LoopBack 3

Install LoopBack

To install the LoopBack CLI tool, enter this command:

npm install -g loopback-cli

## Install LoopBack tools

You have two options for LoopBack tools:

* [**LoopBack CLI tool**](https://loopback.io/doc/en/lb3/Installation#install-loopback-cli-tool)
* [**IBM API Connect v5 developer toolkit**](https://loopback.io/doc/en/lb3/Installation#install-ibm-api-connect-developer-toolkit)

You can also use the legacy StrongLoop CLI tool slc.

### Install IBM API Connect v5 developer toolkit

[IBM API Connect](https://developer.ibm.com/apiconnect/) is an end-to-end API management solution that uses LoopBack to create APIs, and provides integrated build and deployment tools:

* **Integrated experience across the entire API lifecycle**, including API and micro-service creation in Node.js and Java.
* **Self-service access to APIs** with built-in developer portals and social collaboration tools.
* **Unified management and orchestration of Node.js and Java** for deployment on-premises and in IBM Cloud.
* **Built-in security and gateway policies** with extensive security options and governance policies.

To install IBM API Connect v5 developer toolkit:

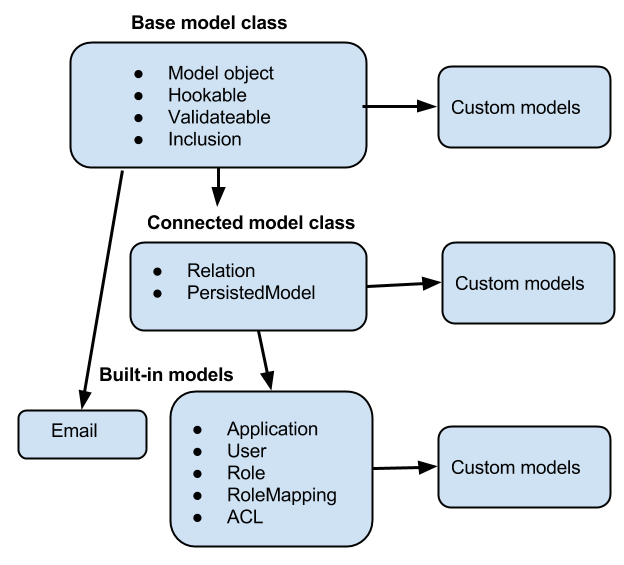
$ npm install -g apiconnect

LoopBack core concepts

## Models

Models are at the heart of LoopBack, and represent back-end data sources such as databases or other back end services (REST, SOAP, and so on). LoopBack models are JavaScript objects with both Node and REST APIs.

**A key powerful feature of LoopBack is that when you define a model it automatically comes with a predefined REST API with a full set of create, read, update, and delete operations.**



The [Basic model object](https://loopback.io/doc/en/lb3/Basic-model-object.html) has methods for adding [hooks](https://loopback.io/doc/en/lb3/Operation-hooks.html) and for [validating data](https://loopback.io/doc/en/lb3/Validating-model-data.html). Other model objects all “inherit from” it. Models have an inheritance hierarchy, as shown at right: When you attach a model to a persistent data source it becomes a [connected model](https://loopback.io/doc/en/lb3/Connected-model-object.html) with create, retrieve, update, and delete operations. LoopBack’s built-in models inherit from it.

### Custom models

You can [define your own custom models](https://loopback.io/doc/en/lb3/Creating-models.html) specific to your application.  You can make your custom models [extend built-in models](https://loopback.io/doc/en/lb3/Extending-built-in-models.html) to build on the predefined functionality of [User](https://apidocs.loopback.io/loopback/#user),  [Application](https://apidocs.loopback.io/loopback/" \l "application), and other built-in models.

You can create LoopBack models in various ways, depending on what kind of data source the model is based on. You can create models:

* [With the LoopBack model generator](https://loopback.io/doc/en/lb3/Using-the-model-generator.html).
* [From an existing relational database](https://loopback.io/doc/en/lb3/Discovering-models-from-relational-databases.html) using *model discovery*. Then you can keep your model synchronized with the database using LoopBack’s [schema / model synchronization](https://loopback.io/doc/en/lb3/Creating-a-database-schema-from-models.html) API.
* [By instance introspection](https://loopback.io/doc/en/lb3/Creating-models-from-unstructured-data.html) for free-form data in NoSQL databases or REST APIs.

