Wakanda-Magic

Einführung in Wakanda-Programmierung

www.aircraft-philipp.com

Inhalt

OO-Konzept, Wakanda-Basisprinzip

Objekt

- Ein Objekt repräsentiert eine reale Entität (Person), ein Konzept (Fussballspiel), eine logische Sache (Führerschein), eine physische Sache (Auto)
- Userdefinierter komplexer Datentyp
- Ein Objekt besteht aus zwei Dingen
    1. Zustand: Eigenschaften (Name, Adresse, Geburtsdatum einer Person)
    2. Verhalten: Methoden (Alter einer Person wird aus Geburtsdatum und aktuellem Datum berechnet)

Das OO-Konzept ist das Prinzip, nach dem Wakanda arbeitet.

Wakanda Eco-System

Einführungsvideo

Datenbank

- Wakanda enthält eine integrierte Datenbank. Link zur Doku

Webserver

- Wakanda integriert einen Webserver, der multithreading-fähig ist. Jeder Request öffnet einen eigenen Thread.

Studio

- Das integrierte Studio ist die grafische Benutzeroberfläche zum Editieren von Quellcode und stellt Tools zur Administration bereit.

Wakanda Eco-System

Solution – die oberste Ebene

- In Wakanda enthält eine Solution ein oder mehrere Projekte. Auf Solution-Level werden bspw. Zertifikate verwaltet oder User und Gruppen incl. Rollen und Rechten gemanagt. Diese haben dann Relevanz in allen in einer Solution vereinigten Projekten. Link zur Doku

Projekt – die Ebene unterhalb der Solution

- Ein Projekt beinhaltet alle relevanten Dateien, Einstellungen und eine interne DB für eine Web-Applikation (Mobil, Desktop, Backend only). Link zur Doku

Wakanda Datastore API (Datenbank)

Die Datastore API ist eine serverseitige API, die Ihnen Zugriff auf alle Klassen, Methoden und Eigenschaften gibt, die Sie benötigen, um den Inhalt eines Datenspeichers auf dem Wakanda Server zu verwalten und zu bearbeiten.

Mit der Datastore API können Sie Funktionen auf dem Server implementieren, die es Ihnen ermöglichen:

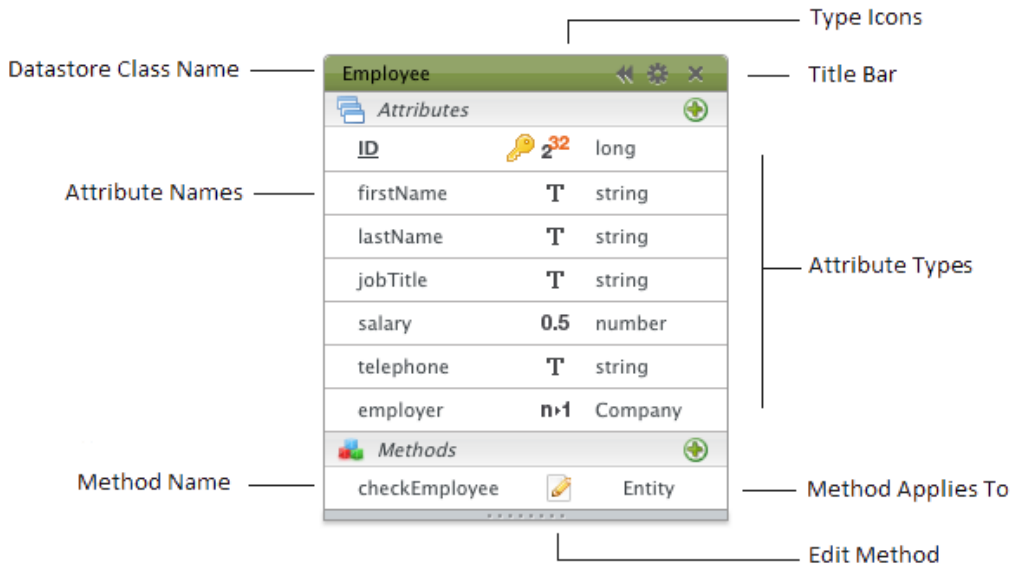
- Suchen oder Sortieren der Entitäten in einer Datenspeicherklasse,
- Transaktionen verwalten,
- Entitäten anlegen, ändern oder löschen und
- Zugriff auf Informationen über das Modell und die Daten in Ihrem Datenspeicher

In Wakanda ist ein Datenspeicher eine Menge von Daten, die strukturell durch ein Modell definiert sind. Das Modell enthält und beschreibt alle Datenspeicherklassen, aus denen der Datenspeicher besteht. Mit Wakanda Studio erstellte Modelle sind JSON-Dateien mit der Endung .waModel (z.B. employees.waModel).Daten beziehen sich auf die Informationen, die in diesem Modell verwendet und gespeichert werden sollen. Beispielsweise sind Namen, Adressen und Geburtsdaten von Mitarbeitern Daten, mit denen Sie in einem Datenspeicher arbeiten können. In Wakanda werden die Daten in einer Binärdatei mit dem Suffix .waData gespeichert (z.B. employees.waData). Standardmäßig wird diese Datei in einem Datenordner erstellt.

Class, Collection, Entity

Datastore Class:

ist eine Art von Struktur, die Attribute und Beziehungsattribute (Verknüpfungen zwischen Datenspeicherklassen) enthalten kann, um ihre Daten konzeptionell zu beschreiben und wie sie alle miteinander interagieren, sowie Methoden zur Interaktion mit den Daten im Modell.



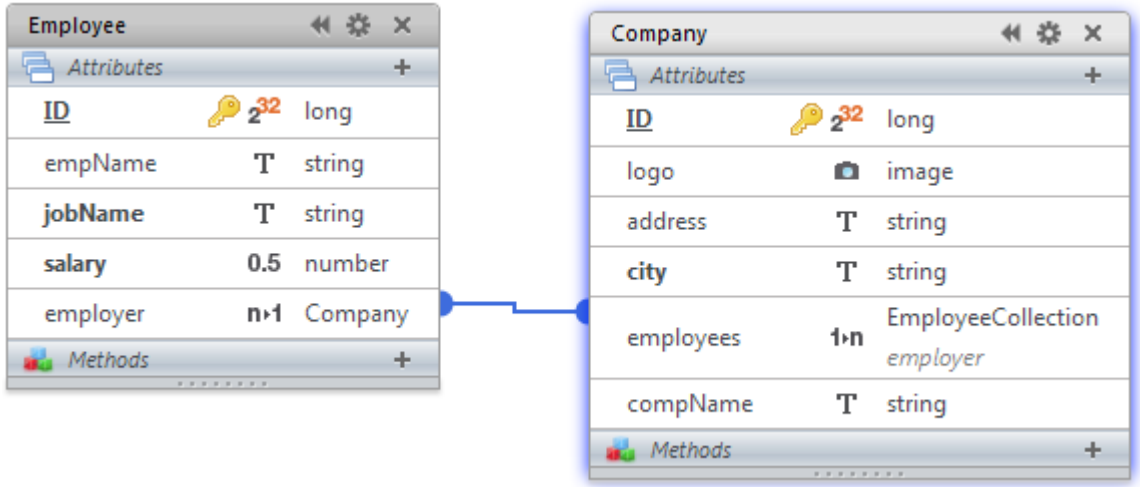
Dies entspricht einer Tabelle in einer SQL-DB.

