LoopBack

API Economy Made Easy With LoopBack 4

Biniam Admikew, Diana Lau, Ivy Ho, Janny Hou,
Kevin Delisle, Kyusung Shim, Taranveer Virk

7 November 2017

What is the API Economy?

API Economy

LoopBack

API Economy is about connecting data and services to create value for customers

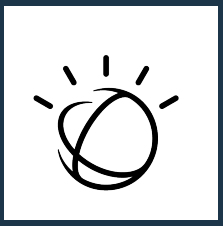
Create differentiating customer experiences

Extend your reach into a larger ecosystem

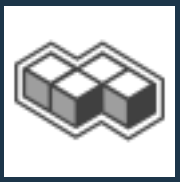
Deliver innovation to market faster

Some popular API's

LoopBack



stripe



Introduction to LoopBack

LoopBack

- Set up models and create REST APIs in minutes
- Model relation support
- Easy authentication and authorization setup
- Connect to backend data stores
- Run Ad-hoc queries
- Add-on components
- MIT Open-Source license

What's new in LoopBack 4

LoopBack

Complete Rewrite

ES2017

TypeScript

More Extensible

Much Simpler

Dependency Injection

LoopBack 4 Terminology

LoopBack

Context
Inversion of Control container that abstracts all state and dependencies of your Application

Application
A container for your application's components, controllers, etc.

Servers
An implementation of a transport protocol

Controllers
A class that implements operations defined by your REST API

Components
A packaged extension providing a controller, provider of a value, etc.

An idea ...

The idea

LoopBack

Prerequisites

Prerequisites

LoopBack

- NodeJS v8.0.0 or later
- TypeScript v.2.5 or later

Step 1: Project Setup

ibm.biz/cascon-loopback
Branch: step-01

Project Setup

LoopBack

Create new directory

We'll create a new project directory for our code. We can call it **cascon-diary**

npm init

Using terminal we will go to **cascon-diary** and run `npm init`

Accept all default answers except:

entry point: (index.js): `dist/index.js`

Start script

In our `package.json` we'll add the following under the `scripts` section to compile and start our project:

```
"start": "tsc && node ."
```

Installing Dependencies

Install LoopBack 4 and other dependencies by running the following in terminal:

```
npm i @loopback/core
@loopback/rest @loopback/context
```

```
npm i --save-dev @types/node
```

Project Setup

LoopBack

TypeScript Config

Create a file in the project directory called `tsconfig.json` with the following contents:

```
{
  "compilerOptions": {
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "module": "commonjs",
    "moduleResolution": "node",
    "target": "es2017",
    "outDir": "dist",
    "sourceMap": true,
    "declaration": true
  }
}
```

Step 2: Hello World

ibm.biz/cascon-loopback
Branch: step-02

NEW FILE: index.ts

LoopBack

```
import {Application} from '@loopback/core';
import {RestComponent, get} from '@loopback/rest';
class DiaryController {
  @get('/')
  helloWorld() {
    return 'Hello LoopBack';
  }
}
class DiaryApp extends Application {
  constructor() {
    super({
      components: [RestComponent],
    });
    this.controller(DiaryController);
  }
}
async function main() {
  const app = new DiaryApp();
  await app.start();
  console.log('App started');
}
main();
```

Running our App

LoopBack

Starting the App

In `terminal` run the following (in project directory):

```
npm start
```

Viewing the App

From your favorite browser visit `localhost:3000`
You should see the following message:
`Hello LoopBack`

Stopping the App

In `terminal` press the following keys together:

```
Ctrl + C
```

Step 3: Refactoring

ibm.biz/cascon-loopback
Branch: step-03

Refactoring: Controllers

LoopBack

New directories

Create a new directory in your project called `src`

Create a new directory in `src` called `controllers`

New file: `/src/controllers/diary.controller.ts`

- In `/src/controllers` create a new fille called `diary.controller.ts`
- Copy and paste the controller from `index.ts` into this file

NOTE: *We will need to import the dependencies and export the class as shown below*

```
import {get} from '@loopback/rest';

export class DiaryController {
  @get('/')
  helloWorld() {
    return 'Hello LoopBack';
  }
}
```

Refactoring: Application

LoopBack

New file: `/src/app.ts`

- In the `src` directory, we'll create a new file called `app.ts`
- We'll move our `DiaryApp` into this new file which should look as follows:

```
import {Application} from '@loopback/core';
import {RestComponent} from '@loopback/rest';
import {DiaryController} from './controllers/diary.controller';

export class DiaryApp extends Application {
  constructor() {
    super({
      components: [RestComponent],
    });

    this.controller(DiaryController);
  }
}
```

Refactoring: index.ts

LoopBack

Updated index.ts

- We've moved the application and controller class out of this file
- We'll import DiaryApp and start the server here. The file should look as follows:

```
import {DiaryApp} from './src/app';
import {RestServer} from '@loopback/rest';

(async function main() {
  const app = new DiaryApp();

  // Catch any startup errors
  try {
    await app.start();
  } catch (err) {
    console.error('Cannot start the application! ', err);
    process.exit(1);
  }

  console.log('App started');
})();
```