All three of these methods create a [Model definition JSON file](https://loopback.io/doc/en/lb3/Model-definition-JSON-file.html) that defines your model in LoopBack, by convention in a LoopBack project’s common/models directory; for example, common/models/account.json.

You can also create and customize models programmatically using the [LoopBack API](http://apidocs.loopback.io/loopback/" \l "loopback-createmodel), or by manually editing the [model definition JSON file](https://loopback.io/doc/en/lb3/Model-definition-JSON-file.html). In most cases, you shouldn’t need to use those techniques to create models, but you generally will use them to modify and customize models.

**Note:**

The [Model definition JSON file](https://loopback.io/doc/en/lb3/Model-definition-JSON-file.html) includes an idInjection property that indicates whether LoopBack automatically adds a unique id property to a model. For a model connected to a database, the id property corresponds to the primary key.  See [ID properties](https://loopback.io/doc/en/lb3/Model-definition-JSON-file.html#id-properties) for more information.

### Model relations

You can express [relationships between models](https://loopback.io/doc/en/lb3/Creating-model-relations.html), such as  [BelongsTo](https://loopback.io/doc/en/lb3/BelongsTo-relations.html),  [HasMany](https://loopback.io/doc/en/lb3/HasMany-relations.html), and  [HasAndBelongsToMany](https://loopback.io/doc/en/lb3/HasAndBelongsToMany-relations.html).

### Model create, retrieve, update, and delete operations

When you connect a model to a persistent data source such as a database, it becomes a [*connected model*](https://loopback.io/doc/en/lb3/Connected-model-object.html)  with a full set of create, read, update, and delete operations from the [PersistedModel](http://apidocs.loopback.io/loopback/" \l "persistedmodel) class:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **REST** | **LoopBack model method (Node API)\*** | **Corresponding SQL Operation** |
| Create | [PUT /modelName](https://loopback.io/doc/en/lb3/PersistedModel-REST-API.html#create-model-instance)  [POST /modelName](https://loopback.io/doc/en/lb3/PersistedModel-REST-API.html#update--insert-instance) | [create()](http://apidocs.loopback.io/loopback/#persistedmodel-create)\* | INSERT |
| Read (Retrieve) | [GET /modelName?filter=...](https://loopback.io/doc/en/lb3/PersistedModel-REST-API.html#find-matching-instances) | [find()](http://apidocs.loopback.io/loopback/#persistedmodel-find)\* | SELECT |
| Update (Modify) | [PUT /modelName](https://loopback.io/doc/en/lb3/PersistedModel-REST-API.html#update-model-instance-attributes) | [updateAll()](http://apidocs.loopback.io/loopback/#persistedmodel-updateall)\* | UPDATE |
| Delete (Destroy) | [DELETE /modelName/modelID](https://loopback.io/doc/en/lb3/PersistedModel-REST-API.html#delete-model-instance) | [destroyById()](http://apidocs.loopback.io/loopback/#persistedmodel-destroybyid)\* | DELETE |

## Application logic

You can add custom application logic in several ways; you can:

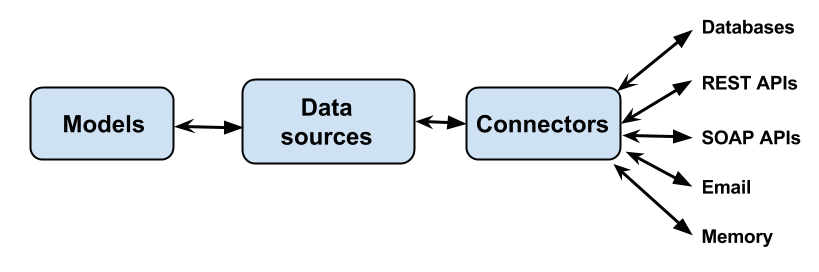
* [Add application logic to models](https://loopback.io/doc/en/lb3/Adding-logic-to-models.html) through [remote methods](https://loopback.io/doc/en/lb3/Remote-methods.html) (custom REST endpoints), [remote hooks](https://loopback.io/doc/en/lb3/Remote-hooks.html) that are triggered by remote methods, and [operation hooks](https://loopback.io/doc/en/lb3/Operation-hooks.html) that are triggered by model create, retrieve, update, and delete methods.
* Add boot scripts that run when the application starts.
* Define custom [middleware](https://loopback.io/doc/en/lb3/Defining-middleware.html), similar to Express middleware.

You can add code to [validate data](https://loopback.io/doc/en/lb3/Validating-model-data.html) before saving it to the model and back-end data store.

