dJStorm Documentation

Release 0.1

maccesch

Sep 27, 2017

Contents

This ORM is similar to the very elegant one of Django but this one is asynchronous.

For maximum compatibility it is completely framework agnostic.

Currently only the HTML5 Local Database is supported as backend. But due to the asynchronous architecture anything is thinkable.

Please note, that this is still in alpha stage. But feel free to try it out. You should already be able to use it in some real situations.

Any help in developing this further is welcome.

CHAPTER 1

Getting Started

Installation

All you need is the file https://github.com/maccesch/djstorm/raw/master/djstorm.js (right click -> save as...)

Since there are no required libraries or frameworks just add the script to your project.

For example in a website:

```
<head>
        ...
        <script type="text/javascript" src="path/to/djstorm.js"></script>
        ...
</head>
```

Quickstart

Here is a complete example of how to use djstorm.js

Include the script:

```
<script type="text/javascript" src="/js/djstorm.js"></script>

<script type="text/javascript">
```

Define the model:

```
var Language = new Model({
    Meta: {
        dbTable: "languages"
    },

    name: new CharField({ maxLength: 50 }),
```

```
    shortcut: new CharField({ maxLength: 5, primaryKey: true }),

    toString: function() {
        return this.name;
    }
});
```

And another more complex one:

```
var TYPE_CHOICES = [
    [1, "Book"],
    [2, "Brochure"],
    [3, "Flyer"]
];

var Literature = new Model({
    Meta: {
        dbTable: "literature_types"
    },

    title: new CharField(),
    author: new CharField({ maxLength: 50 }),
    orderId: new CharField({ maxLength: 10, primaryKey: true }),
    type: new IntegerField({ choices: TYPE_CHOICES }),
    languages: new ManyToManyField(Language, { relatedName: 'literatureTypes' }),

    toString: function() {
        return this.title + " by " + this.author;
    }
});
```

Note: Automatic creation of tables isn't supported yet. So for now you have to create the appropriate tables yourself.

Now create some instances:

```
var en = new Language({
    name: "English",
    shortcut: "en"
});
en.save();

var de = new Language({
    name: "German",
    shortcut: "de"
});
de.save();


var book = new Literature({
    title: "Alice's Adventures in Wonderland",
    author: "Lewis Carroll",
    orderId: 'AA',
    type: 1,
    languages: [en, de]
});
book.save();
```

Or make some queries:

```
Literature.objects.filter({ author__exact: "Lewis Carroll" }).all(processLiterature);

function processLiterature(instances) {
    for (var i = 0; i < instances.length; ++i) {
        instance = instances[i];
        instance.languages.set([en]);
        instance.author = "Llorrac Siwel";
        instance.save();
        document.body.innerHTML += instances[i].toString();
    }
}
```

```
</script>
```

CHAPTER 2

Internals

Class Field

Field of a model.

class **Field**(*params*)

Arguments

- **params** (*Object*) – Parameters for this field
- **params.primaryKey** (*Boolean*) – This field is the primary key.
- **params.unique** (*Boolean*) – This field is unique.
- **params.null** (*Boolean*) – This field can be null.
- **params.choices** (*Boolean*) – Array of [dbValue, displayValue] This field can hold exclusively values from choices.

See also:

Model

Methods

getParams

Returns the params object of this field.

`Field.prototype.getParams()`

toJs

Converts the value, that was fetched from a database query result, to its JavaScript equivalent. Callback is then called with the converted instance.

`Field.prototype.`**`toJs`**(*value*, *callback*)

Arguments

- **value** –
- **callback** –

toSql

Returns value as SQL formatted string

`Field.prototype.`**`toSql`**(*value*)

Arguments

- **value** –

validate

If value is valid returns true else returns an error msg string

`Field.prototype.`**`validate`**(*value*)

Arguments

- **value** –

Class Model

Meta Model Class. Used to define database models in an object-oriented way.

class `Model`(*modelDef*)

Arguments

- **modelDef** (*Object*) – The model definition, that is, field definitions and meta data

Returns (*Function*) Model instance constructor

Examples

```
// define a model
var TYPE_CHOICES = [
     [1, "Book"],
     [2, "Brochure"],
     [3, "Flyer"]
];

var Literature = new Model({
    Meta: {
        dbTable: "literature_types"
    },

    title: new CharField(),
    author: new CharField({ maxLength: 50 }),
    orderId: new CharField({ maxLength: 10, primaryKey: true }),
```

```
    type: new IntegerField({ choices: TYPE_CHOICES })
});

// use the model to create a new instance
var literature = new Literature({
    title: "Alice's Adventures in Wonderland",
    author: "Lewis Carroll",
    orderId: 'AA',
    type: 1
});
```

See also:

Field

Methods

getFields (*static*)

Returns a dictionary of field names and types

`Model.`**`getFields`**`()`

Returns (*Object*) { fieldName1: FieldType1, fieldName2: FieldType2, ... }

See also:

Field

save

Save method that every model instance has.

