bedoon Documentation

Release 0.1

Riad Benguella

March 26, 2016

Contents

Bedoon is a nodejs/mongo no backend solution

Contents:

CHAPTER 1

Getting started

1.1 Requirements

- nodejs
- mongodb

1.2 Quick start

A simple bedoon application looks like this

```
// Loading Config
var Bedoon = require("bedoon");

var config = {
    db: 'mongodb://localhost/mydatabase',
    resources: {

        post: {
            type: 'document',
            schema: {
                attributes: {
                    title: String,
                    content: String
                }
            }
        }

    }
}

var bedoon = new Bedoon(config);
var port = 3700;
console.log("Listening on port " + port);
bedoon.app.listen(port);
```

And then run your application

```
$ node example.js
Listnening on port 3700
```

And this will automatically generate for you different json apis for handling posts

```
GET    /post         # to get all your posts
GET    /post/:id     # to get a post by id
POST   /post         # to create a new post
PUT    /post/:id     # to update an existing post
DELETE /post/:id     # to delete a post
```

CHAPTER 2

Resource Config

This page explains how to configure your resources (storage and apis)

2.1 Global config object

The bedoon configuration object must have a "db" attribute with the url of your mongo database and a resources attribute which contains all your resources config like this :

```
var config = {
    db: "mongodb://localhost/mydatabase",
    resources: {
        resource_name: {
            // resource config details
        }
    }
};
```

2.2 Resource config object

A resource have a type ("document" is the default type), and an schema that contains resource attributes, relations ... A basic resource configuration look like this :

```
resource_name: {
    type: 'document'
    schema: {
        attributes: {
            attribute1: String,
            attribute2: String
        }
    }
}
```

The above resource will create a mongo document called "resource_name" and each record will have two attributes, "attribute1" and "attribute2" and an auto-generated "_id" attribute. This will also create the following apis :

Find All

```
GET /resource_name # To retrieve all the records
```

response:

```
{
    status: "success",
    data: [
        {
            _id: "id",
            attribute1: "value",
            attribute2: "value"
        }

        // ...
    ]
}
```

Find One

```
GET /resource_name/:id # To retrieve a record by its id
```

response:

```
{
    status: "success",
    data: {
        _id: "id",
        attribute1: "value",
        attribute2: "value"
    }
}
```

Find Query

```
GET /resource_name?attribute1=value # To retrieve records with some filters
```

response:

```
{
    status: "success",
    data: [
        {
            _id: "id",
            attribute1: "value",
            attribute2: "value"
        }

        // ...
    ]
}
```

Create a record

```
POST /resource_name # To retrieve a record by its id
```

request body:

```
{
    attribute1: "value",
    attribute2: "value"
}
```

response:

```
{
    status: "success",
    data: {
        _id: "id",
        attribute1: "value",
        attribute2: "value"
    }
}
```

Update a record

```
PUT /resource_name/:id # To retrieve a record by its id
```

request body:

```
{
    _id: "id",
    attribute1: "value",
    attribute2: "value"
}
```

response:

```
{
    status: "success",
    data: {
        _id: "id",
        attribute1: "value",
        attribute2: "value"
    }
}
```

Delete a record

```
DELETE /resource_name/:id # To retrieve a record by its id
```

CHAPTER 3

Relations Config

Bedoon can handle different sort of relations between resources : has many relations and has one relations. Each relation type has different strategies that alter the storage and the object contents.

3.1 Id Strategy

With this strategy, the related resources are stored in separated documents, and the relation is materialized by referencing the id of the child object in the parent one (or an array of ids in the case of hasMany relations)

example

```
var config = {
    db: "mongodb://localhost/mydatabase"
    resources: {
        user: {
            type: "document",
            schema: {
                attributes: {
                    username: String,
                    name: String
                }
            }
        },

        post: {
            type: "document",
            schema: {
                attributes: {
                    title: String
                },
                hasMany: {
                    comments: {type: "id", target: "comment"},
                    author: {type: "id", target: "user"}
                }
            }
        },

        comment: {
            type: "document",
            schema: {
                attributes: {
                    message: String
```

```
                }
            }
        }
    }
}
```

With this configuration a post record will look like this

example

```
{
    title: "post title",
    comments: ["id_comment1", "id_comment2"],
    author: "id_user"
}
```

3.2 Embed Strategy

With this strategy, the child resources are stored inside the parent document, and the relation is materialized by embedding the child object in the parent one (or an array of child objects in the case of hasMany relations)

example

```
var config = {
    db: "mongodb://localhost/mydatabase"
    resources: {
        user: {
            type: "embed",
            schema: {
                attributes: {
                    username: String,
                    name: String
                }
            }
        },

        post: {
            type: "document",
            schema: {
                attributes: {
                    title: String
                },
                hasMany: {
                    comments: {type: "embed", target: "comment"},
                    author: {type: "embed", target: "user"}
                }
            }
        },

        comment: {
            type: "embed",
            schema: {
                attributes: {
                    message: String
                }
            }
        }
```

```
    }
}
```

With this configuration a post record will look like this.. code-block:: javascript

```
{
    title: "post title",
    comments: [
        {
            message: "message comment 1"
        },
        {
            message: "message comment 2"
        }
        // ...
    ],
    author: {
        username: "username value",
        name: "name"
    }
}
```

Note that the user and comment resource have a type “embed”, this avoid creating APIs for these resources.