Stellt Methoden und Events auf Klassenebene zur Verfügung.

Class, Collection, Entity

Entity Collection:

Entity collections sind Sets von Entitäten, die zur gleichen Datenspeicherklasse gehören. Eine Entity collection kann 0, 1 oder X Entitäten aus der Datenspeicherklasse enthalten, wobei X die Gesamtzahl der in der Datenspeicherklasse enthaltenen Entitäten darstellen kann. In den meisten Fällen können Sie mit dem Objekt "datastoreclass" oder "ds.datastoreclass" arbeiten wie mit einer Entity collection, die alle Entitäten von "datastoreclass" enthält.



Class, Collection, Entity

Entity:

In einem Wakanda-Datenspeicher ist die Entität die Grundeinheit der Information und wird durch die Datenspeicherklasse strukturiert, zu der sie gehört. Eine Entität ist ein Objekt, das für jedes Attribut der Klasse einen Wert enthält. Die Datastore API bietet alle Werkzeuge, die Sie benötigen, um auf Entitäten zuzugreifen, sie zu erstellen, zu modifizieren und zu löschen. Die Entity-Klasse stellt Ihnen Methoden zur Verfügung, mit denen Sie mit Entitäten arbeiten können, die in zwei Typen unterteilt sind:

- Methoden, die mit Entitäten und
- Methoden, die mit dem Entity-Iterator in einer Entity-Sammlung arbeiten.

Eine Entity entspricht einem Datensatz in einer SQL-DB.

Eine Entity stellt Methoden auf sich selbst zur Verfügung, der Zugriff erfolgt über „this“:

```
Employee.entityMethods.getEmployeeName = function(params) {
        return this.firstname;
}
Employee.entityMethods. getEmployeeName.scope = 'public';
```

Native Datentypen

Scalar Attribute Types

Modified in v11

Wakanda manages the following data types (which are null by default until a value is entered):

Data Type	Icon	Description	Value Range
blob		A "binary large object" containing binary data.	
bool		A Boolean value: either true or false.	
byte	2^8	A sequence of 8 bits.	
date		If the **Date only** property is selected for this attribute type, the date value will include only the "MM/DD/YYYY" format (e.g., "10/05/2010"). Otherwise, the date value including the time, stored in UTC. The date is expressed in the following format: YYYY-MM-DDTHH:MM:ss.SSSZ (e.g., "2010-10-05T23:00:00.000Z" for October 5, 2010 in the Central European Timezone). SSS, which represent the milliseconds, which can be between 0 to 999, has been added to the date in the Dev Branch.	
duration		A duration between two dates	
image		A reference to an image file or an actual image.	
long	2^{32}	A whole number, greater than or equal to a standard number	-2,147,483,648 to 2,147,483,647
long64	2^{64}	A whole number, greater than or equal to a standard number	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
number	0.5	A numeric value, corresponding either to a Real, and Integer or Long Integer.	±1.7e±308 (real), -32,768 to 32,767 (integer), -2^31 to (2^31)-1 (long)
string	T	A sequence of characters.	
uuid		Universally Unique Identifier: a 16 bytes (128 bits) number containing 32 hexadecimal characters	
word	2^{16}	A 16-bit signed integer	-32767 to 32768

Added in v10

object	A pure JavaScript object containing any kind of property/value pairs, including arrays. This data type can be indexed. Functions and recursive references are not supported	Example: `{x:2,y:"blue",z:{a:1, b:2}, v:[3,4,5]}`

Attribute und Optionen mit Datentypen

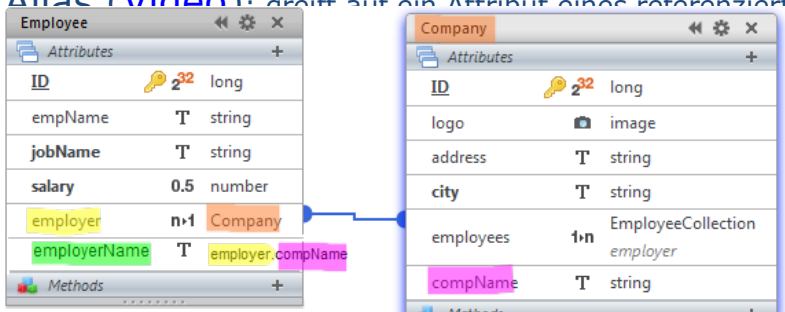
Wenn Attribute einer Datenklasse definiert/angelegt werden, können neben dem Datentyp auch weitere Parameter definiert werden.

- autoComplete
- unique
- defaultValue
- scope
- ….

Virtuelle Datentypen

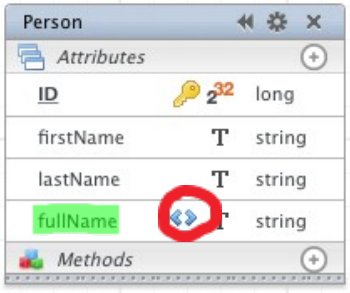
Es gibt 2 virtuelle Datentypen in Wakanda:

Alias (Video): greift auf ein Attribut eines referenzierten Unterobjektes zu:



Employee.employerName = new Attribute("alias", "string", "employer.compName");