Running our App

LoopBack

Starting the App

In `terminal` run the following (in project directory):

```
npm start
```

Viewing the App

From your favorite browser visit `localhost:3000`
You should see the following message:
`Hello LoopBack`

Stopping the App

In `terminal` press the following keys together:

```
Ctrl + C
```

Step 4: Basic Diary Application

ibm.biz/cascon-loopback
Branch: step-04

Basic App: types.ts

LoopBack

New file: /src/types.ts

- Before we go about creating APIs, we're going to define a Diary type
- We're also going to define an OpenAPI Schema for request properties

```
import {SchemaObject} from '@loopback/openapi-spec';
export type Diary = {
  title: string;
  post: string;
  id: number;
};
export const diarySchema: SchemaObject = {
  properties: {
    title: {
      type: 'string',
    },
    post: {
      type: 'string',
    },
  },
};
```

Basic App

LoopBack

Install new dependency

- We used `@loopback/openapi-spec` but haven't installed it yet
- Install it by running the following command in `terminal`

```
npm i @loopback/openapi-spec
```

New Directory

- Create a new directory in `src` called `datastores`

New File: `/src/datastores/diary.datastore.ts`

- Create a new file in the `datastores` directory called `diary.datastore.ts`
- This is where we'll create a datastore to store diary entries

New File: /src/datastores/diary.datastore.ts

LoopBack

Writing a basic DataSource – store and retrieve Diary entries

```
import {Diary} from '../types';
export class DiaryDataStore {
  diaries: {[key: number]: Diary};
  id: number;
  constructor() {
    this.diaries = {}; // We will store our entries in an object
    this.id = 1; // We'll increment this to create id for new entries
  }
  getDiaries(): Diary[] {
    return Object.values(this.diaries);
  }
  getDiaryById(id: number): Diary {
    return this.diaries[id];
  }
  createDiary(diary: Diary): Diary {
    diary.id = this.id;
    this.diaries[this.id] = diary;
    this.id++;
    return diary;
  }
}
```

Using the DataStore

LoopBack

Binding the DataStore

- Since it's a memory store, we'll want a single instance in the entire application
- We'll make the following changes in `/src/app.ts`

- Add the following at the top of the file

```
import {BindingScope} from '@loopback/context';
import {DiaryDataStore} from './datastores/diary.datastore';
```

- In the constructor, add the following line to bind DiaryDataStore to `'datastores.diary'`

```
this.bind('datastores.diary')
.toClass(DiaryDataStore)
.inScope(BindingScope.SINGLETON);
```

Basic Diary Controller

LoopBack

Clean start

- We can start by deleting the existing contents of `/src/controllers/diary.controllers.ts`

Start with imports

- We'll start with adding the following imports (some should be familiar)

```
import {get, post, param, operation, RestBindings} from '@loopback/rest';
import {inject} from '@loopback/core';
import {ServerResponse} from 'http';
import {Diary, diarySchema} from '../types';
import {DiaryDataStore} from '../datastores/diary.datastore';
```

Diary Controller: Constructor

LoopBack

Constructor

- We'll use the constructor to inject in the Response Object and DiaryDataStore into the Controller

```
export class DiaryController {
  constructor(
    @inject('datastores.diary') public diaryStore: DiaryDataStore,
    @inject(RestBindings.Http.RESPONSE) public res: ServerResponse
  ) {}
}
```

Diary Controller: Methods

LoopBack

Adding basic GET and POST methods

```
@get('/')
getDiaries(): Diary[] {
  return this.diaryStore.getDiaries();
}
@get('/{id}')
@param.path.number('id')
getDiaryById(id: number): Diary {
  return this.diaryStore.getDiaryById(id);
}
@post('/')
@param.body('diary', diarySchema)
createDiary(diary: Diary): Diary {
  return this.diaryStore.createDiary(diary);
}
@operation('OPTIONS', '/') // For CORS pre-flight requests
optionsHeader() {
  this.res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS');
  this.res.setHeader('Access-Control-Allow-Headers',
    'Content-Type, Access-Control-Allow-Headers');
}
```

Running our App

LoopBack

Starting the App

In `terminal` run the following (in project directory):

```
npm start
```

Viewing the App

Try the various APIs from your favorite browser by visiting:

```
localhost:3000/swagger-ui
```

Stopping the App

In `terminal` press the following keys together:

```
Ctrl + C
```

Step 5: Making it Cognitive

ibm.biz/cascon-loopback
Branch: step-05

Making it Cognitive

LoopBack

Install new Dependencies

- Run the following command in terminal to install new dependencies

```
npm i watson-developer-cloud bluebird @types/watson-developer-cloud @types/bluebird
```

