API Versioning mit Ruby on Rails: Welche Edelsteine sind die Besten?

29.01.2018, 14:52 | IT, New Media & Software

Pressemitteilung von: *Applaunch*
Presseagentur: *Applaunch*

API Versioning mit Ruby on Rails: Welche Edelsteine sind die Besten?

Die API-Versionierung hilft, das Verhalten einer API für verschiedene Clients zu ändern. Eine API-Version wird von einer eingehenden Clientanforderung bestimmt und basiert entweder auf der Anforderungs-URL oder auf den Anforderungsheadern. Es gibt eine Reihe gültiger Ansätze für die Versionierung.

Wann ist die API-Versionierung erforderlich?

Die API-Versionierung kann in bestimmten Fällen ignoriert werden, z. B. wenn eine API als interner Client fungiert oder wenn eine API, die Sie bereits verwendet haben, einige kleinere Änderungen erfährt (z. B. Hinzufügen neuer Felder oder neuer Daten zur Antwort).

Wenn Sie jedoch einige wichtige Änderungen an Ihrem Code oder an der Geschäftslogik Ihrer App vornehmen und diese Änderungen sich auf vorhandene Clients auswirken, ist die API-Versionierung die einzige Möglichkeit, alte Clients nicht zu beschädigen.

Wie kann eine API-Version vom Client angegeben werden?
Hier ist eine Liste von Orten, an denen API-Versionen allgemein angegeben sind:

1. URL-Pfadparameter:

Die API-Version wird in den URL-Pfad eingefügt

HTTP GET:

https://domain.com/api/v2/resources

2. URL Get Parameter oder Anfrage body Parameter
HTTP GET:

https://domain.com/api/resources?version=v2

3. Akzeptieren Sie Header als versionierten Medientyp

HTTP GET:

https: // domain / api / bücher

Akzeptieren:

application / vnd.your_app_name.v2 + json

4. Benutzerdefinierter Header

HTTP GET:

https: // domain / api / bücher

API-VERSION: 2

Es gibt eine anhaltende Debatte darüber, wie man eine API-Version richtig spezifiziert.

URLs gelten nicht als ideal für diese Aufgabe, da sie eine Ressource, aber nicht die Version dieser Ressource darstellen. Dies ist jedoch der einfachste Ansatz und eignet sich zum Testen.

Ein benutzerdefinierter Header wird als übermäßig angesehen, da die HTTP-Spezifikation bereits den Accept-Header besitzt, der denselben Zweck erfüllt.

Die Header-API-Versionierung akzeptiert die beste Option gemäß der HTTP-Spezifikation. Es ist jedoch nicht einfach, solche APIs im Vergleich zu anderen Ansätzen zu testen. Da es nicht ausreicht, eine API-URL zu öffnen, müssen Sie eine Anfrage mit korrekten Headern verfassen.

Wenn es darum geht, welche Version einer API es zu wählen gilt, stimmen die meisten Entwickler der Verwendung der ersten API-Version als Standard zu. Wenn Ihr API-Client (iOS / Android-Gerät, Webbrowser usw.) keine erforderliche API-Version angibt, muss Ihre API die allererste Version der Antwort zurückgeben, da die einzige sichere Annahme ist, dass dieser Client zuvor erstellt wurde Die API hatte überhaupt eine Versionierung.

API Versionierung mit Ruby on Rails
Rails hat eine große Menge an Edelsteinen zum Erstellen von APIs mit Versionierung. Sehen wir uns ihre Fähigkeiten im Detail an.

Versionist
Dieses Schmuckstück unterstützt drei Versionierungsstrategien: HTTP-Header, URL-Pfad und Anforderungsparameter. Routen, Controller, Presenter / Serializer, Tests und Dokumentation sind Namensräume. Dies isoliert den Code einer API-Version von einem anderen. Dies kann übertrieben erscheinen, da die meisten Änderungen in Ansichten oder Serialisierern vorgenommen werden. Aber es ist korrekter, da das Isolieren von Logik innerhalb von Namespaces ein sauberer und offensichtlicherer Ansatz ist als der Umgang mit einer Mischung verschiedener Versionen innerhalb eines Controllers.

