REST-API = RESTful API
-> representational state transfer
-> verwendet um Anwendungen in microservice-architectures zu integrieren
-> mechanismus, der es einer Anwendung/einem Service ermöglicht auf eine Ressource in einer anderen Anwendung/einem anderen Service zu zugreifen
-> Zugreifender Service = Client
-> Service auf dem Zugegriffen wird = Service

- REST-APIs kommunizieren über HTTP-requests für read/write/update/delete innerhalb einer Ressource
    -> nutzen von GET requests für retrieval
    -> POST für create
    -> PUT für update
    -> DELETE …für delete, obvio
- alle HTTP-Methoden können in API-calls verwendet werden
- gut designte REST-APIs funktioniren wie eine website im Webbrowser
- Status einer Ressource zu einem Zeitpunkt/einer Instanz = resource representation
    -> diese Information kann in basically jedem Format geliefert werden (json, html, xlt, python, php, plaintext -> json ist beliebt weil Mensch und Maschine es lesen kann
- request headers sind wichtig, da diese URIs, metadata, Autorisation, caching, cookies und mehr beinhalten

Architektural constraints der Rest API:
1) Uniform Interface
- Alle API-Requests an die selbe Ressource müssen gleich sein, egal wo die Anfrage herkommt
- REST-API muss sicherstellen, dass die selben Daten nur eine URI (uniform resource identifier) haben
- (Ressourcen sollten nicht zu groß sein, aber alles enthalten, dass der Client brauchen könnte)

2) Decoupling Server & Client
- Sever und Client müssen unabhängig voneinander sein
- Client darf nur die URI der angefragten Ressource wissen & darf nicht anderweitig mit dem Server interagieren
- Server darf ebenso wenig die Client-Anwendung ändern/modifizieren & nur die angefragte Ressource via HTTP bereitstellen

3) Stateless
- REST-APIs sind stateless, d.h. der Server bearbeitet/erfüllt jede Anfrage unabhänig von vorherigen Anfragen -> Client kann Ressourcen in jeder Reihenfolge anfragen & jede Anfrage ist in sich geschlossen/unabhängig von einander
    -> Anfragen müssen immer alle benötigten Informationen enthalten
- Server Anwendungen dürfen keine Daten, die mit der Client Anfrage verbunden sind, speichern

4) Cacheability
- Ressourcen sollten, sofern möglich, Client-Side oder Server-Side cachebar sein (Caching is the ability to store copies of frequently accessed data in several places along the request-response path)
- Server-Antworten müssen Informationen darüber bereitstellen, ob caching für die gelieferte Ressource erlaubt ist
=> soll Client-Side Performance stärken & Server-Side Skalierbarkeit stärken
(Stateless & Cache -> basically: Caching = Bereithalten einer Kopie einer häufig Angefragten Ressource, der Server weiß, dass diese Ressource oft angefragt wird, weiß aber nichts über den Anfrager selbst (I think))

5) Layered System architecture
- es soll nicht davon ausgegangen werden, dass Client und Server direkt kommunizieren, im Normalfall stehen mehrere Vermittler/Zwischenstellen im Kommunikationsloop
=> REST-API soll so designed sein, dass weder Client noch Server weiß, ob sie mit einem Vermittler oder der Endsoftware kommunizieren

6) Code on Demand (optional)
- REST-APIs senden normalerweise statische Resourcen, sollte es jedoch Ausführbarer Code sein, so sollte dieser nur on demand ausgeführt werden

Anforderungen für uns:
- einfach zu deployen, so wenig Komponenten wie möglich, so viele Komponenten wie nötig
- skalierbare Nutzung, möglichst keine fixen Kosten durch dauerhaft angemietete Windows VMs oder ähnliches.
- HTTP-Statuscodes ausgeben -> wie genau wir einzelne Fehler behandeln ist uns überlassen
- etwa 100 Anfragen pro Werktag (=8h)
- Sicherheit für den Durchstich irrelevant, sonst OAuth2.0
- Rückgabe mit festdefiniertem Schema als JSON
- Rückgabe mit Klasse, Confidence, Bounding-Box, Liste der erkannten Texte (Text, Confidence, Boundingbox)
- API-Kosten (0-10€/1000 Aufrufe), möglichst niedrige Fixkosten -> „wenn eure Lösung steht werden wir das einfach testen und auswerten.“
- Doku OpenAPI Spezifikation in JSON oder YAML
- Thema CallbackURL erst später
- Übergabe reicht mir die aktualisierte YAML (mit Link zum Endpunkt) und ggf. kurze Anleitung wie der Endpunkt zu benutzen ist bzw. welche bekannten Besonderheiten / Probleme auftreten können.
(- Analysezeit pro Bild 30-60s)

Anforderungen (Out of Scope):
- Realeinsatz Anfrage = Normal ca. 2000 Aufrufe/8h
= Peak ca. 1000/h
- Wartung (keine Angabe/komplett out of scope für uns)

https://www.ibm.com/think/topics/rest-apis
https://www.techtarget.com/whatis/definition/URI-Uniform-Resource-Identifier
https://aws.amazon.com/what-is/restful-api/
https://restfulapi.net/caching/
http://web.archive.org/web/20130116005443/http://tomayko.com/writings/rest-to-my-wife