Seatbelt Documentation

Release 0.0.1

Thomas Meadows

Jul 09, 2017

Contents

Seatbelt is a web framework designed to help simplify the creation and management of routes while adding typescript support. The included decorators make the framework extremely modular allowing you to use any nodejs server including express, restify, hapi, and koa along with potentially any orm such as waterline or bookshelf.

CHAPTER 1

Warning

Seatbelt is currently in alpha. This text will be erased once a full version is released.

CHAPTER 2

Basic Installation and usage

You are currently viewing the advanced documentation for seatbelt. To view basic installation and usage visit https://seatbelt.js.org.

CHAPTER 3

Table Of Contents

Install

1. Install the cli utilities

```
npm install @seatbelt/cli -g
```

2. Create a new project

```
seatbelt --new [dirname]
```

3. Pick a server and orm on creation of a new project
4. Enter the directory and run the index.js

```
cd [dirname]
node index.js
```

Usage

Models

You can creates models from any folder within your project. Currently waterline is supported but other orms should be supported in the future.

Waterline

Initialized waterline ORM plugin

In order to initialize waterline you will first have to add it to the plugins class of the server you are using the following format where your server file is located.

```
@DRestify()
export class Server implements IServer {
  public plugins = [
    waterlinePlugin({
      adapters: {
        memory: require('sails-memory')
      },
      connections: {
        default: {
          adapter: 'memory',
          schema: true
        }
      }
    })
  ];
}
```

Create Model In Waterline

```
import { DModel } from '@seatbelt/orm-waterline';

@DModel({
  connection: 'default',
  identity: 'test',
  attributes: {
    firstName: 'string',
    lastName: 'string'
  }
})
export class Test {}
```

Access Models From Waterline from a route or service

From a Route

```
import { DService, DRoute, DPolicy, DValidateRequest, IRoute, IController} from
↪'@seatbelt/core';

@DRoute({
  path: '/',
  type: ['GET', 'POST']
})
export class HomeRoute implements IRoute {
  public models: any;
  public controller (controller: IController) {
    return this.models.test.create(controller.params)
    .then(results => {
      return controller.send({ status: 200, json: controller });
```

```
    })
    .catch(err => {
      return controller.send({ status: 500, json: err });
    });
  }
}
```

Policies

Creating a Policy

Policies can be created in any folder of your project by using the format below. Params sent by the route call can be accessed from controller.params in the same way they can be accessed from routes.

```
import { DPolicy, IPolicy, IPolicyController } from '@seatbelt/core';

@DPolicy()
export class NewPolicy implements IPolicy {
  public controller (controller: IPolicyController) {
    console.log('policy working');
    return controller.next();
  }
}
```

Using a Policy

After creation the policy can be used simply by calling the Policy Decorator directly before declaring your controller.

```
import { DService, DRoute, DPolicy, DValidateRequest, IRoute, IController} from
↪'@seatbelt/core';

@DRoute({
  path: '/',
  type: 'GET'
})
export class HomeRoute implements IRoute {
  @DPolicy('NewPolicy')
  public controller (controller: IController) {
    this.services.Poke.poke();
    return controller.send({ status: 200, json: controller });
  }
}
```

Routes

Creating a Route

Routes can be created in any directory, using any file format. They will automatically be added to your server when the app runs. Params that are sent in either query parameters or in a post body are automatically added to the controls on the property params. The following example will display the params when the home route is queryed by get or post request.

```
import { DRoute, IRoute, IController} from '@seatbelt/core';

@DRoute({
  path: '/',
  type: ['GET', 'POST']
})
export class HomeRoute implements IRoute {
  public controller (controller: IController) {
    return controller.send({ status: 200, json: controller.params });
  }
}
```

Servers

Seatbelt is capable of using any server through decorator wrappers. Currently all major node servers are supported: Express, Restify, Koa, and Hapi.

Restify

Install

```
npm install @seatbelt/restify --save
```

Usage

```
import { DRestify } from '@seatbelt/server-restify';
import { IServer } from '@seatbelt/core';

@DRestify()
export class Server implements IServer {}
```

Express

Install

```
npm install @seatbelt/express --save
```

Usage

```
import { DExpress } from '@seatbelt/server-express';
import { IServer } from '@seatbelt/core';

@DExpress()
export class Server implements IServer {}
```

Hapi

Install

```
npm install @seatbelt/hapi --save
```

Usage

```
import { DHapi } from '@seatbelt/server-hapi';
import { IServer } from '@seatbelt/core';

@DHapi()
export class Server implements IServer {}
```

Koa

Install

```
npm install @seatbelt/koa --save
```

Usage

```
import { DKoa } from '@seatbelt/server-koa';
import { IServer } from '@seatbelt/core';

@DKoa()
export class Server implements IServer {}
```

Services

Creating a Service

Services can be created from any folder in your project.

```
import { DService } from '@seatbelt/core';

@DService()
export class Poke {
  public poke() {
    console.log('poke');
  }
}
```

Using Services

From a Route

```
import { DService } from '@seatbelt/core';

@DService() public services: any;
public controller (controller: any) {
  this.services.Poke.poke();
  return controller.send({ status: 200, json: controller });
}
```

From another Service

```
import { DService } from '@seatbelt/core';

@DService()
export class NewService {
  @DService() public services: any;
  public hi() {
    this.services.Poke.poke();
    console.log('hi');
  }
}
```

License