	RESTful	SPARQL	GRAPHCOOL
Was ist das überhaupt?	Ein architekturelles Konzept um eine API vom Client zu trennen.	Eine graphenbasierte Abfragesprache um auf ein Resource Description Framework zuzugreifen.	Zustandslose Abfragesprache, die die Definition von Daten durch den Client ermöglicht.
Warum sollte man das nutzen?	Es gibt keine übergeordneten Tools und REST interessiert die Art des Protokolls nicht. Es ist dafür konzipiert Daten von A nach B und zurück von B nach A zu transportieren. Durch eine REST Implementierung ist der Server nicht explizit an einen Client oder Protokoll gebunden.	Wenn wir Daten im RDF Format abfragen wollen, müssen wir zwangsweise auf SPARQL zurückgreifen. Die Abfragesprache wurde ausschließlich für diesen Zweck entwickelt. “SPARQL is to the Semantic Web (and, really, the Web in general) what SQL is to relational databases.” - Lee Feigenbaum Im Vergleich zu SQL werden die Beziehungen zwischen Daten explizit beschrieben. Dies ist ein völlig anderer Ansatz als bei relationalen Datenbanken, wo die Verknüpfung der Daten über Primary und Foreign Keys stattfindet. Die Daten beschreiben sich selbst nicht, wohin gegen ein RDF Triplet das tut. Die Abfragesprache SPARQL ist dadurch um einiges intuitiver. Stellen wir uns einmal vor wir benötigen eine Verkettung von mehr als nur 2 Datensätzen. Das wird in SQL schon aufwändig, wohingegen wir in SPARQL einfach nur den Graphen traversen müssen.	Im Gegensatz zu REST definiert GraphQL seine eigenen Conventions und funktioniert über einen einzigen HTTP Endpoint. Die Performance ist besser gegenüber normalen REST APIs und wir haben als Nutzer die Möglichkeit genau (besser gesagt wir müssen!) anzugeben welche Attribute eines Objektes wir zurückgeliefert bekommen wollen bzw generell eine Abfrage mit einer Objektstruktur zu stellen, anstelle eines Text-Strings. Der klare Vorteil hier liegt in der Deprecation von Attributen. Stellen wir uns mal vor wir wollen in unseren REST Routen ein Fieldset hinzufügen. Die Clients wären nicht mehr kompatibel. Sie MÜSSEN ihren Query-String bzw die Fieldset Parameter anpassen. Lösen würden wir das Problem durch Versionierung, doch man stelle sich einmal vor eine gesamte Version zu ändern, nur weil man EIN Fieldset ändern/entfernen/hinzufügen möchte. Ein grundlegendes Problem das mit GraphQL gelöst wird. Der Client weiß nichts davon wenn wir ein neues Attribut hinzugefügt haben, bis jemand das irgendwo auf der Seite findet und es seinem Query hinzufügt. Die Ansteuerung des Endpoints verändert sich nicht.
Use Case	Das WWW gibt sich größte Mühe uns NICHT zu erklären weshalb REST nun eigentlich besser ist als Non-Rest HTTP.	Immer dann, wenn wir Daten die als Graph repräsentiert sind queryn wollen. Sinnvoll wenn mehrere	Wenn gute Client Performance gefordert ist. Prinzipiell spricht nichts dagegen sowohl REST als auch GraphQL zu verwenden. Wir sehen in

	Unserer Meinung nach ist REST deshalb besser, weil die Web API sauberer und einfacher zu verstehen ist. Durch URIs die sich auf die jeweilige Methodik beziehen ist es für den Developer einfacher zu entwickeln und für den User einfacher anzuwenden. Bei Non-Rest HTTP gibt es nur Get und Post. Es mussten also alle Getter über Get und alle Setter über Post definiert werden. Durch Put und Delete haben wir jetzt vollständige CRUD Operations. REST sollte prinzipiell also immer mindestens als Standard vorhanden sein. Vorausgesetzt man setzt auf HATEOAS um die Kontroll-Logik der API zu definieren. Ansonsten wäre die API vermutlich eh nicht “wirklich” RESTful und GraphQL womöglich eine brauchbare aktuelle Alternative gegenüber REST Level 3.	Cross-References vorliegen. So könnten wir uns vorstellen das man RDF z.B. für ein digitales Gesetzbuch verwenden könnte.	GraphQL aber eine sinnvolle Alternative zu REST-Level 3 in RESTful APIs.
REST Level	2	0	Die Angabe eines REST-Levels für GraphQL wäre unzutreffend. REST-Level 3 beschreibt bei einer RESTful API, dass sinnvolle Strukturverweise in den Response mit aufgenommen werden. Das ist aber bei GraphQL grundsätzlich einfach überflüssig, da es bei GraphQL nur einen Root-Endpoint gibt, von dem alle Daten bezogen werden können. Eine Erklärung der Struktur durch Rückgabe von URIs ist prinzipiell also nicht mehr notwendig. Auf kurz oder lang denken wir, wird GraphQL den Ansatz des REST-Level 3 ersetzen.