



finding the beginners mind highly-extensible, open-source Node.js framework

<https://loopback.io/>

GROUND UP SERIES



LOOPBACK.IO

Syed Awase Khirni

Syed Awase earned his PhD from University of Zurich in GIS, supported by EU V Framework Scholarship from SPIRIT Project (www.geo-spirit.org). He currently provides consulting services through his startup www.territorialprescience.com and www.sycliq.com. He empowers the ecosystem by sharing his technical skills worldwide, since 2008. He provides training in Java Technology Stack, .Net Technology Stack, R, DataScience, Client Side frameworks (Angular, KnockOut, Aurelia, Vue, Ember, Backbone etc..), Node Stack, Machine Learning, Python Stack, Php Stack.

www.territorialprescience.com

www.sycliq.com



Terms of Use

You shall not circulate these slides without written permission from **Territorial Prescience Research I Pvt Ltd.**

If you use any material, graphics or code or notes from these slides, you shall seek written permission from TPRI and acknowledge the author Dr. Syed Awase Khirni

If you have not received this material, post-training session, you shall destroy it immediately and not use it for unauthorized usage of the material. If any of the material, that has been shared is further used for any unauthorized training by the recipient, he shall be liable to be prosecuted for the damages. Any supporting material that has been provided by the author, shall not be used directly or indirectly without permission.

If this material, has been shared to any organization prior to the training and the organization does not award the contract to TPRI, it should not use the training material internally. If by any chance, the organization is using this training material without written permission, the organization is liable to pay for the damages to TPRI and is subjected to legal action, jurisdiction being Bangalore. It shall also pay for any expenses, legal, recovery and all applicable damages and costs incurred by TPRI.

TPRI has right to claim damages ranging from USD 50000 to USD 10,0000 dollars as damages, for unauthorized usage.

Any organization, which does not intend to go ahead with training or does not agree with the terms and conditions, should destroy the material from its network immediately. The burden of proof lies on the client, with whom this material has been shared.

Recovery of the damages and all expenses incurred including legal fees will be born by the client organization/candidate/party, which has violated these terms and conditions.

Only candidates who have attended the training session in person from Dr. Syed Awase Khirni, TPRI are entitled to hold this training material. They cannot further circulate it, or use it or morph it, or change it to provide trainings. This training material cannot be used by any other candidates other than the registered individuals for the class room based session.

TPRI reserves all the rights to this material and code plays and right to modify them as and when it deems fit.

If you agree with the terms and conditions, please go ahead with using the training material. Else please close and destroy the slide and inform TPRI immediately.



Slide Version Updates

Please read terms of use for authorized access

Original Series

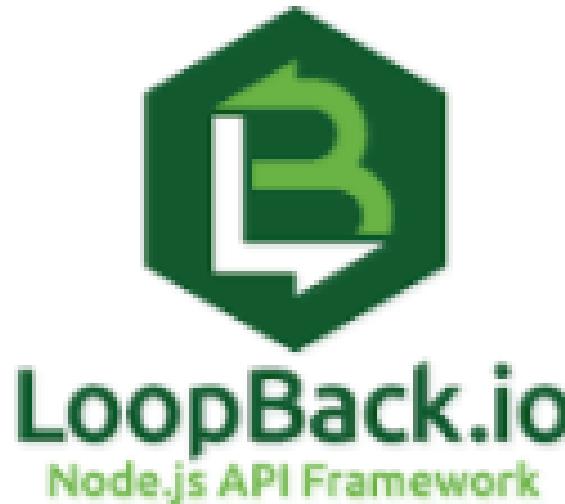
Last Updated	Version	Release Date	Updated by	Code Plays Done @



<https://docs.strongloop.com/display/ALLDOCS/PDF+version>

LoopBack.io: Introduction

SYED AWASE KHIRNI



<https://loopback.io/>

A highly extensible, open-source Node.js framework that enables you to create dynamic end-to-end REST APIs with little or no coding. **Open source with MIT Licensing agreement.**



Application Development and Service Model Evolution

Please read terms of use for authorized access

Original Series

**Application
Server**

**AppServer
IaaS**

PaaS

**mBaaS
Mobile
backend as
a service
(MBaaS),**

API Server

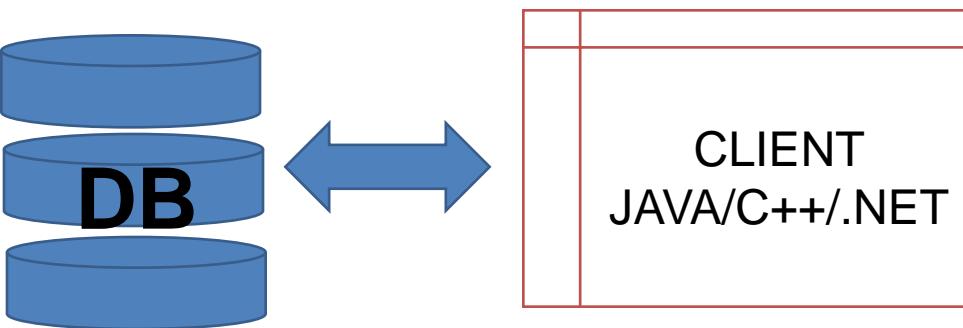


A Business process demands cross-domain collaboration

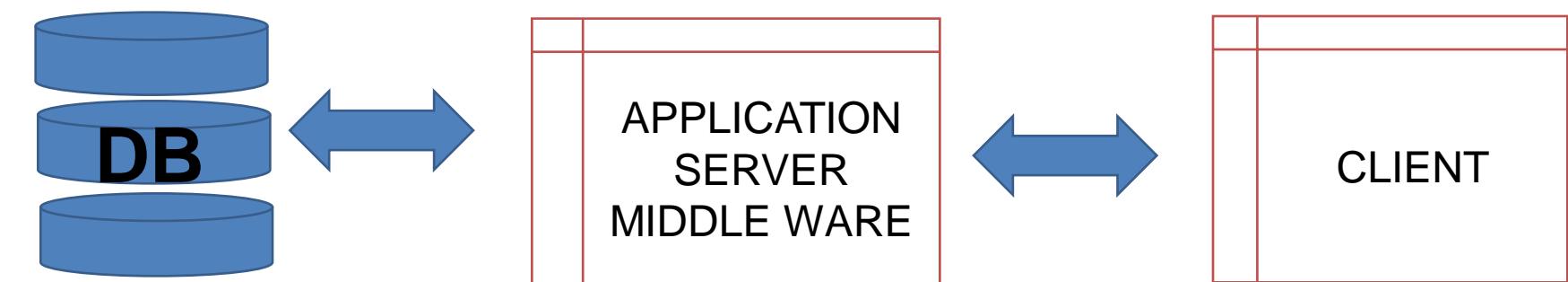
**Technology required to support
cross-channel/platform business logic**