Calculated (Video): berechnet ein Attribut dynamisch, wird ausgelöst durch Event

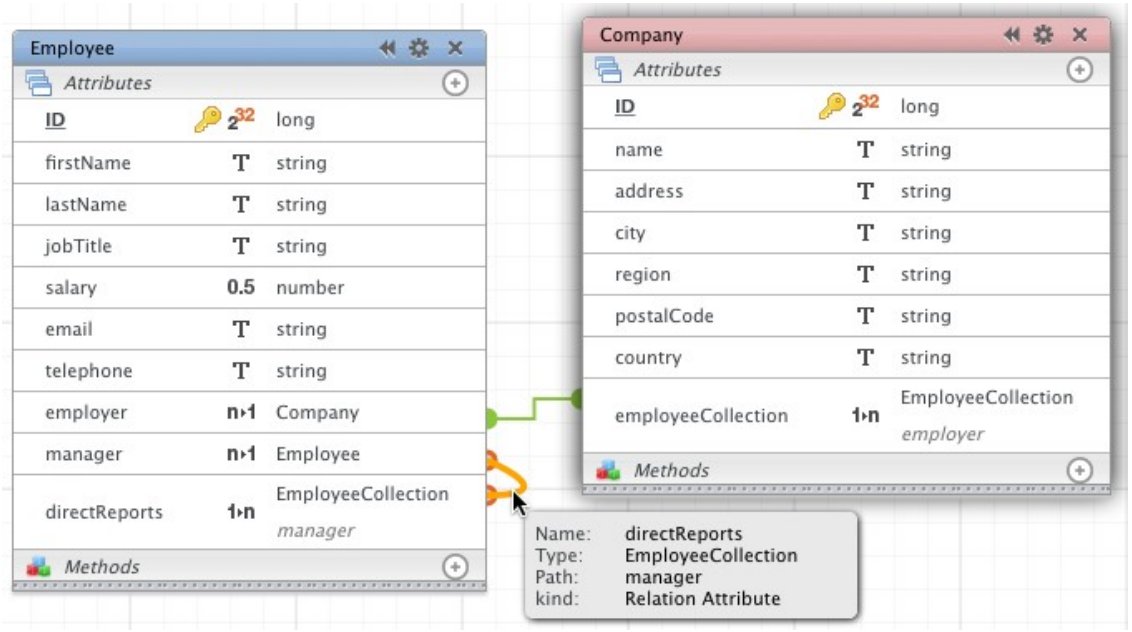


onGet, onSet, inQuery, onSort

```
Person.fullName.onGet = function() {
   return this.firstName + " " + this.lastName;
}
```

Rekursive Relationen innerhalb der gleichen Datenklasse

Ein rekursives Beziehungsattribut bezieht eine Entität in einer Datenspeicherklasse auf eine andere Entität in derselben Datenspeicherklasse. In unserem Fall hat jeder Mitarbeiter eine Führungskraft.



model.Employee.manager = new Attribute("relatedEntity", "Employee", "Employee");

model.Employee.directReports = new Attribute("relatedEntities", "EmployeeCollection", "manager", {reversePath:true});

Entity Stamp

Jede Entity hat ein Attribut ‚__STAMP'. Dieses wird komplett von der Datenbank verwaltet und dient dazu, Konflikte beim Modifizieren von einzelnen Entities (Datensätzen) zu vermeiden (Locking). Dabei gibt es zwei unterschiedliche Arten des Entity-Locking:

- **Optimistisches Locking**

  Alle Entitäten können immer in Read-Write geladen werden; es gibt kein a priori "Locking" von Entitäten. Jede Entität hat einen internen Locking-Stempel, der bei jedem Speichern inkrementiert wird. Wenn ein Client versucht, eine Entität mit der Methode save() auf dem Server zu speichern, vergleicht der Wakanda-Server den Stamp-Wert der zu speichernden Entität mit dem der in den Daten gefundenen Entität (im Falle einer Änderung):

  - Wenn die Werte übereinstimmen, wird die Entität gespeichert und der interne Stamp-Wert erhöht.
  - Wenn die Werte nicht übereinstimmen, bedeutet dies, dass ein anderer Mandant diese Entität in der Zwischenzeit geändert hat. Die Sicherung wird nicht durchgeführt und ein Fehler wird zurückgegeben.

- **Pessimistisches Locking**

  Sie können Entitäten bei Bedarf sperren und entsperren, wenn Sie auf Daten zugreifen. Wenn eine Entität innerhalb einer Benutzersitzung gesperrt wird, wird sie in dieser einzelnen Sitzung zum Lesen/Schreiben geladen, aber für alle anderen Sitzungen gesperrt. Die Entität kann in diesen Sitzungen nur im Nur-Lese-Modus geladen werden; ihre Werte können weder bearbeitet noch gespeichert werden.
  Dieses Feature basiert auf zwei Methoden der Entity-Klasse:

  - **lock()**
  - **unlock()**

Events

Zusätzlich zu den Datenklassenmethoden, die beim Aufruf ausgeführt werden, bietet Wakanda eine Reihe von Ereignissen sowohl auf Entitäten als auch auf Attributen, die die Ausführung von spezifischem Code auslösen können. Nicht zu verwechseln mit den Methoden On Get, On Set, On Sort und On Query eines berechneten Attributs, sind Ereignisse für alle Attribute und Datenspeicherklassen verfügbar.

Sie können jeder Datenspeicherklasse und jedem Attribut der Datenspeicherklasse ein oder mehrere Ereignisse zuordnen. Wenn Sie ein Ereignis für eine Datenspeicherklasse oder ein Datenspeicherklassenattribut anlegen, müssen Sie eine JavaScript-Funktion angeben, die bei jedem Auslösen des zugehörigen Ereignisses auf dem Server ausgeführt wird.Sie können die Ausführung des Ereigniscodes nicht direkt anstoßen. Ereignisse werden vom Server automatisch aufgerufen, basierend auf Benutzeraktionen oder Operationen, die durch Code auf Datenspeicherklassen und deren Attribute ausgeführt werden.Ereignisse werden auf der untersten Ebene des Datenspeichers, auf dem Server, ausgelöst. Sie werden im Allgemeinen verwendet, um Geschäftsregeln für die Datenspeicherklasse zu erstellen und zu verwalten. Beispielsweise kann das Ereignis onValidate/validate die Gültigkeit der in die Entität eingegebenen Daten überprüfen, unabhängig davon, woher diese Daten stammen: JavaScript-Code, Benutzereingaben, etc.