Update: /src/types.ts

- Add a new type called Tone to the file

```
export type Tone = {
  score: number;
  tone_id: string;
  tone_name: string;
};
```

Update Diary type

- Add the following property to the Diary type

```
tones: Tone[];
```

```
export type Diary = {
  title: string;
  post: string;
  id: number;
  tones: Tone[];
};
```

Making it Cognitive: ToneAnalyzer

LoopBack

Getting Credentials

- For CASCON attendees, find a file called `creds.ts` on your desktop. Move it to your project directory

- For anyone else, you can visit https://console.bluemix.net/ to get a free account for **Watson ToneAnalyzer** service
- Once you have a free account, create a file called `cred.ts` in your project directory
- Past the following into the file and put in your credentials

```
export const creds = {
  url: 'https://gateway.watsonplatform.net/tone-analyzer/api',
  username: '<YOUR USERNAME>',
  password: '<YOUR PASSWORD>',
  version: '<LATEST WATSON VERSION DATE (ex: 2017-09-21)>'
};
```

Using the Credentials

LoopBack

Bind the credentials

- We'll read the credentials from `creds.ts` and bind them for dependency injection
- Import creds at the top of `app.ts` by adding the following line:

```
import {creds} from '../creds';
```

- Bind the creds by adding the following lines in the constructor

```
this.bind('tone_analyzer.creds').to(creds);
```

Controller: /src/controllers/diary.controller.ts

LoopBack

Adding new imports

- Add the following imports at the top of the file

```
import {Promise} from 'bluebird';
import {ToneAnalyzerV3} from 'watson-developer-cloud';
```

Add new property

- Add a new property called `tone_analyzer` to `DiaryController`

```
export class DiaryController {
  // Add property line here
  tone_analyzer: any;
}
```

Controller: Constructor

LoopBack

Updated Constructor

- We'll be injecting credentials for ToneAnalyzer into the constructor and initializing the service
- The service is callback based, to make it easier to use we'll Promisify it as well in the constructor

```
constructor(
  @inject('datastores.diary') public diaryStore: DiaryDataStore,
  @inject(RestBindings.Http.RESPONSE) public res: ServerResponse,
  @inject('tone_analyzer.creds') creds: any,
) {
    this.tone_analyzer = new ToneAnalyzerV3({
      username: creds.username,
      password: creds.password,
      version_date: creds.version,
  });
  this.tone_analyzer.tone = Promise.promisify(this.tone_analyzer.tone);
}
```

Cognitive Controller

LoopBack

Saving Tone Information

- Update `createDiary` method to get a tone analysis and saving that as part of the Diary entry

```
@post('/')
@param.body('diary', diarySchema)
async createDiary(diary: Diary) {
const tone = await this.tone_analyzer.tone({ text: diary.post });
diary.tones = tone.document_tone.tones;

return this.diaryStore.createDiary(diary);
}
```

Cognitive Controller

LoopBack

Retrieving Diary Entries by Tone

- Update `getDiaries` to accept an option tone query parameter to filter Diary entries

```
@get('/')
@param.query.string('tone')
getDiaries(tone?: string): Diary[] {
  const diaries = this.diaryStore.getDiaries();
  if (!tone) return diaries;
  tone = tone.toLowerCase();
  let result: Diary[];
  for (const diary of diaries) {
    for (const diaryTone of diary.tones) {
      if (diaryTone.tone_name.toLowerCase() === tone) {
        result.push(diary);
        break;
      }
    }
  }
  return result;
}
```

Running our App

LoopBack

Starting the App

In `terminal` run the following (in project directory):

```
npm start
```

Viewing the App

Try the various APIs from your favorite browser by visiting:

```
localhost:3000/swagger-ui
```

Stopping the App

In `terminal` press the following keys together:

```
Ctrl + C
```

Done!

ibm.biz/cascon-loopback

Next Steps

LoopBack

Extensions!

https://github.com/strongloop/loopback-next/issues/509

Calling for Contributors on LB Extensions

https://github.com/strongloop/loopback-next/issues/647

Do we want a new Juggler?

https://github.com/strongloop/loopback-next/issues/537

Loopback4-extension-starter

https://github.com/strongloop/loopback4-extension-starter

Loopback4-example-getting-started

https://github.com/strongloop/loopback4-example-getting-started

LoopBack

Thank you

Credits

LoopBack

- Edit by Gregor Cresnar from Noun Project
- Fullscreen by Deemak Daksina S from Noun Project
- Simplify by Chris Homan from Noun Project
- Injection by Surya from Noun Project
- Diary by Rajive from Noun Project
- Silence by Alex Muravev from Noun Project
- Smile by Alex Muravev from Noun Project
- Cry by Alex Muravev from Noun Project
- Mad by Alex Muravev from Noun Project
- Worry by Alex Muravev from Noun Project
- Confusion by Alex Muravev from Noun Project
- Congratulations by Sewon Park from Noun Project