2- Tier Architecture



3- Tier Architecture



Please read terms of use for authorized access

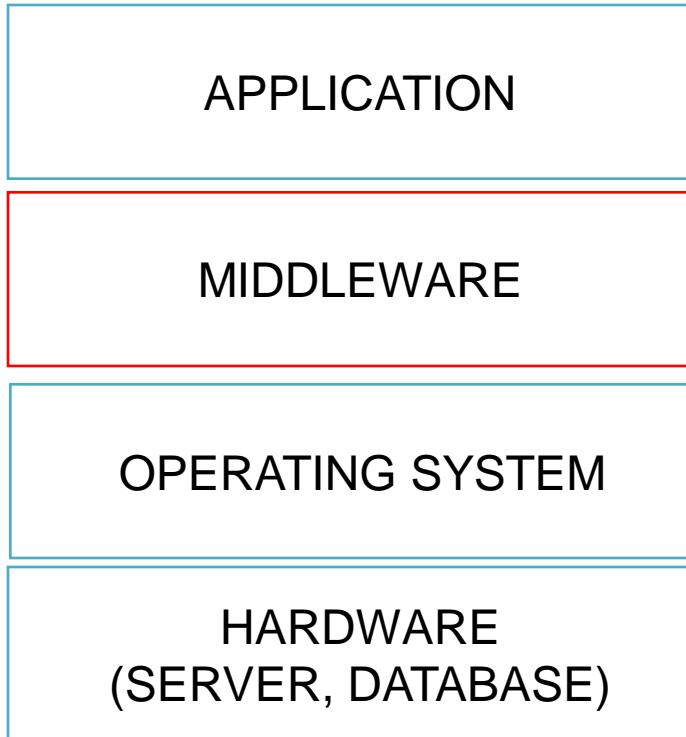
Original Series



Middleware

Please read terms of use for authorized access

Original Series

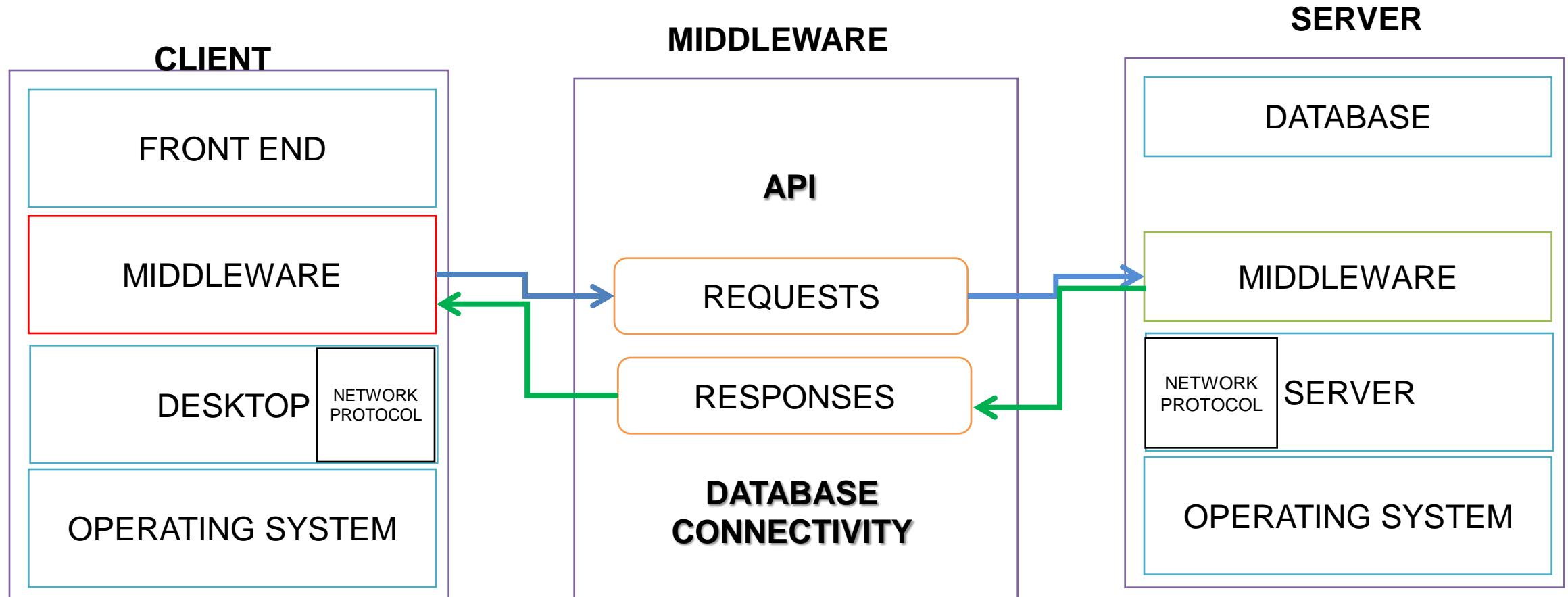


Middleware is any software that acts like a “glue” between an application and its network. It controls the flow of information between an application and the server, database and the operating system.