Events

Liste aller in Wakanda verfügbaren Events:

Event	Loading order	Context of call	Can be refused (*)	Comments
init	Datastore class then attributes	After a new entity is created in memory on the server	No	In this event, you can put code for initializing attributes, for example, the calculation of a custom ID.
load (Ds class)	Datastore class then attributes	After an entity is loaded on the server	No	In this event, you can put any calculations that set an attribute value or code that checks the level of user authorization.
load (Attribute)	Datastore class then attributes	After the first access to the attribute on the server	No	In this event, you can put code setting the initial value of an attribute.
validateremove	Attributes then Datastore class	Before removing an entity on the server (called before 'remove')	Yes	Checks validity of remove operation following business rules.
remove	Attributes then Datastore class	Before removing an entity on the server	Yes	You can use this event to remove any related entities within a transaction.
save	Datastore class then attributes	Before saving an entity on the server	Yes	You can use this event to check the data integrity of the entity.
validate	Attributes then Datastore class	Before saving an entity on the server (called before 'save')	Yes	Checks validity of data entered in the entity, for example a minimum.
clientrefresh	Datastore class then attributes	When the serverRefresh( ) method is executed on the client side	No	You can use this event to check the data integrity of the entity just like in a 'save' but in the context of a serverRefresh( )
set	Attributes only	After modifying an attribute value	No	Immediately recalculates values based on the attribute, for example, calculation of net pay from gross.
restrict	Datastore class only	When all the entities in a datastore class are accessed	No	Must return a collection. Allows you to control data available from the client side. Useful when combined with extended classes. Can be defined as a class property in the Model designer

Events

Events ablehnen:

Sie können die folgenden Ereignisse auf der Ebene der Datenspeicherklasse oder des Attributs "ablehnen":
save, validate, remove, validateremove

Wenn Sie ein Ereignis ablehnen, wird die entsprechende Aktion nicht ausgeführt, die Ereignisaufrufkette gestoppt und ein Fehler zurückgegeben. Um ein Ereignis abzulehnen, kann die vom Ereignis aufgerufene Funktion einfach ein Objekt zurückgeben, das die Fehlereigenschaft mit einem anderen Zahlenwert als Null enthält. In diesem Fall wird die Fehlernummer als Wert zurückgegeben. Optional kann dieses Objekt die Fehlermeldung über die Eigenschaft *errorMessage* enthalten, die einen String als Wert haben muss.

```
model.Employee.events.validate = function(event) {
        if(this.salary < 0) {
                        return { error: 100, errorMessage:"Salary cannot be negative" };
        }
}
```

Wenn der Server die save()-Anweisung für eine Entität erhält, deren Gehaltsattribut negativ ist, gibt das Validate-Ereignis den Fehler als JavaScript-Ausnahme zurück; das Save-Ereignis wird nicht aufgerufen und die Entität wird nicht gespeichert. Der Fehlerstapel enthält den generischen Fehler "Validation failed" sowie den benutzerdefinierten Fehler "Salary cannot be negative". Um Sicherungs- oder Validierungsfehler verarbeiten zu können, können Sie Aufrufe vom Typ ***save()*** in eine Try/Catch-Struktur einfügen:

```
try
{
   emp.save();
}
catch (err)
{
   myerr = err;
   // process the error here
}
```

Events

Clientseitige Verarbeitung des zurückgegebenen Fehlers:

Innerhalb eines Promise wird der Fehler in der onError-Sektion empfangen und verarbeitet.

```
Observable.fromPromise(aPromise).subscribe(
        next => {
        },
        err => {
                const _E = JSON.parse(err.response).__ERROR[0];
                usrMsg.set({ class: 'failed', content: _E['message'] });
        },
        () => {
        }
);
```

Events – „restricting Queries“

Mit Wakanda können Sie automatisch einschränkende Abfragen für jede Ihrer Datenspeicherklassen erstellen. Eine einschränkende Abfrage wird automatisch angewendet, wenn auf alle Entitäten in einer Datenspeicherklasse zugegriffen wird, entweder durch Auffinden einer Entität per Schlüssel oder durch Verwendung einer dieser Datenklassenmethoden:

all(), query() oder find()

Einschränkende Abfragen können auf jede beliebige Datenspeicherklasse angewendet werden, sind aber besonders nützlich bei erweiterten (vererbten) Datenspeicherklassen und deren Attributen. In diesem Zusammenhang können sie automatisch Entity-Collections in den abgeleiteten Datenspeicherklassen auf Basis der in den erweiterten Datenspeicherklassen-Entitäten gespeicherten Werte erzeugen. Beispielsweise könnte die Entity-Sammlung in der Datenspeicherklasse 'ClosedIssue' (die von der Datenspeicherklasse 'Issue' abgeleitet wurde) auf der folgenden einschränkenden Abfrage basieren:

ds.Issue.query("isClosed = :1", true)

Die Einschränkung von Abfragen in Kombination mit erweiterten Klassen ist auch für Sicherheits- oder Benutzer-/Gruppenprobleme nützlich, da sie die Kontrolle der verfügbaren Daten auf der Client-Seite ermöglicht. Die Verwendung von Systemvariablen wie *$userid* bei der Einschränkung von Abfragen ist ebenfalls eine leistungsstarke Funktion zur Steuerung des Datenzugriffs, basierend auf aktuellen Werten.

Wakanda bietet zwei Möglichkeiten, eine einschränkende Abfrage zu schreiben:

- in der Datenspeicherklassen-Eigenschaft “Restricting Query“, die aber nur im Datenmodell-Editor zur Verfügung steht.
- im Ereignis „restrict“, das für Datenspeicherklassen und Attribute zur Verfügung steht. In diesem Fall können Sie z.B. einen beliebigen Abfragecode schreiben, der eine Collection zurückgibt:

```
dataClass.events.restrict = function(event){
    var col = ds.Issue.query("isClosed =:1", true);
    return col; // a collection must be returned
}
```

Events – „restricting Queries“

Programmierprinzipien für die einschränkende Queries:

Die Einschränkung von Queries kann eine beliebige Berechnung enthalten, die für Ihre Geschäftsanwendung sinnvoll ist. Queries können Attributwerte vergleichen und/oder Benutzerberechtigungen prüfen. Sie können dann entweder Sammlungen oder Entitäten zurückgeben.

Es ist jedoch wichtig, dass Sie beim Schreiben einer einschränkenden Abfrage die folgenden Prinzipien beachten:

Immer eine Sammlung zurückgeben

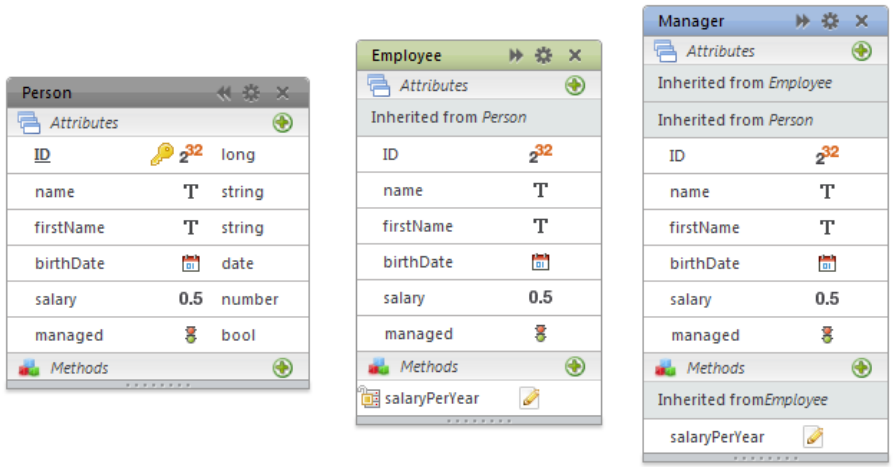
Wenn Sie ein restrict-Event verwenden, muss die Funktion eine gültige Sammlung aus der Datenspeicherklasse zurückgeben. Wenn Sie es unterlassen, eine Sammlung zurückzugeben, oder wenn die zurückgegebene Sammlung ungültig (undefiniert) ist, sendet der Server keine Entitäten und Sie erhalten eine leere Sammlung.