### Middleware phases

Middleware refers to functions executed when HTTP requests are made to REST endpoints. Since LoopBack is based on [Express](http://expressjs.com/), LoopBack middleware is the same as [Express middleware](http://expressjs.com/en/guide/using-middleware.html). However, LoopBack adds the concept of phases, to clearly define the order in which middleware is called. Using phases helps to avoid ordering issues that can occur with standard Express middleware.

## Data sources and connectors



LoopBack generalizes backend services such as databases, REST APIs, SOAP web services, and storage services as data sources.

Data sources are backed by connectors that then communicate directly with the database or other back-end service. Applications don’t use connectors directly, rather they go through data sources using the [DataSource](https://apidocs.loopback.io/loopback-datasource-juggler/" \l "datasource) and  [PersistedModel](http://apidocs.loopback.io/loopback/" \l "persistedmodel) APIs.

# Routing

## Summary of Express routing

For those not familiar with routing in Express, here are some key points:

* Routing refers to the rules for capturing requests to the server, and the subsequent passing through and handling of requests through a chain of middleware functions.
* A middleware function accepts three objects, the request object ([req](http://expressjs.com/4x/api.html" \l "req)), the response object ([res](http://expressjs.com/4x/api.html#res)), and the next middleware in the chain ([next](http://expressjs.com/guide/writing-middleware.html)); in that order.
* You load middleware either using [app.use()](http://expressjs.com/4x/api.html" \l "app.use) or by assigning it as the callback function of a [route definition](http://expressjs.com/4x/api.html#app.METHOD).
* Multiple middleware can be matched to handle the requests to a [route](http://expressjs.com/4x/api.html#app.METHOD), these matched middleware make up the middleware chain for the request. The request will pass through each middleware in the order they were loaded, unless one of the middleware in the chain terminates the propagation.
* Any middleware in the chain may terminate the request propagation by sending a response back to the client.
* A middleware can send the response to the request using one of the response methods in the [response object](http://expressjs.com/4x/api.html#res) or pass on the request to the next middleware by calling next().
* If a middleware sends the server response, conventionally the request does not propagate further in the middleware chain. Any call to next() will likely result in an error.
* A middleware function can also take four arguments. In this case, it is an [error handling middleware](http://expressjs.com/guide/error-handling.html). The parameters to the function in their order are: the error object ([err](http://expressjs.com/guide/error-handling.html)), the request object ([req](http://expressjs.com/4x/api.html" \l "req)), the response object ([res](http://expressjs.com/4x/api.html#res)), and the next middleware in the chain ([next](http://expressjs.com/guide/writing-middleware.html)).

For more details about routing in Express, see <http://expressjs.com/guide/routing.html>.

## LoopBack routing

### LoopBack middleware chain

The LoopBack middleware chain is composed of middleware added to the chain in the following order, and the request passes through them in the given order.

**Note:**

Except 14, 15, and 16, the listed items refer to [LoopBack middleware](https://loopback.io/doc/en/lb3/Defining-middleware.html) phases.

1. initial:before
2. initial
3. initial:after
4. session:before
5. session
6. session:after
7. auth:before
8. auth
9. auth:after
10. parse:before
11. parse
12. parse:after
13. routes:before
14. [Express middleware](http://expressjs.com/guide/writing-middleware.html)
15. [Components](https://loopback.io/doc/en/lb3/LoopBack-components.html)
16. [Boot scripts](https://loopback.io/doc/en/lb3/Defining-boot-scripts.html)
17. routes
18. routes:after
19. files:before
20. files
21. files:after
22. final:before
23. final
24. final:after

A middleware loaded earlier in the chain gets the prior opportunity to handle the request. If it happens to send a response or fail to call next(), the request will terminate at it, and not propagate any further.

As evident from the list above, LoopBack takes control of the loading order of the middleware in the app, and prioritizes its relevant middleware over those loaded using Express’ interface app.use(), components, or boot scripts.

**Note:**

If you add middleware on the route or route:after phase, it will not execute after the route is matched. Instead, it will be ignored because the route was already matched.

The middleware to be loaded during the middleware phases are configured in the [middleware.json](https://loopback.io/doc/en/lb3/middleware.json.html) file of the app. The order of the phases in the file are semantic, and cannot be listed randomly.

LoopBack also supports [custom phases](https://loopback.io/doc/en/lb3/Defining-middleware.html#adding-a-custom-phase). Custom phases can be defined in any position of the middleware chain, and may be used to prioritize over LoopBack’s built-in middleware phases.