Um Routineaufgaben zu automatisieren, bietet versionist Rails-Generatoren zum Generieren neuer Versionen Ihrer API sowie neuer Komponenten innerhalb einer bestehenden Version. Es stellt auch einen Rails-Generator bereit, der eine vorhandene API-Version in eine neue API-Version kopiert. Dies funktioniert jedoch nicht gemäß dem DRY-Ansatz, da dies zu einer Code-Duplizierung führt. Ich habe diese Generatoren noch nie benutzt. Normalerweise erstelle ich alle

benötigten Controller und Serialisierer manuell. Ich kopiere auch nicht den gesamten Code von der vorherigen Version; Ich erben nur von der vorherigen Versionskontrolle.

Ein wesentlicher Nachteil des Versions-Edelsteins ist, dass der von ihm bereitgestellte API-Versions-Mechanismus keine Rückfälle auf die vorherige Version unterstützt, wenn die angegebene Logik nicht in die neue Version kopiert wurde. Das Juwel erwartet, dass der gesamte erforderliche Code in jeder neuen Version dupliziert wird. Aber wenn Sie nur ein Antwortformat ändern müssen, scheint dies übertrieben. Dieses Juwel ist aber immer noch ziemlich gut. Es ist leicht und konzentriert sich nur auf die API-Versionierung. Das ist im Vergleich zu einigen Edelsteinen, die bestimmte Methoden der API-Versionierung diktieren (z. B. rocket_pants und versioncake), nett.

Hier sehen Sie ein Beispiel für versionierte Routen aus dem Versionist-Juwel, das den Accept-Header mit dem versionierten Medientyp verwendet:
Namensraum: versionist_api do

api_version (

Header: {

Name: "Akzeptieren",

Wert: 'application / vnd.versionist_api.v2 + json'

},

Modul: "V2",

Standardwerte: {format:: json}

) machen

Ressourcen: nur Bücher: [: index,: create,: show,: update,: destroy]

Ende

api_version (

Header: {

Name: 'Akzeptieren',

Wert: 'application / vnd.versionist_api.v1 + json'

},

Modul: 'V1',

Standard: Wahr,

Standardwerte: {format:: json}

) machen

Ressourcen: nur Bücher: [: index,: create,: show,: update,: destroy]

Ende

Ende

Versionskuchen
Dieses Juwel hat einen anderen Ansatz. In den meisten Fällen erfolgt die Versionierung für API-Ansichten und Controller sind nicht Namespaced. Eine nette Eigenschaft von Versioncake ist, dass es Rückfälle zu früheren Versionen hat. Zusammen mit path, query param, accept header und custom header bietet es auch die Möglichkeit, einen eigenen Versionierungsansatz zu erstellen, der ein Anforderungsobjekt akzeptiert. Auf diese Weise können Entwickler eine API-Version an jeder beliebigen Stelle in der Anforderung in einer beliebigen Form angeben.

Da versioncake keinen Controller für jede Version unterstützt, verfügt es über spezielle Methoden, um auf die angeforderte Version und die neueste Version innerhalb der Instanz des Controllers zuzugreifen. Dies kann jedoch dazu führen, dass ein unerfahrener Entwickler schlechten Code schreibt, wenn er innerhalb von Controllern eine bedingte Logik hat, die von diesen Versionsparametern abhängt. In diesem Fall ist es besser, das Factory-Muster zu verwenden, bei dem die Controller-Aktion als einzelnes Objekt für jede Version implementiert wird (das interactor-Juwel kann für diesen Zweck verwendet werden).

Versioncake verfügt über eine Vielzahl von Funktionen (Details finden Sie in der Vergleichstabelle), einschließlich einiger exotischer Funktionen wie der Versionsabwertung. In einer Hinsicht sieht es wie eine vollständige Lösung für die API-Versionierung aus; aber in einem anderen scheint es ein bisschen schwer, da einige seiner zusätzlichen Funktionen möglicherweise nicht in generischen API-Anwendungsfällen verwendet werden.