Immer auf übergeordnete Klasse abfragen

Bei Verwendung im Kontext von erweiterten Datenspeicherklassen muss immer eine einschränkende Abfrage auf die "erweiterte" (oder "übergeordnete") Datenspeicherklasse angewendet werden. Wenn die Abfrage auf die abgeleitete Datenspeicherklasse angewendet wird, sind die Ergebnisse ungültig.

Betrachten Sie beispielsweise das folgende Modell, bei dem Mitarbeiter von Person und Manager von Mitarbeiter erweitert wird. Die Klasse Manager sollte eine Sammlung von Mitarbeitern enthalten, die derzeit Manager sind. Die jeweiligen Entity-Sammlungen basieren auf der Einschränkung von Abfragen, die auf ein übergeordnetes Datenspeicherklassenattribut filtern:

Events – „restricting Queries“



Employee restricting query event (wrong code)	Manager restricting query event (wrong code)
`var col=ds.Employee.query("salary isnot null"); return col;`	`var col=ds.Manager.query("managed=true"); return col;`

Das wird NICHT funktionieren, da ein Konflikt in der einschränkenden Query-Kette existiert. Als Ergebnis erhalten Sie keinen Fehler, aber der Konflikt erzeugt einen falschen Wert: Sie erhalten Personen, für die das verwaltete Attribut "wahr" ist, die aber nicht unbedingt Angestellte sind.

Der KORREKTE Weg, die einschränkenden Query-Events zu schreiben, ist:

Employee restricting query event (correct code)	Manager restricting query event (correct code)
`var col=ds.Person.query("salary isnot null"); return col;`	`var col=ds.Employee.query("managed=true"); return col;`

Funktionen / Methoden

Zusätzlich zu Attributen können Datastore-Klassen auch eine andere Art von benutzerdefinierten Eigenschaften enthalten, die als Methoden bezeichnet werden. Methoden sind Teile der Programmierlogik, die beim Aufruf ausgeführt werden. Methoden der Datastore-Klasse können einen von drei Scopes haben: Entität, Collection o. Klasse. Der Scope jeder Methode ist im Wakanda-Modell definiert. Zur Laufzeit werden Klassenmethoden mit dem Scope „Entity" zu einer Eigenschaft aller Entitäten der Klasse. In diesem Scope bezieht sich das Schlüsselwort „*this*" auf die Entität. Entity-Methoden im Scope „Collection" werden zu Eigenschaften von Entity-Collections für diese Klasse. In diesem Scope bezieht sich „this" auf die Entity-Collection. Der Umfang von Collection erlaubt es Entwicklern, Methoden zu entwickeln, die über eine gesamte Collection funktionieren. Datastore-Klassenmethoden mit dem Scope „Class" sind im Wesentlichen statische Methoden und benötigen keine Instanz einer Entität oder Entity-Collection, um zu funktionieren. In diesem Bereich bezieht sich das Schlüsselwort „this" auf die Klasse selbst.

Klassenmethode definieren:

```
dataClass.methods.methodName = function(params) {
// code here
}
dataClass.methods.methodName.scope = 'public';
```

Collectionmerhode definieren:

```
dataClass.collectionMethods.methodName = function(params) {
// code here
}
dataClass.collectionMethods.methodName.scope = 'public';
```

Entitymethode definieren:

```
dataClass.entityMethods.methodName = function(params) {
// code here
}
dataClass.entityMethods.methodName.scope = 'public';
```

Zugriff / Aufruf von Funktionen / Methoden

Der Zugriff auf Methoden der Datenspeicherklasse wird über zwei Eigenschaften gesteuert:

Permissions: Mit diesem Parameter können Sie eine Benutzergruppe zu Aktionen hinzufügen, so dass nur die angemeldeten Benutzer der Gruppe die zugehörige Aktion ausführen dürfen.

Scope: Der Methodenscope definiert die Kontexte, in denen er veröffentlicht wird. Eine "öffentliche" Datenklassenmethode kann von überall her verwendet werden. Eine "public on server"-Datenspeicherklassenmethode kann nur vom Server aus verwendet werden. Standardmäßig ist der Bereich "public on server".

Aufruf von Methoden der Datenspeicherklasse:

Standardmäßig können alle Methoden der Datenspeicherklasse aus serverseitigem JavaScript-Code als Attribute der Datenspeicherklasse aufgerufen werden (verfügbar über das ds-Objekt). Zum Beispiel:

```
var allTeachers =  ds.Course.getAllTeachers();
```

Methoden der Datastore-Klasse können über einen REST-HTTP-Request aufgerufen werden, sofern ihr Scope „public“ ist. Nur Klassen- und Collection-Datenspeicherklassen können durch REST-Anfragen aufgerufen werden, auf Entity-Ebene nicht.

Vererbung

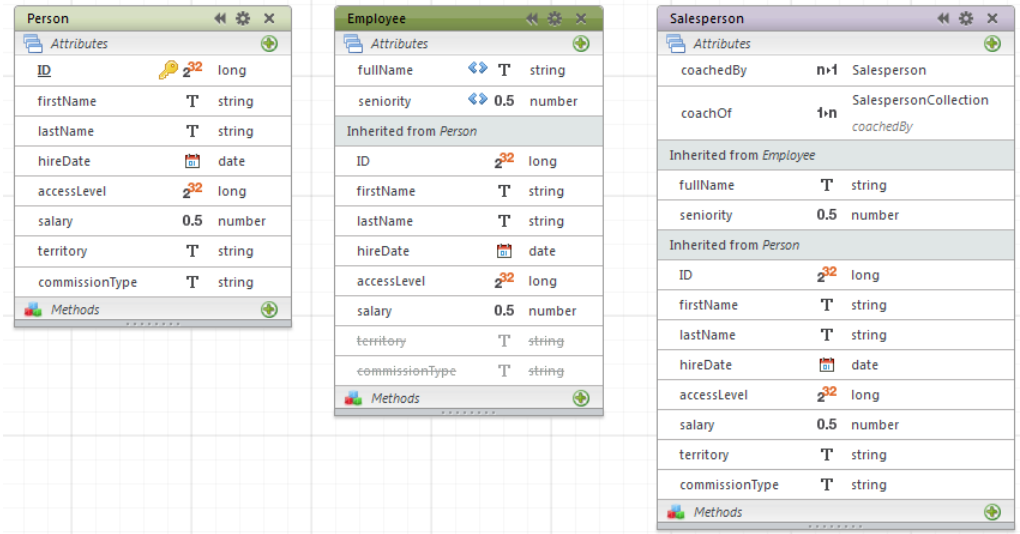
Obwohl in der objektorientierten Programmierung üblich, ist Vererbung ein Begriff, der in der Datenmanagement-Welt nicht üblich ist. Es gibt jedoch einige Datensituationen, die von der Vererbung profitieren könnten. Sehen wir uns folgende partielle relationale Datenbankstruktur an:

Diese normalisierte Struktur beschreibt Personen (innerhalb und außerhalb der Organisation), Mitarbeiter (nur innerhalb der Organisation) und Vertriebsmitarbeiter (eine bestimmte Untergruppe von Mitarbeitern). Obwohl im obigen Diagramm mit N->1-Beziehungen dargestellt, sehen unsere Geschäftsregeln vor, dass jeder Vertriebsmitarbeiter genau einen Mitarbeiter und jeder Mitarbeiter genau eine Person repräsentiert. Ein Verkäufer kann nicht ohne Bezug zu einem Mitarbeiter existieren und ein Mitarbeiter kann nicht ohne Bezug zu einer Person existieren. Darüber hinaus leiten Mitarbeiterdatensätze ihren Schlüsselwert aus dem entsprechenden Personendatensatz und Verkäufer ihren Schlüssel aus dem entsprechenden Mitarbeiter ab. Weniger formal können wir sagen, dass ein Verkäufer eine Art Mitarbeiter ist, und dass ein Mitarbeiter eine Art Person ist. Später, wenn das Datenverwaltungssystem Verkäufer ansprechen muss, kann es dies tun, indem es auf der Salespeople-Tabelle arbeitet, aber natürlich muss es sich mit den anderen Tabellen verbinden, um auf ein vollständiges Bild der Daten zuzugreifen.

Alternativ könnten wir die Daten denormalisieren und alle Felder in einer Tabelle zusammenfassen, aber dann müssten wir zusätzliche Felder hinzufügen, die die verschiedenen Datensatztypen von Mitarbeitern und Verkäufern kategorisieren und diese Typen verwalten. Außerdem, wenn wir alle Felder in einer Tabelle zusammenfassen, müsste jede Abfrage ein Merkmal Verkäufer enthalten, um nur Datensätze vom Typ Verkäufer zu identifizieren. Dies stellt eine größere Belastung für die nachgelagerte Geschäftslogik dar, um die verschiedenen Arten von Personen und ihre entsprechenden signifikanten Bereiche zu identifizieren.

Vererbung

Diese Situation ist ein Beispiel, das besser durch Vererbung angegangen wird. In Wakanda tun Sie dies, indem Sie eine Basisdatenspeicherklasse erweitern, um eine neue Klasse anzulegen. Also, in Wakanda könnte das Modell werden:



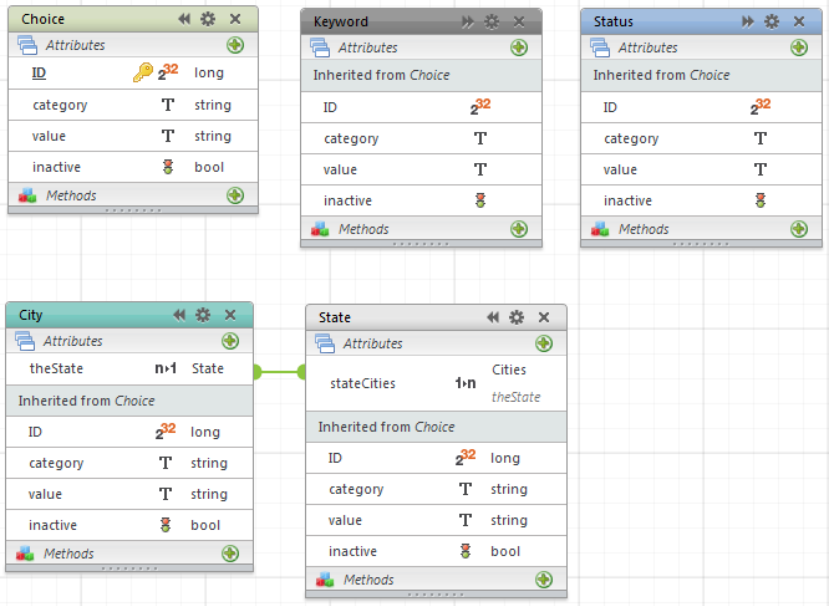
Im obigen Wakanda-Modell erbt die Klasse Salesperson von der Klasse Employee, die von Person erbt. Später zur Laufzeit setzt sich jede Mitarbeiter-Entität aus allen Eigenschaften von Mitarbeiter und Person zusammen (abzüglich entfernter Attribute) und jede Verkäufer-Entität aus Eigenschaften von Person sowie den für Mitarbeiter und für sich selbst hinzugefügten Eigenschaften. Normalerweise werden Entitäten in jeder Klasse durch " restricting queries " verwaltet, die beim Zugriff auf eine Klasse automatisch entsprechende Entity Collections erstellen. Beispielsweise könnte die Query zur Einschränkung der Klasse "Mitarbeiter" wie "Gehalt ist nicht null" aussehen (siehe unten: Einschränkung von Queries und Vererbung). Wenn eine Wakanda-Anwendung nur mit Mitarbeitern oder Verkäufern arbeiten soll, muss sie nicht jede Abfrage explizit verfeinern, um nicht benötigte Entitäten herauszufiltern". Darüber hinaus verwaltet Wakanda beim Anlegen einer abgeleiteten Klasse die zugrundeliegenden Daten und generiert bei Bedarf die vererbten Entitäten. Wenn z.B. eine neue Vertriebsmitarbeiter-Entität angelegt und gespeichert wird, verwaltet Wakanda automatisch das Anlegen des entsprechenden neuen Mitarbeiters und der entsprechenden Person. Wenn der Klasse Person ein neues Attribut hinzugefügt wird, steht es in allen abgeleiteten Datenspeicherklassen zur Verfügung.