Create a simple API

## Create new application

If using loopback-cli:

$ lb

## Create models

Using StrongLoop tools:

$ lb model

It will ask if you want to attach the model to any data sources that have already been defined.

At this point, only the default in-memory data source is available.  Press **Enter** to select it:

...

[?] Select the data-source to attach CoffeeShop to: (Use arrow keys)

❯ db (memory)

Then the generator will prompt you for the base class to use for the model.  Since you will eventually connect this model to a persistent data source in a database, press down-arrow to choose **PersistedModel**, then press**Enter**:

[?] Select model's base class: (Use arrow keys)

Model

❯ PersistedModel

ACL

AccessToken

Application

Change

Checkpoint

[PersistedModel](http://apidocs.loopback.io/loopback/#persistedmodel)is the base object for all models connected to a persistent data source such as a database.  See [LoopBack core concepts](https://loopback.io/doc/en/lb3/LoopBack-core-concepts) for an overview of the model inheritance hierarchy.

Next, you’ll be asked whether you want to create the model on the server only or in the /common directory, where it can potentially be used by both server and [client LoopBack APIs](https://loopback.io/doc/en/lb3/LoopBack-in-the-client).

? Common model or server only?

❯ common

server

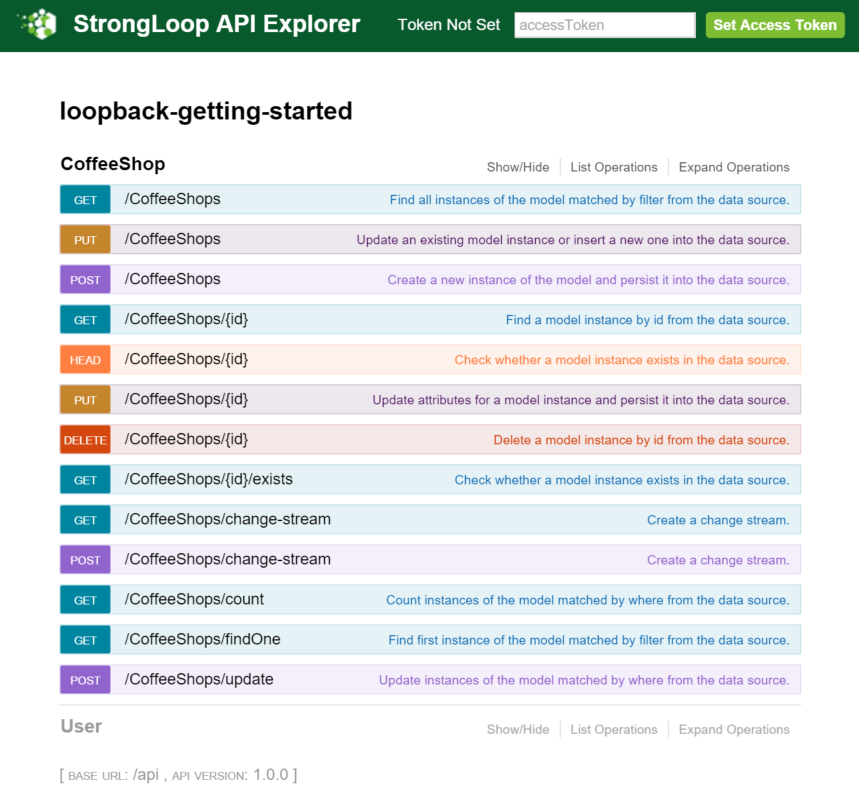
## Run the application

Start the application:

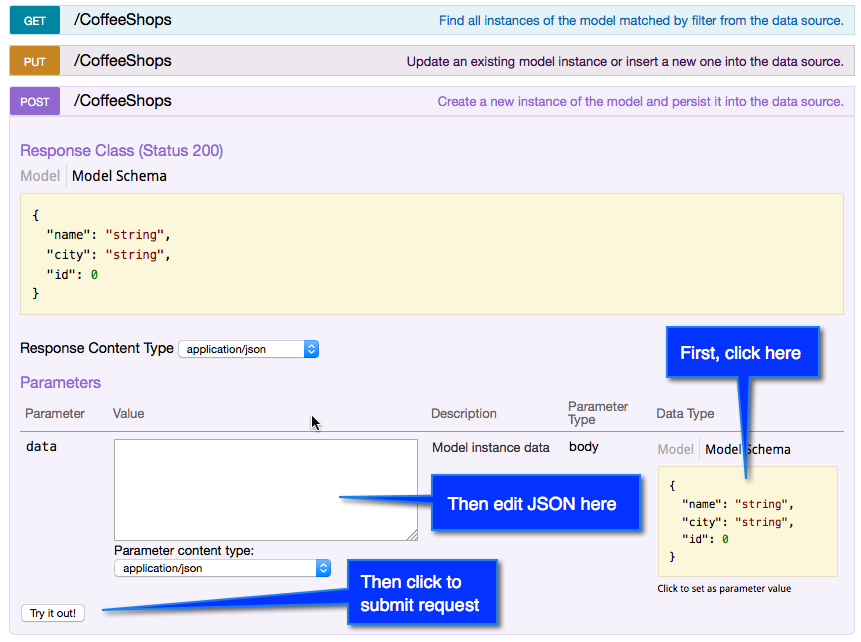
$ node .

## Exploring the CoffeeShop model

Click on **CoffeeShops** to show all its API endpoints:



Click on  **POST  /CoffeeShops**   **Create a new instance of the model and persist it into the data source**to expand that operation:



Add some text for the name property.  You don’t have to put anything for the id property, because LoopBack will automatically manage it to ensure there is always a unique ID for each model instance.

{

"name": "My Coffee Shop",

"id": 0

}

Then click the **Try it out!** button.

Connect your API to a data source

## Add a data source

Now you’re going to define a data source using the [Data source generator](https://loopback.io/doc/en/lb3/Data-source-generator):

$ lb datasource

The generator will prompt you to name the data source:

[?] Enter the data-source name:

Enter **mysqlDs** and hit **Enter**.

Next, the generator will prompt you for the type of data source:

? Select the connector for mysqlDs: (Use arrow keys)

❯ In-memory db (supported by StrongLoop)

IBM DB2 (supported by StrongLoop)

IBM DashDB (supported by StrongLoop)

IBM MQ Light (supported by StrongLoop)

IBM Cloudant DB (supported by StrongLoop)

IBM DB2 for z/OS (supported by StrongLoop)

MongoDB (supported by StrongLoop)

(Move up and down to reveal more choices)

Press the down-arrow key to highlight **MySQL (supported by StrongLoop)**, then hit **Enter**.

Then the tool will prompt you for the data source configuration settings. For MySQL, you can either enter all the settings in URL format, or individually.

Connector-specific configuration:

? Connection String url to override other settings (eg: mysql://user:pass@host/db):

Press **Enter** to skip the URL connection string, since you’ll enter the settings individually.

**Important:** If you have a MySQL database server that you can use, please use it. Create a new database called “getting\_started.” If you wish, you can use a different database name. Just make sure the mysqlDs.database property in datasources.json matches it (see below).

To use the StrongLoop MySQL server enter the settings shown below. To use your own MySQL server, enter the hostname, port number, and login credentials for your server.

? host: demo.strongloop.com

? port: 3306

? user: demo

? password: L00pBack

? database: getting\_started

? Install loopback-connector-mysql@^2.2 Yes

When the tool prompts you to install the connector, hit **Enter** to make the tool run npm install loopback-connector-mysql --save. The tool will also add the data source definition to the server/datasources.json file, which will look as shown below.  Notice the “mysqlDs” data source you just added, as well as in-memory data source named “db,” which is there by default.

**/server/datasources.json**

{

"db": {

"name": "db",

"connector": "memory"

},

"mysqlDs": {

"host": "demo.strongloop.com",

"port": 3306,

"url": "",

"database": "getting\_started",

"password": "L00pBack",

"name": "mysqlDs",

"user": "demo",

"connector": "mysql"

}

}

## Connect CoffeeShop model to MySQL

Now you’ve created a MySQL data source and you have a CoffeeShop model; you just need to connect them.  LoopBack applications use the [model-config.json](https://loopback.io/doc/en/lb3/model-config.json) file to link models to data sources.  Edit /server/model-config.json and look for the CoffeeShop entry:

**/server/model-config.json**

...

"CoffeeShop": {

"dataSource": "db",

"public": **true**

}

...

Change the dataSource property from db to mysqlDs.  This attaches the CoffeeShop model to the MySQL datasource you just created and configured:

**/server/model-config.json**

...

"CoffeeShop": {

"dataSource": "mysqlDs",

"public": **true**

}