Ein weiterer Nachteil von Versioncake ist, dass es sichtorientiert ist. Edelsteine wie jbuilder und rabl können mit versioncake verwendet werden, da ihre Vorlagen als Ansichten gespeichert werden. Aber modernere und populärere Edelsteine wie active_model_serializers können nicht mit versioncake verwendet werden. Dies kann in Ordnung sein, wenn Sie es vorziehen, einige Ansichtsteile als Teilbereiche zu verwenden (z. B. wenn Felder von Version 1 in einer Antwort der Version 2 vorhanden sind); Mit active_model_serializers können Sie die normale Vererbung von Ruby-Klassen verwenden.
Traube
Grape ist nicht nur ein API-Versions-Tool. Es ist ein REST-ähnliches API-Framework. Grape wurde entwickelt, um auf Rack zu laufen oder bestehende Webanwendungsframeworks wie Rails und Sinatra zu ergänzen, indem es eine einfache domänenspezifische Sprache zur Verfügung stellt, um einfach RESTful APIs zu entwickeln.

Was die API-Versionsverwaltung betrifft, bietet traube vier Strategien an: URL-Pfad, Accept-Header (ähnlich dem versionierten Medientyp-Ansatz), Accept-Version-Header und Request-Parameter.

Es ist auch möglich, Rückfälle zu früheren Versionen zu haben, die die spezifische Codeorganisation verwenden, die hier beschrieben wird: http://code.dblock.org/2013/07/19/evolving-apis-using-grape-api-versioning.htmlHier ist ein kurzes Beispiel von API Versioning Fallbacks in Trauben:

Und hier ist ein Modul für die Standardkonfiguration der ersten Version:

Modul GrapeApi

Modul V1

Modul Defaults

Erweitern Sie ActiveSupport :: Concern

enthalten tun

Dies würde die erste API-Version auf die zweite als Fallback reagieren lassen

Version ['v2', 'v1'], unter Verwendung von:: header, vendor: 'grape_api'

....

Ende

Ende

Ende

Und die zweite Version:

Modul GrapeApi

Modul V2

Modul Defaults

Erweitern Sie ActiveSupport :: Concern

enthalten tun

Version "v2", mit:: Pfad

Version 'v2' unter Verwendung von:: header, vendor: 'grape_api'

Ende

Ende

Ende

Bei trave_api / base.rb wird die zweite Version vor der ersten Version installiert. Damit können Anfragen an Version 2 mit V2-Logik (falls vorhanden) bearbeitet werden oder auf Version 1 zurückgreifen.

Modul GrapeApi

Klasse Base

Portrait

Wir von applaunch.io entwickeln seit über 5 Jahren gemeinsam mit unseren Kunden App Projekte zu verschiedenen Ideen aus den Bereichen Dating, Social Media, Messenger, Utility, Food, Sharing, Dienstleistungen, etc. Unsere Kunden sind dabei keine großen Agenturen oder Unternehmen, sondern vorwiegend Privatpersonen oder Startups mit guten Ideen. Die Besonderheit von applaunch.io liegt in der persönlichen Nähe zu jedem einzelnen Projekt. Wir sind selbst Gründer und haben eigene App Ideen realisiert. Wir beraten daher jeden unserer Kunden auch im Bereich Vermarktung, App Performance Optimierung, Zusammenarbeit mit Influencern und und und. Wer bei applaunch.io seine App entwickeln lässt, erhält somit am Ende ein Marktoptimiertes Produkt mit einem schönen Design und einer optimierten Handhabung. Während der gesamten Entwicklungsphase erhalten unsere Kunden zudem laufende Updates und Änderungsvorschläge, um die App optimal für den Markt vorzubereiten. Vor jeder App Entwicklung setzen wir uns gemeinsam mit unseren Kunden zusammen und gehen jedes Detail der Entwicklung durch. Bevor wir mit der Entwicklung anfangen, erhält jeder Kunde einen vollständigen App Prototypen in vorm eines klickbaren Designprototypen. Erst wenn die App genau den Anforderungen entspricht, wird mit der Entwicklung begonnen. Unsere Kunden schätzen dieses Vorgehen sehr und haben teilweise auch mehrere Projekte mit uns umgesetzt. Unsere Kerngebiete sind dabei React Native, Ionic, Swift, Native Entwicklung und Hybride Apps mit Anbindungen zu Web Browsern.

News-ID: 990619 • Views: 262 (Stand: 02.05.2026)

Link zur Pressemitteilung:

https://www.openpr.de/news/990619/API-Versioning-mit-Ruby-on-Rails-Welche-Edelsteine-sind-die-Besten.html