Vererbung

Einschränkende Abfragen und Vererbung

Ohne eine einschränkende Abfrage verwaltet Wakanda die Vererbungsklassifizierung. Das heißt, wenn eine abgeleitete Entität angelegt wird, stellt Wakanda sicher, dass sie als Entität in der abgeleiteten Datenklasse verfügbar ist. Dieses Verhalten kann jedoch durch eine einschränkende Abfrage übersteuert werden. Wenn eine einschränkende Query für eine abgeleitete Datenspeicherklasse bereitgestellt wird, wird die Query zur Grundlage für die Vererbungsklassifizierung. Dies ermöglicht dem Entwickler den direkten Zugriff auf die Klassifizierung der Entität und bietet eine Möglichkeit, Entitäten entlang der Vererbungskette zu fördern und zu degradieren.

Betrachten wir zum Beispiel das folgende Modell:



Typisch für viele Datenanwendungen, bezeichnet dieses Modell Datenspeicherklassen, die zur Unterstützung von Benutzeroberflächen-Pick-List-Mechanismen verwendet werden. Choice ist die Basisklasse für die anderen vier. Schlüsselwort, Status, Stadt und Staat sind alle abgeleitete Klassen. Jede der abgeleiteten Klassen enthält eine einschränkende Query in Form von:

Vererbung

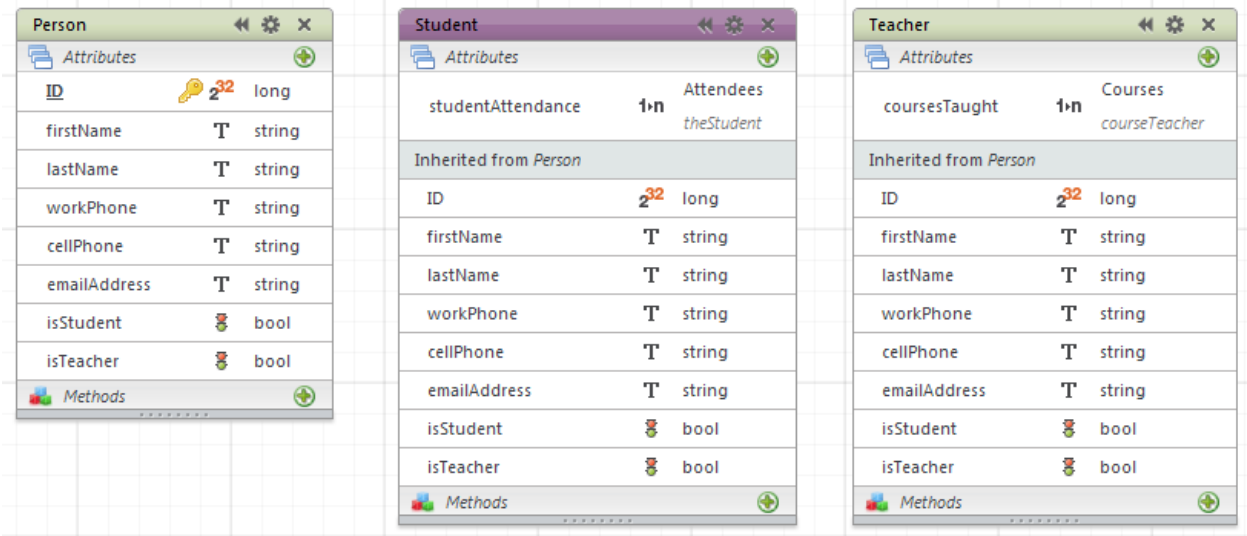
Beachten Sie, dass das Attribut category mit dem Namen der abgeleiteten Klasse verglichen wird. Im Einsatz kann jede der abgeleiteten Klassen verwendet werden, ohne dass zur Laufzeit eine Filterabfrage eingebunden werden muss. Wenn wir einer abgeleiteten Klasse bestimmte Attribute hinzufügen müssen, können wir dies tun, ohne die anderen Typen zu beeinflussen. Wenn wir jedoch der Basisklasse Choice neue Attribute hinzufügen, z.B. sortOrder oder Abkürzung, werden diese Attribute in allen abgeleiteten Datenspeicherklassen verfügbar. Wir können die Klassifizierung von abgeleiteten Entitäten automatisieren, indem wir eine Methode für das OnInit-Ereignis der Klasse Choice wie folgt hinzufügen:

```
onInit:function() {
        //get the name of the class of the entity
        var myType = this.getDataClass().getName();
        //store it in the category attribute
        this.category = myType;
}
```

Das Ereignis On Init wird ausgeführt, wenn eine Entität angelegt wird (mehr dazu später). Der obige Code bestimmt den Namen der Klasse und ordnet ihn der Kategorie zu, wodurch die Entität automatisch kategorisiert wird. Da alle abgeleiteten Datenspeicherklassen dieses Verhalten erben, wird die Kategorie beim Anlegen eines der abgeleiteten Typen gesetzt. Wenn wir eine Entität im Basistyp Choice anlegen, wird ihr eine Kategorie "Choice" zugewiesen, die jedoch vor dem Speichern der Entität geändert werden kann. Beachten Sie auch, dass die abgeleiteten Klassen State und City eine Beziehung zwischen ihnen haben. Abgeleitete Klassen können so miteinander verknüpft werden, dass ihre entsprechenden Basisklassen nicht miteinander verbunden sind.

Natürlich impliziert das obige Vererbungsschema, dass eine einzelne Entität nur ein Typ wie Schlüsselwort, Status, Stadt, Staat oder Auswahl sein kann. Aber die Wakanda-Vererbung erlaubt es, dass eine einzelne Entität mehr als ein vererbter Typ ist. Betrachten wir das folgende Modell:

Vererbung



In diesem Modell werden sowohl Schüler als auch Lehrer von der Person abgeleitet. Schüler ist abhängig von der folgenden einschränkenden Abfrage isStudent = True, während Lehrer abhängig von der einschränkenden Abfrage isTeacher = True ist. Da diese beiden Werte unabhängig sind, kann eine Entität in der Klasse Person sowohl ein Student als auch ein Lehrer sein.

Manchmal ist ein einschränkender Query-String nicht flexibel genug. Wie würden wir einigen Benutzern den Zugriff auf mehrere Entitäten erlauben, anderen aber den Zugriff einschränken? Ein flexiblerer Ansatz ist die Verwendung des Datenklassenereignisses On Restricting Query. Dieses Ereignis dient dazu, eine Entity-Sammlung zurückzugeben, die alle Entitäten der Datenklasse repräsentiert. Der Vorteil dieses Ansatzes ist, dass Sie programmatisch eine Entity-Sammlung bereitstellen können, die für jeden authentifizierten Benutzer unterschiedlich sein kann.

Restrict-Event

Das Ereignis On Restricting Query kann neben der Vererbung verwendet werden und kann zu jeder Datenspeicherklasse hinzugefügt werden. Beispielsweise kann eine Datenklasse Verkauf eine einschränkende Query-Logik haben, die den Verkauf auf den aktuellen Benutzer beschränkt. Der Vorteil dabei ist, dass kein Benutzer auf die Verkäufe eines anderen Benutzers zugreifen kann, unabhängig davon, wie die Informationen abgerufen werden.