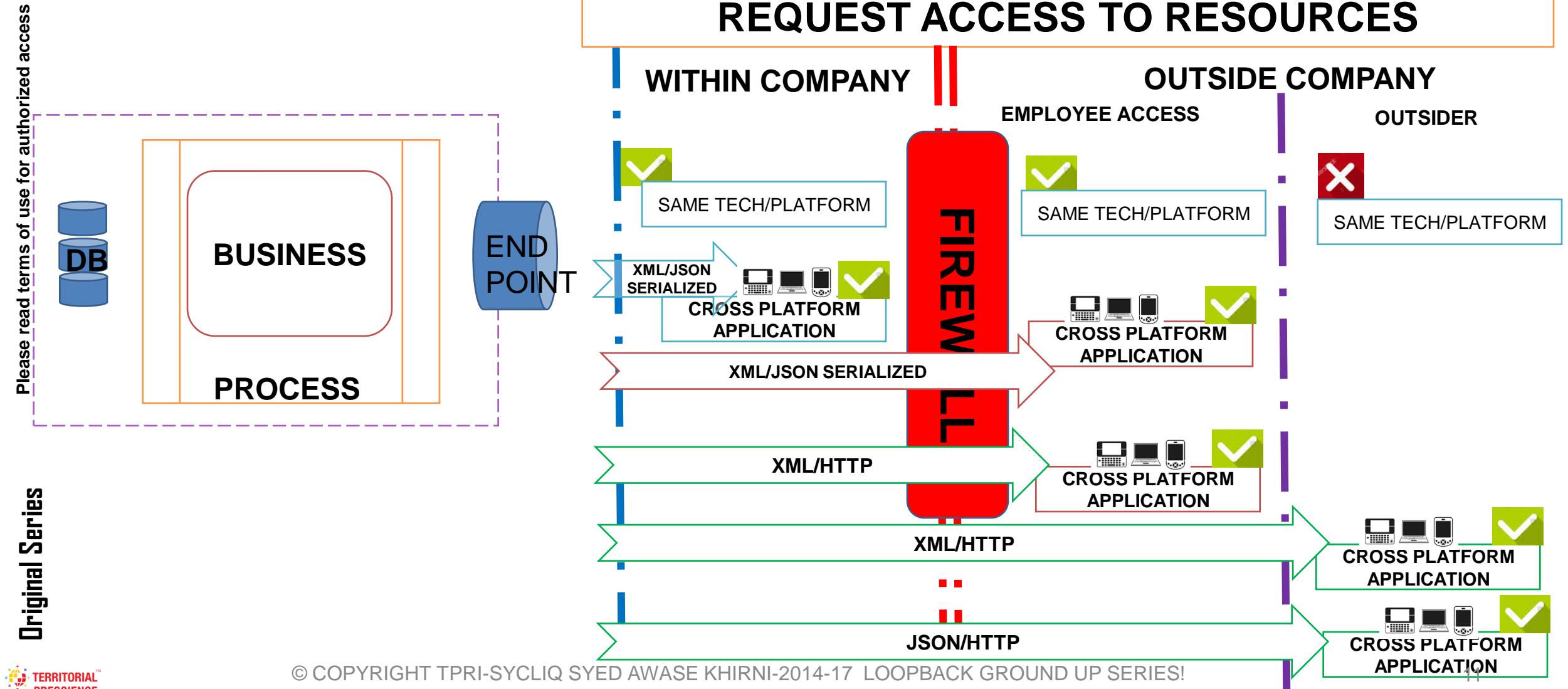
Middleware

Original Series
Please read terms of use for authorized access





Need for Web Services

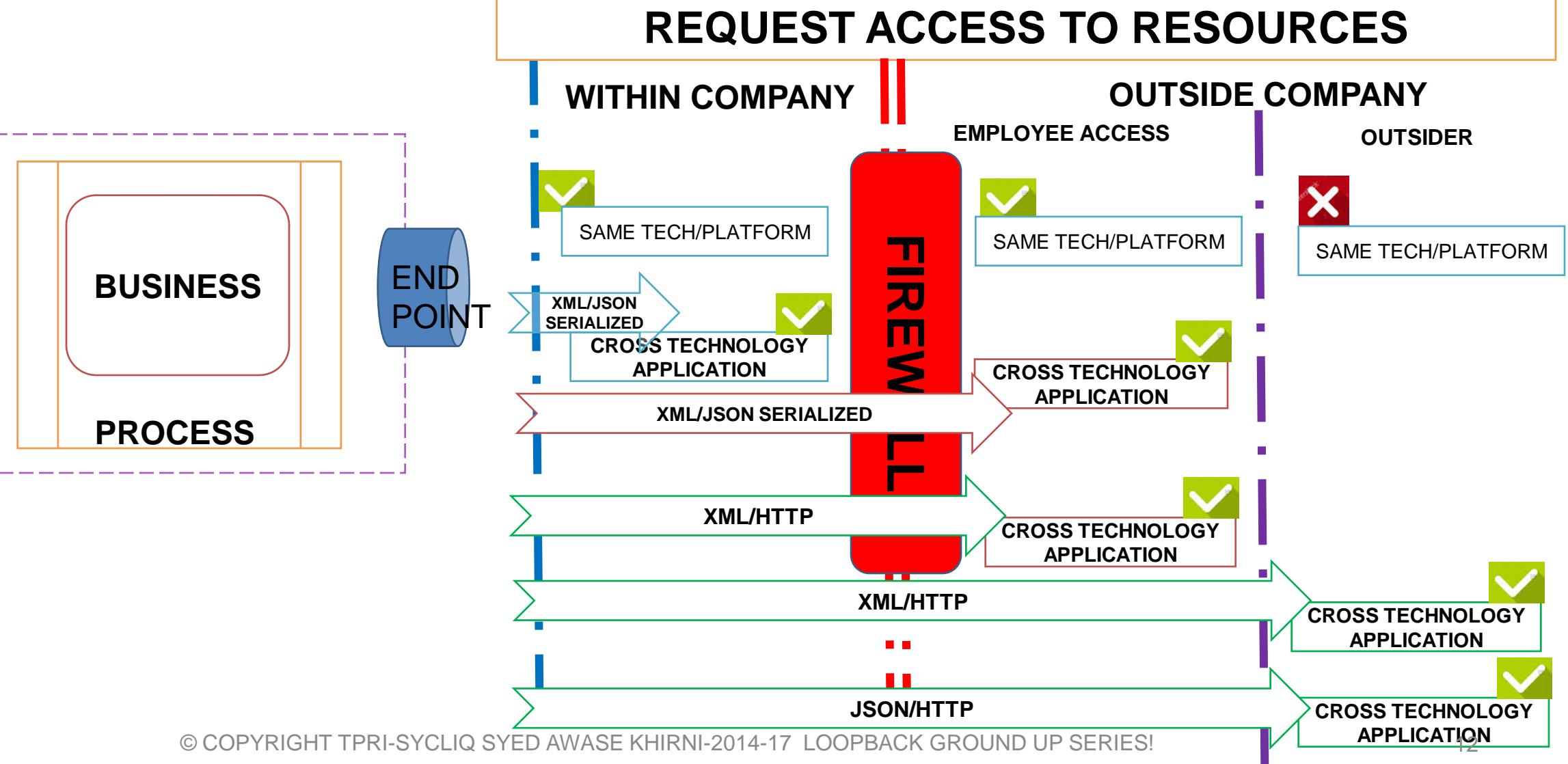




Need for Web Services

Please read terms of use for authorized access

Original Series





LoopBack

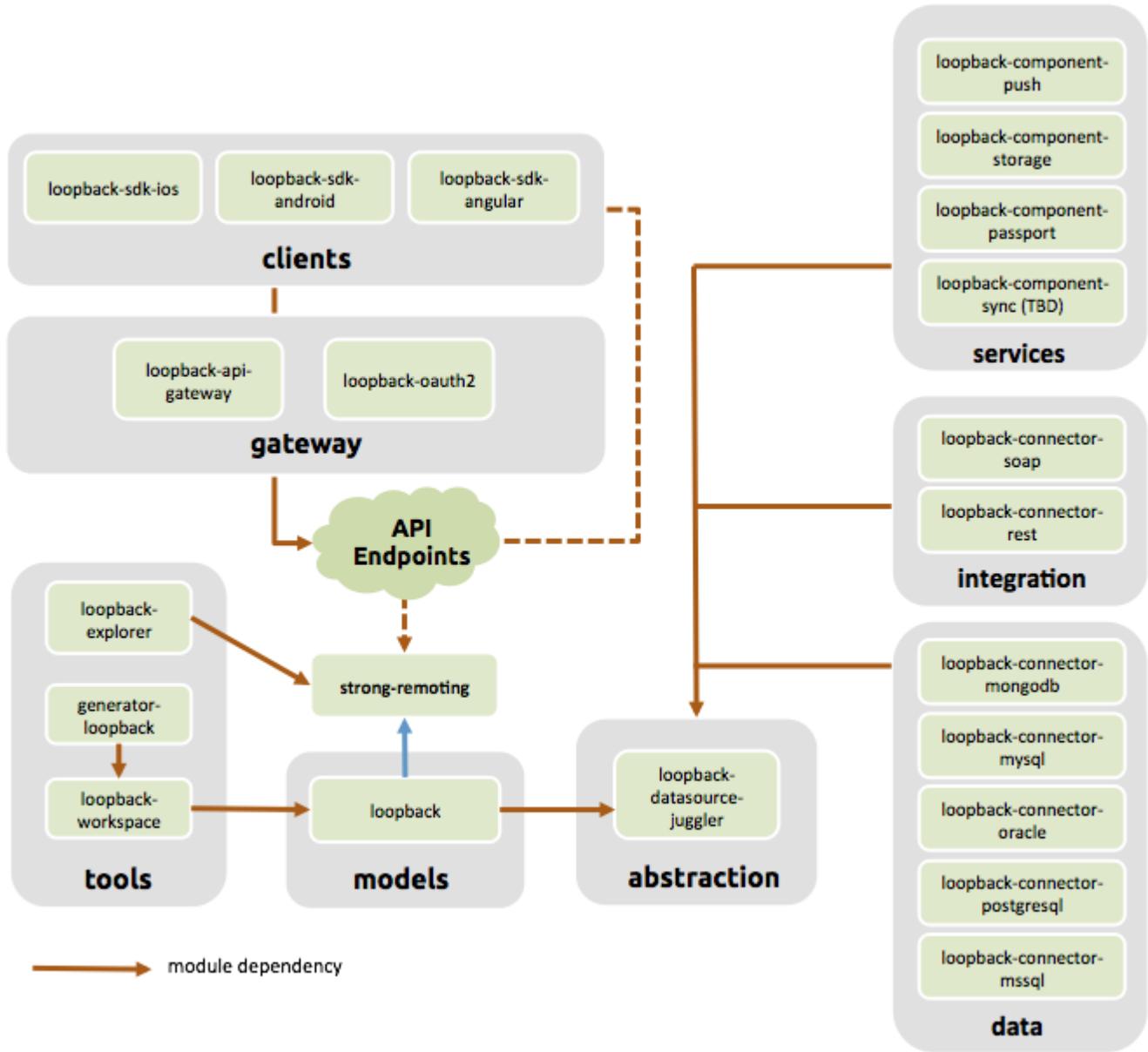
- Create dynamic end-to-end REST APIs with little or no coding.
- Access data from major relational database, MongoDB, SOAP and REST APIs
- Incorporate model relationships and access controls for complex APIs
- Use geolocation, file, and push services for mobile apps.
- Easily create client apps using Android, iOS and JavaScript SDKs
- Run your application on-premises or in the cloud.



