Version umbaute. Zu einem späteren Zeitpunkt konnte wieder auf Ingress gewechselt werden.

Weitere einschränkende technische Probleme sind nicht aufgetreten.

4.2 SOLL/ IST VERGLEICH

Die Umsetzung der gegebenen Prioritäten gelang dem Projektteam wie folgt:

Alle geforderten PRIO 1 Aufgaben wurden gänzlich umgesetzt. Einzige Ausnahme bildet der ADLRecorder, welcher nach Absprache in einer äußerst schlanken SpringBoot-Variante erstellt wurde.

Alle geforderten PRIO 2 Aufgaben wurden gänzlich umgesetzt.

Alle geforderten PRIO 3 Aufgaben wurden gänzlich umgesetzt.

Ein Ingress Service dient als Schnittstelle zwischen den Services im Cluster und der externen Verfügbarkeit des und für das Frontend.

4.3 OAUTH KONZEPT

Bei dem Konzept der OAuth-Autentifizierung handelt es sich um ein „offenes Sicherheitsprotokoll für die Token basierte Autorisierung und Authentifizierung im Internet.“[2] Dadurch kann Dritten der Zugriff auf gewünschte Ressourcen ermöglicht werden ohne Nutzernamen und Passwort offenzulegen. Bei der OAuth2 Authentifizierung wird von 4 Genehmigungsprozessen gesprochen.[3]

Authorisierungscode:

- Client bittet Resource Owner um Login
- Resource Owner erstellt mit Autorisierungscode für Client Accesstoken

[2] https://www.security-insider.de/was-ist-oauth-a-712468/
[3] https://www.ionos.de/digitalguide/server/sicherheit/was-ist-oauth/

Implizite Autorisierung:

- Ähnlich dem Prinzip des Authorisierungscodes
- Authorization-Server erstellt Access-Token

Passwortfreigabe durch Resource Owner:

- Client erhält vom Resource Owner Userdaten

Client-Berechtigung:

- Client wird für die Nutzung gänzlich berechtigt

4.3 FAZIT

Rückblickend das Projekt ADLIS betrachtend, kommt das Projektteam zu dem Schluss, dass das Softwareergebnis den gewünschten Anforderungen und Funktionalitäten entspricht.

Die gestellte Entwicklungsumgebung und Repositories liefen mit wenigen Ausnahmen einwandfrei und ermöglichten eine gute Umsetzung der Prüfung. Basierend auf dem vorangegangenen Unterricht bezüglich des Prüfungsthemas sah sich das Team als ausreichend vorbereitet, um mit der Kubernetes Umgebung arbeiten zu können. Mit dieser Vorbereitung und eigenem Engegement war es gut möglich das Prüfungsthema umzusetzen. Das erlernte Wissen und die gesammelten Erfahrungen können praxisnah eingebunden und angewendet werden.

Sowohl die Anforderungen, als auch den zeitliche Rahmen empfand das Projektteam für angemessen und gut umsetzbar.

4.4 AUSBLICK

Diese ADLIS Projekt gibt einen guten, stark vereinfachten Überblick über die Zukunft der Automobilindustrie. Der jetzige ADLRecorder steht zukünftig für jedes einzelne Fahrzeug, welches eine Vielzahl an Daten direkt weiterleiten kann. Dies ermöglicht eine Reihe von Verbesserungen durch Auswertungen wie Verschleiß, Fehler oder auch

Fahrverhalten. Bei so einem Datenverkehr entstehen selbstverständlich auch wieder viele Gefahren des Datenklaus und -missbrauchs, die es zu verhindern gilt. Das Team sieht bei der korrekten Erfassung und Sicherung des Datenverkehrs ein enormes Potential für Hersteller und Verbraucher, die Nachhaltigkeit und Sicherheit im Straßenverkehr zu steigern.

GLOSSAR

B

branches
: Verzweigungen, um parallel an unterschiedlichen Versionen im git zu arbeiten · k

C

Cloud
: Zusammenschluss von mehreren Servern · d, e

Cloudogu
: Toolchain für Softwareentwickler, die alle Phasen des Lebenszyklus der Softwareentwicklung abdeckt · d

containerisierte
: enthält Anwendung und zur Laufzeit benötigte Ressourcen · h

CORS
: Mechanismus,der Webanwendungen ermöglicht Ressourcen nicht nur vom UrsprungsServer zu laden · l

D

Dependencies
: Pakete externer Bibliotheken oder Software · j

Docker
: freie Software zur Isolierung von Anwendungen mit Hilfe von Containervirtualisierung · d

F

Framework
: Programmiergerüst · e

G

gemergt
: Zusammenfügen des Codes unterschiedlicher Verzweigungen eines Git-Repositories · k

Git
: Open-Source Tool zur verteilten Versionskontrolle von Software · d

Group-Ui
: konzernweite Design- und Frontendbibliothek · k

H

H2-Datenbank
: Datenbank-Management-System · j

Helm
: Package-Manager für Kubernetes · j

HTTP
: Webprotokoll zur Kommunikation zwischen Browser und Server · k

I

Ingress
: Kubernetes-Ressource als konfiguration für die Weiterleitung von externem Traffic an interne Dienste · k

J

Java
: Programmiersprache · e

K

Kubernetes
: Open-Source-Orchestrierungssoftware zum Bereitstellen, Verwalten und Skalieren von Containern · d

M

Mongo
: dokumentbasiertes Datenbank-Managementsystem · e

O

OAuth
: standadisierte, sichere API-Autorisierung · h

P

Pods
: kleinste Einheit einer Kubernetes Anwendung bestehend aus einem oder mehreren Containern · h

Postgresql
: objektrelationales Datenbank-Managementsystem · e

Postman
: Werkzeug zum Testen von APIs · j

R

Repository
: virtueller Speicher · d

REST-API
: Interface zur Kommunikation über HTTP-Schnittstellen mit verschiedenen Systemen · f

S

Spring Initializr

webbasiertes Tool zur Generierung einer SpringBoot Projektstruktur · j

Spring-Boot
: quelloffenes Framework für die Java-Plattform · e

Swagger
: Format zur Beschreibung von API-Strukturen und -Funktionen · j

---

T

TestDrivenDevelopment
: Entwicklungsansatz bei dem zunächst der Test geschrieben und anschließend die Methode angepasst wird · k

---

U

UserInterface
: grafische Oberfläche als Schnittstelle zwiscehn programm und Benutzer · k

---

Y

yaml
: vereinfachte Auszeichnungssprache zur Datenserialisierung · j