REST-API

Mit der REST-API können Sie Informationen über die Datenspeicherklassen in Ihrem Projekt abrufen, Daten manipulieren, sich in Ihre Webanwendung einloggen und vieles mehr. Dieses Handbuch ist in drei Kategorien unterteilt:

- Benutzer authentifizieren
- Allgemeine Informationen
- Daten manipulieren

REST Requests

Bei jeder REST-Anfrage gibt der Server den Status und eine Antwort (mit oder ohne Fehler) zurück.

Status der Anfrage

Mit jedem REST-Request erhalten Sie den Status zusammen mit der Antwort. Nachfolgend finden Sie einige der Status, die auftreten können:

Status	Description
0	Request not processed (server might not be started).
200 OK	Request processed without error.
401 Unauthorized	Permissions error (check user's permissions).
404 Not Found	The datastore class is not accessible via REST ("Public on Server" as scope) or the entity set doesn't exist.
500 Internal Server Error	Error processing the REST request.

Antwort

Die Antwort (im JSON-Format) variiert je nach Anforderung.
Wenn ein Fehler auftritt, wird er zusammen mit der Antwort vom Server gesendet oder es ist die Antwort vom Server.

REST-API

Über die REST-API können auch serverseitige Methoden zur Verfügung gestellt werden. Da Datenklassen auch Methoden haben können und diese über die REST-API ansprechbar sind, können wir diesen Mechanismus nutzen und über den Pfad der Datenklasse die Methode aufrufen:

http://server:port/rest/Datenklasse/methode()

Dabei muss die Datenklasse selbst keinerlei Attribute haben, kann also leer sein. Die Datenklasse muss im Scope „public" sein und die Methoden müssen ebenfalls im public-Scope liegen.

Damit können wir neben der Datenbank auch alle anderen Funktionalitäten über die REST-API publizieren. In den aufgerufenen Methoden stehen alle Systemfunktionen zur Verfügung.

Worker (Threads)

Wakanda bietet eine Möglichkeit, JavaScript-Code und Systemprozesse in separaten Threads mittels Workern auszuführen. Ein Worker wird durch Verweis auf eine einzelne JavaScript-Datei erstellt. Wenn der Worker instanziiert wird, wird er zu einem Objekt im Speicher, das darauf wartet, aufgerufen zu werden. Ein Worker kann eine beliebige Anzahl von internen Methoden haben; er muss jedoch eine On-Message-Funktion implementieren, um auf externe Ereignisse zu reagieren, wenn es welche gibt.

Es gibt 3 Arten von Workern:

- Dedizierter Worker
- Shared Worker
- Node Worker

Ein dedizierter Worker kann nur über den übergeordneten Thread angesprochen werden, der ihn erstellt hat, während ein Shared Worker über einen beliebigen Thread angesprochen werden kann. Dedizierte Worker enden, wenn der Eltern-Thread endet, während Shared Worker weiterhin existieren, selbst wenn der Thread, der den Shared Worker erzeugt hat, endet.

Beispiele für Dedicated und Shared Worker.

Node Worker:

- Ein Node Worker führt bestimmten Code in einem separaten Thread aus.
- Sie können mit einem NodeWorker von überall in Ihrer Anwendung kommunizieren.
- Sie können NodeJS APIs und Module innerhalb eines Node Workers verwenden.

Node Worker Doku

Angular-Wakanda

Website mit Download und Quickstart

Angular-Wakanda ist ein Modul, mit dem es möglich ist innerhalb einer AngularJS oder einer Angular2/4/5/6 – Anwendung mit der Wakanda-DB zu interagieren. Grundlage des Moduls ist die REST-API, das Modul stellt alle Funktionen der REST-API bequem zur Verfügung.

Das Modul stellt die folgenden Funktionen zur Verfügung:

Authentifizierung

Um die erforderlichen Berechtigungen für die Verwendung von DataClasses zu erhalten, ist es oft erforderlich, sich zu authentifizieren. Wakanda Backend hat ein natives Verzeichnis und kann benutzerdefinierte Authentifizierungen verarbeiten. Der Konnektor stellt eine API für die Authentifizierung zur Verfügung.

Authentifizierungsmethoden:

login(), logout(), currentUser(), currentUserBelongsTo(groupName)

Configuration

wakandaConfig ist ein nützlicher Provider, um auf den Hostnamen des Wakanda-Servers zuzugreifen, mit dem die Anwendung arbeitet.

Config-Methoden:

setHostname(hostname), getHostname()

Angular-Wakanda

DataClass API Reference

Ein Hinweis zu den zurückgegebenen Werten

Viele der dataClass-Methoden geben ein leeres Objekt oder Array zurück, wobei ein $promise-Objekt auf das Promise-Objekt der Operation verweist. Wenn das Promise-Objekt aufgelöst wird, ist das Objekt oder Array gefüllt. So können Sie eine direkte Einbindung vornehmen, und das Objekt oder Array wird korrekt gefüllt, wenn die asynchrone Behandlung endet.

Verfügbare Methoden:

find(), all(), query(), create(), userDefined()

Metadaten:

name, collectionName, attr()

Entity API Reference

Alle in der Datenklasse definierten Attribute sind direkt in allen Entitäten dieser Datenklasse verfügbar.

Zurückgegebene Werte werden wie in der dataClass API als Promise-Objekt ausgeführt.

Verfügbare Methoden:

save(), remove(), fetch(), recompute(), toJSON(), userDefined()

Angular-Wakanda

Entity collection API Reference

Zugriff auf die Entität:

Entity-Collections sind Arrays, mit einigen Utility-Methoden. Sie können direkt mit dem Operator [] auf ihre Objekte zugreifen. Elemente in der Entity-Collection sind Entitäten, so dass alle Methoden, die in der Entity-API-Referenz beschrieben sind, auf ihnen verfügbar sind.

Zurückgegebene Werte werden wie in der dataClass API als Promise-Objekt ausgeführt.

Verfügbare Properties:

totalCount, queryParams

Daten abrufen:

fetch()

Paging:

nextPage(), prevPage(), more()

Methoden:

toJSON(), userDefined()

Übungen

Neue Datenklassen anlegen

Attribute als native Datentypen definieren

Attribute als Alias und als berechnete Typen definieren

Methoden auf Datenklasse, Collection und Entity anlegen, via REST testen

Neue Datenklasse aus bestehenden Datenklassen ableiten (Vererbung)

Related Entity/EntityCollections anlegen

Shared Worker anlegen und testen

Wakanda-Angular untersuchen (Datenbrowser)