LoopBack Architecture

Please read terms of use for authorized access

Original Series

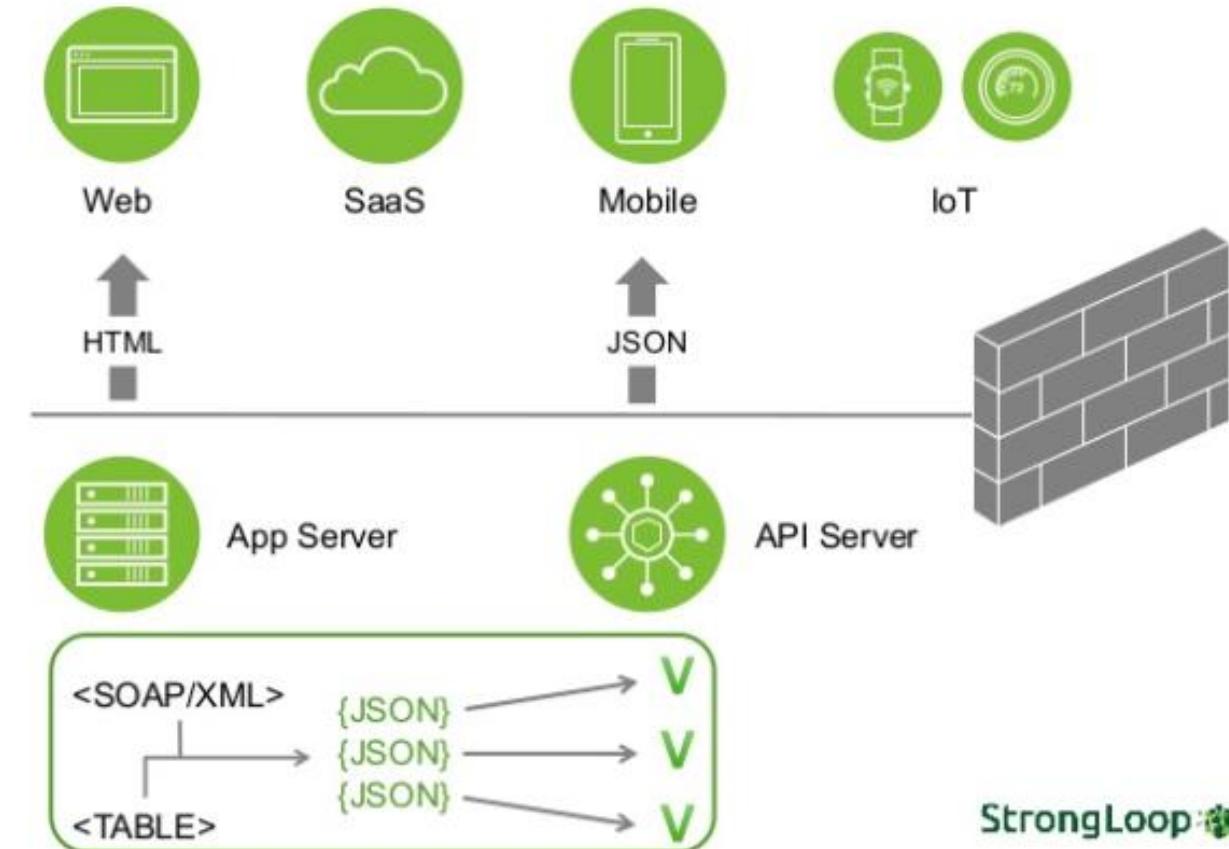




LoopBack API

Please read terms of use for authorized access

Original Series





Frameworks to build WEB API

Please read terms of use for authorized access



express



Moleculer



koa

next generation web framework for node.js



Frisby.js

Original Series



Evolution of Frameworks

Please read terms of use for authorized access

Original Series

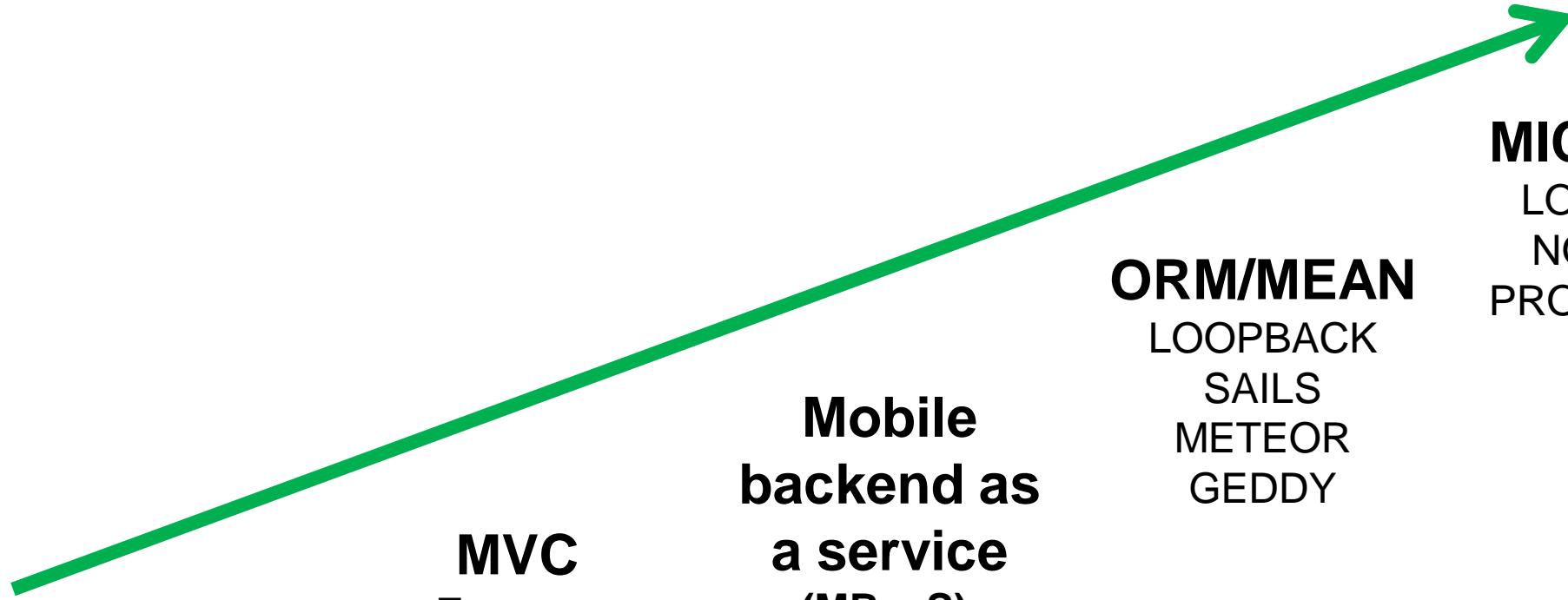
KISS
Callback
Reactor
Observer

MVC
Express
HAPI
RESTIFY
TOTAL
PARTIAL

Mobile backend as a service (MBaaS),
Loopback
Parse
Meteor
Feedhenry

ORM/MEAN
LOOPBACK
SAILS
METEOR
GEDDY

MICRO/IOT
LOOPBACK
NODERED
PROPRIETARY





LoopBack vs Others

Please read terms of use for authorized access

	LoopBack	Express	Hapi	Sails	Restify	Meteor
Type	API Framework	Http Server Library	Http Server Framework	Web MVC Framework	RestHTTP Library	Full stack Javascript App platform
Original Series	Enterprise connectivity, API explorer, generators, client, SDKs, websocket, microservices	HTTP routing, middle ware	Modularity, security	Rails Familiarity, MVC	Simplicity, REST routing	Universal javascript, reactive rendering, web socket, microservices