`Model.prototype.`**`save`**`(`*onComplete*`)`

Arguments

- **onComplete** (*Function*) – Callback when saving is finished. It is passed the saved model instance.

```
var Literature = new Model({ ... });

var literature = new Literature({ ... });

// save to database
literature.save();
```

validate

Validation method that every model instance has. Validates every field of the model.

`Model.prototype.`**`validate`**`()`

Returns (*Boolean|String*) true if every field is valid. If that is not the case the validation error message is returned.

Attributes

`Model.`**`objects`**

(*static*)

The default model manager to be used for querys

Class ModelManager

Model Manager class that provides the actual database operations for models

class **`ModelManager`** (*modelDef*)

Arguments

- **`modelDef`** (*`Object`*) – The model definition. See *Model*.

Methods

save

Saves a model instance to the db. Called by Model's save.

`ModelManager.prototype.`**`save`** (*modelInstance*, *onComplete*)

Arguments

- **`modelInstance`** – The model instance to save
- **`onComplete`** (*`Function`*) –

  callback function that is called when instance has been saved. Takes the saved instance as parameter.

See also:

Model

Class QuerySet

Class that represents a list of model instances that are retrieved by a database query.

class **`QuerySet`** (*modelDef*, *manager*)

Arguments

- **`modelDef`** (*`Object`*) – Definition of the model. See Model.
- **`manager`** (`ModelManager`) – The model manager of the model that is the base for this query set.

See also:

Model

Methods

all

Fetches all instances of the model which are in the db. This method actually hits the database and evaluates the query set.

`QuerySet.prototype.`**`all`**(*onComplete*)

Arguments

- **`onComplete`** –

  {Function} Callback that is called with a list of all instances.

clone

Creates a deep copy of this object (except cache).

`QuerySet.prototype.`**`clone`**()

convertLookups

Converts a lookup object into an SQL WHERE condition.

`QuerySet.prototype.`**`convertLookups`**(*queryObj*, *values*)

Arguments

- **`queryObj`** –
- **`values`** –

delete

Deletes all objects this query set represents.

`QuerySet.prototype.`**`delete`**(*onComplete*)

Arguments

- **`onComplete`** – {Function} Callback that is called when deletion is done. No arguments are passed.

exclude

Returns a QuerySet which represents all instances of the model which do NOT validate against queryObj. This QuerySet remains unchanged.

`QuerySet.prototype.`**`exclude`**(*queryObj*)

Arguments

- **`queryObj`** –

  {Object} field lookups or Q object.

filter

Returns a QuerySet which represents all instances of the model which validate against queryObj. This QuerySet remains unchanged.

`QuerySet.prototype.`**`filter`**(*queryObj*)

Arguments

- **`queryObj`** –

  {Object} field lookups or Q object.

get

Fetches the object that machtes the lookup parameters given by queryObj. The format of queryObj is the same as in filter(). If no or more than one result is found, an exception is thrown.

`QuerySet.prototype.`**`get`**(*queryObj*, *onComplete*)

Arguments

- **`queryObj`** –

  {Object} field lookups or Q object.

- **`onComplete`** –

  {Function} Callback that is called with the fetched instance.

orderBy

Returns a new QuerySet that is ordered by the given fields.

`QuerySet.prototype.`**`orderBy`**()

```
Entry.objects.orderBy('-pub_date', 'headline').all(...);
```

Class RelatedManager

Class that represents all instances of modelDef whose field named foreignKey has the value id. Also used for the reverse relation of ForeignKey.

class `RelatedManager`(*modelDef*, *relModelDef*, *foreignKey*, *id*, *joinModelDef*)

Arguments

- **`modelDef`** (*`Object`*) – Model definition. See *Model*.
- **`relModelDef`** (*`Object`*) – Model definition of the related model.
- **`foreignKey`** (*`String`*) – Name of the foreign key field.

If joinModelDef is provided then foreignKey is a column of the join table and the join table is joined with the modelDef table. If joinModelDef is not given, foreignKey is a column in the modelDef table.

param id Value of the foreignKey column, that the related model instances have in common

param Object joinModelDef Model definition of an intermediate join table. This is used for ManyToManyFields.

See also:

Model

Class SingleManager

Proxy for a single model instance, that should be loaded lazily. Used for ForeignKeys. The actual instance is loaded when get() is called.

class `SingleManager` (*modelDef*, *id*)

Arguments

- **`modelDef`** (*Object*) – Model definition of the base model. See *Model*.
- **`id`** – The value of the primary key of the instance this object represents.

Methods

get

Fetches the instance from the database and calls callback with it as argument.

`SingleManager.prototype.`**`get`** (*callback*)

Arguments

- **`callback`** (*Function*) – A callback function that is called when the instance that this proxy represents was fetched from the database.

Passes the instance as parameter to the callback.

set

Sets the instance

`SingleManager.prototype.`**`set`** (*instance*)

Arguments

- **`instance`** (Model) – The instance this proxy reprents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index