Suitable For	Webapps, API	Simple web apps	Web apps, APIs	Web apps, APIs	Simple REST APIs	Web apps
Github Stars	8000	29000+	7000+	16000+	6000+	36000+
Support	IBM/Stronglo op	IBM/Stronglo op	NA	NA	NA	Meteor Development Group



LoopBack vs Others

Please read terms of use for authorized access

Original Series

	LoopBack	Express	Hapi	Sails	Restify	Meteor
Pure Node runtime	Yes	Yes	Yes	Yes	Yes	No
Client SDKs	Angular, Browser, Nodejs, iOS, Android, Xamarin	NA	NONE	NONE	NONE	Javascript, cordova for iOS and Android, React, AngularJS
Export API definition	Yes	With strong-remoting	None	None	None	With meteor-rest
Tools	API Explorer, CLI code-generators	CLI app generator	Yeoman generator	Yeoman generator	Yeoman generator	CLI tool
Extensions	Push, File Storage, Passport Oauth 2.0, Express Middleware	Express/Connect middleware	Hapi plugins			Propreitary, package system and repository, npm



LoopBack vs Others

Please read terms of use for authorized access

Original Series

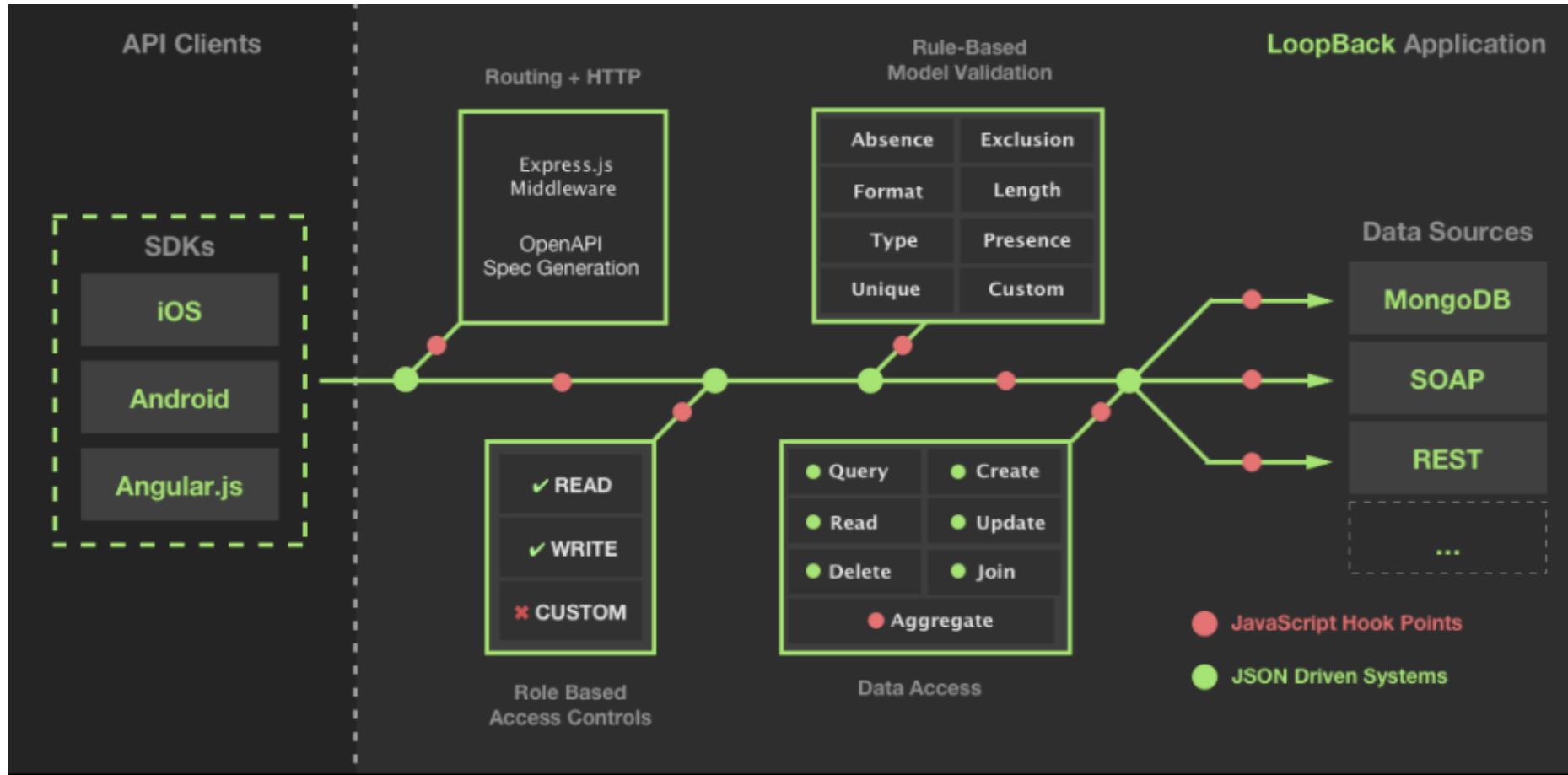
	LoopBack	Express	Hapi	Sails	Restify	Meteor
Data sources	In-memory/file, mongoDB, MySQL, Oracle, PostgreSQL, SQL Server, ATG, Email, REST, SOAP, many more	None	None	In-memory, File, PostgreSQL, MySQL, MongoDB	None	MongoDB, MySQL and PostgreSQL, via 3 rd -party packages.
ACLs	Yes	No	No	No	No	Basic allow/deny



A typical LoopBack Application Data Flow

Please read terms of use for authorized access

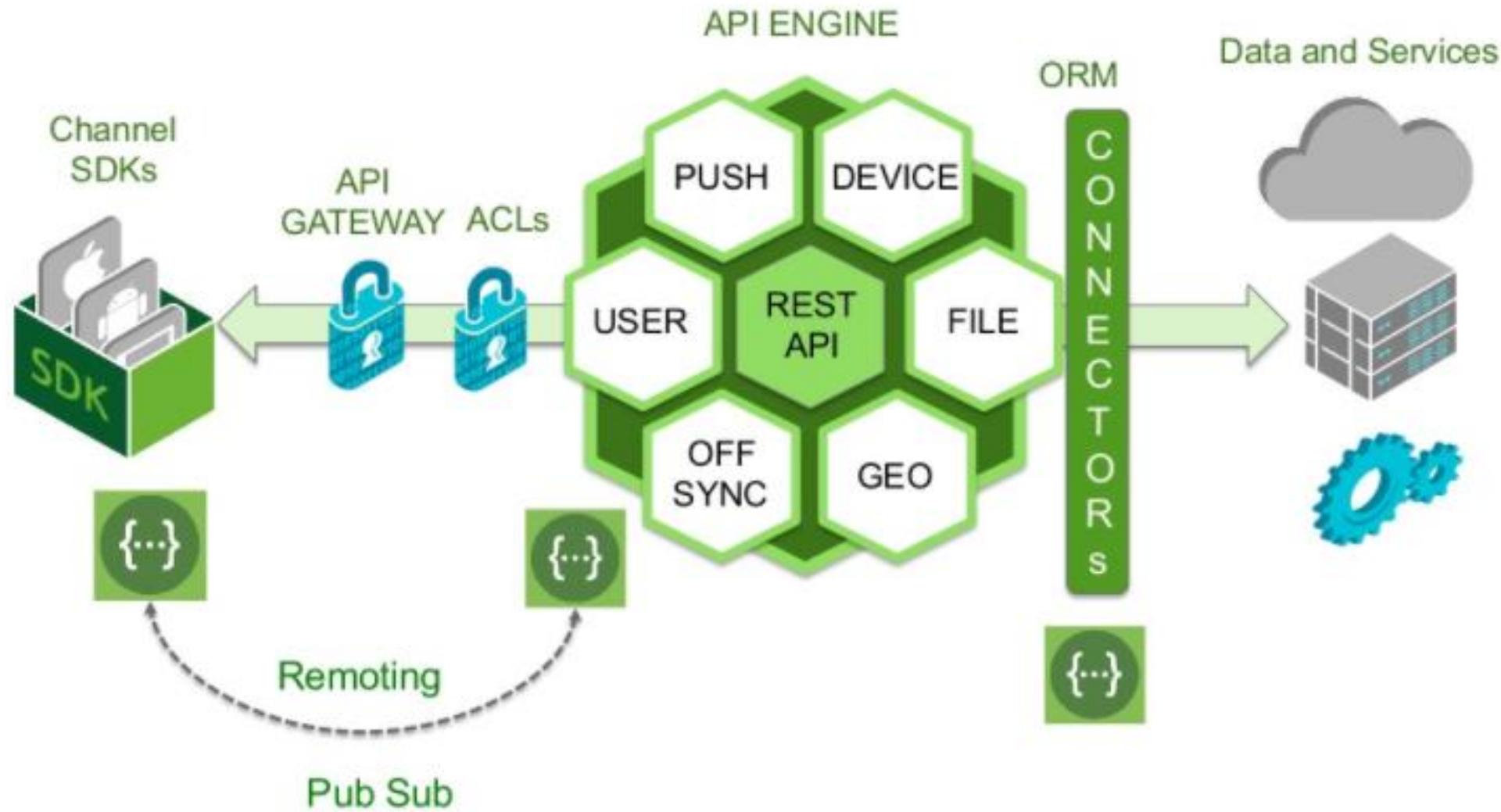
Original Series





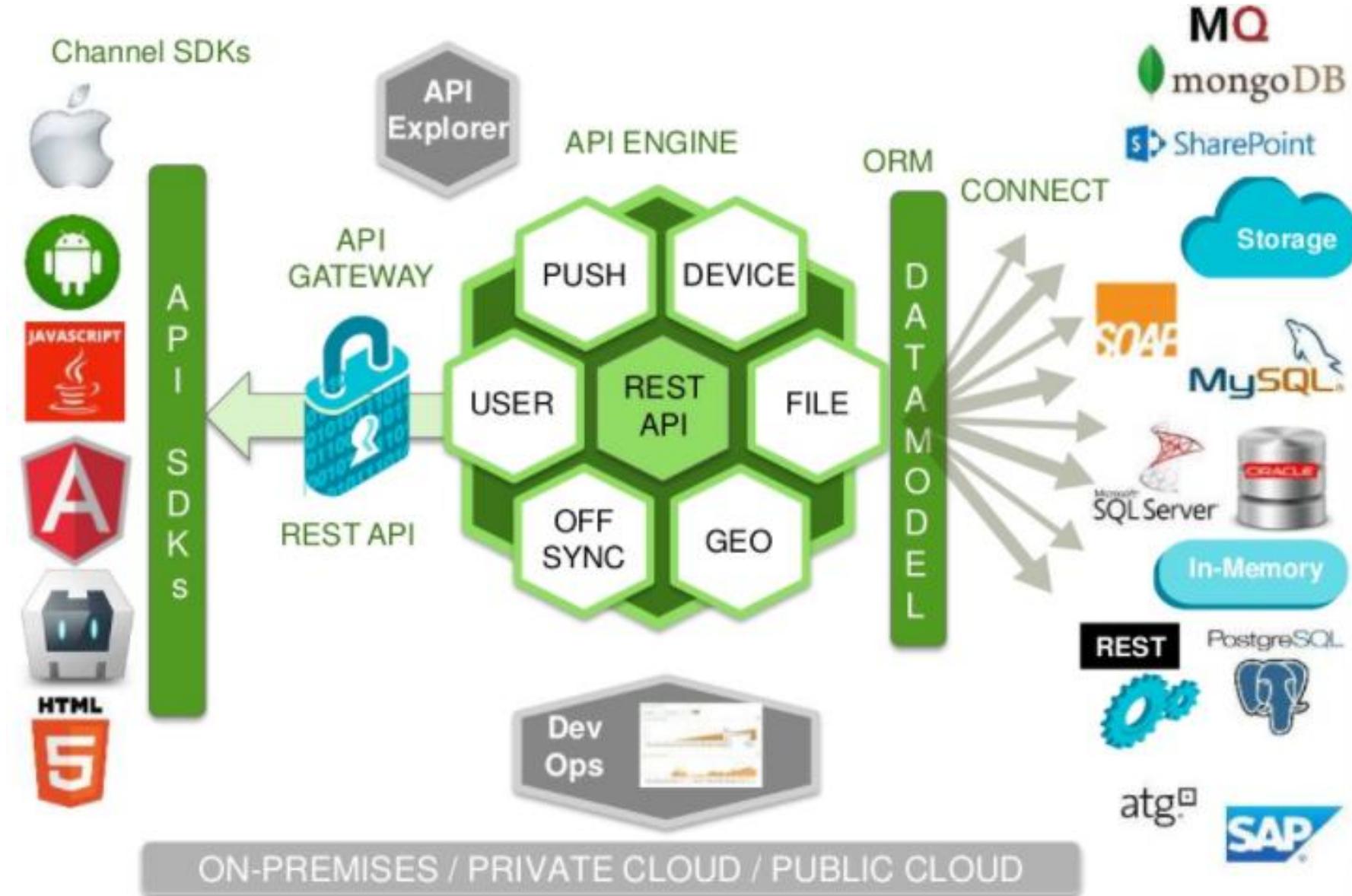
Please read terms of use for authorized access

Original Series





Please read terms of use for authorized access





LoopBack STACK



Node.js Enterprise DevOps

Enterprise Support for Node.js

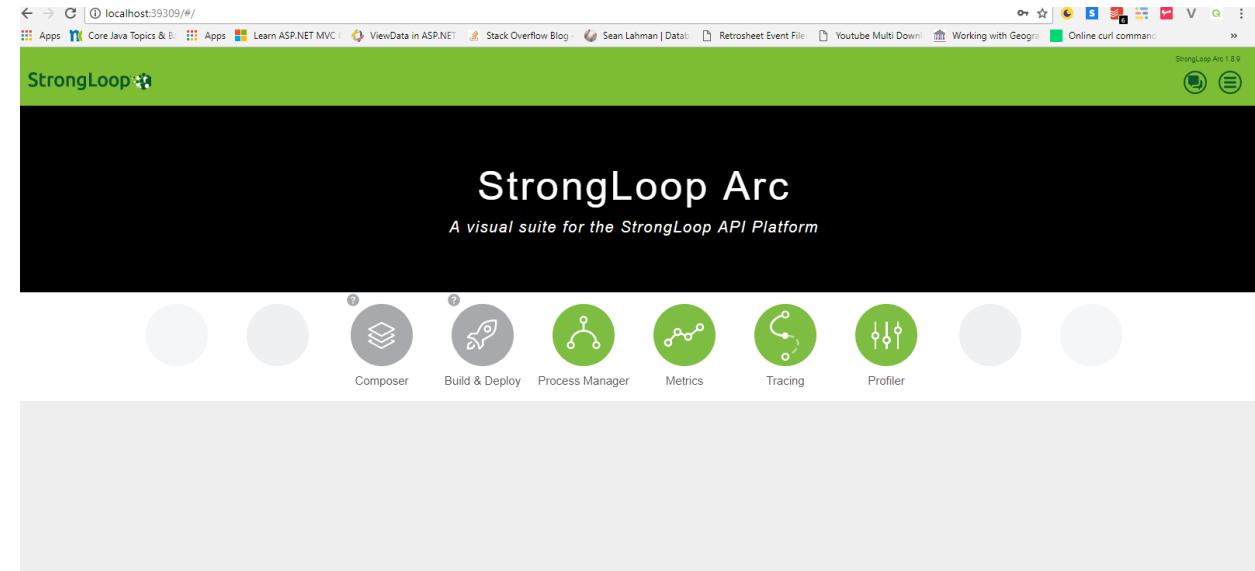
Please read terms of use for authorized access

Original Series



Arc UI: Architect View

- Widget driven
- Visual composition, management and operations
- Marquee features, evolving platform.
- Deprecated recently in 2015.
- Register with
<https://developer.ibm.com/apiconnect/>
- slc arc => use the login to use arc





StrongLoop Console (SLC)

- Developer view
- Command line interface
- Full featured
- Greater customization support

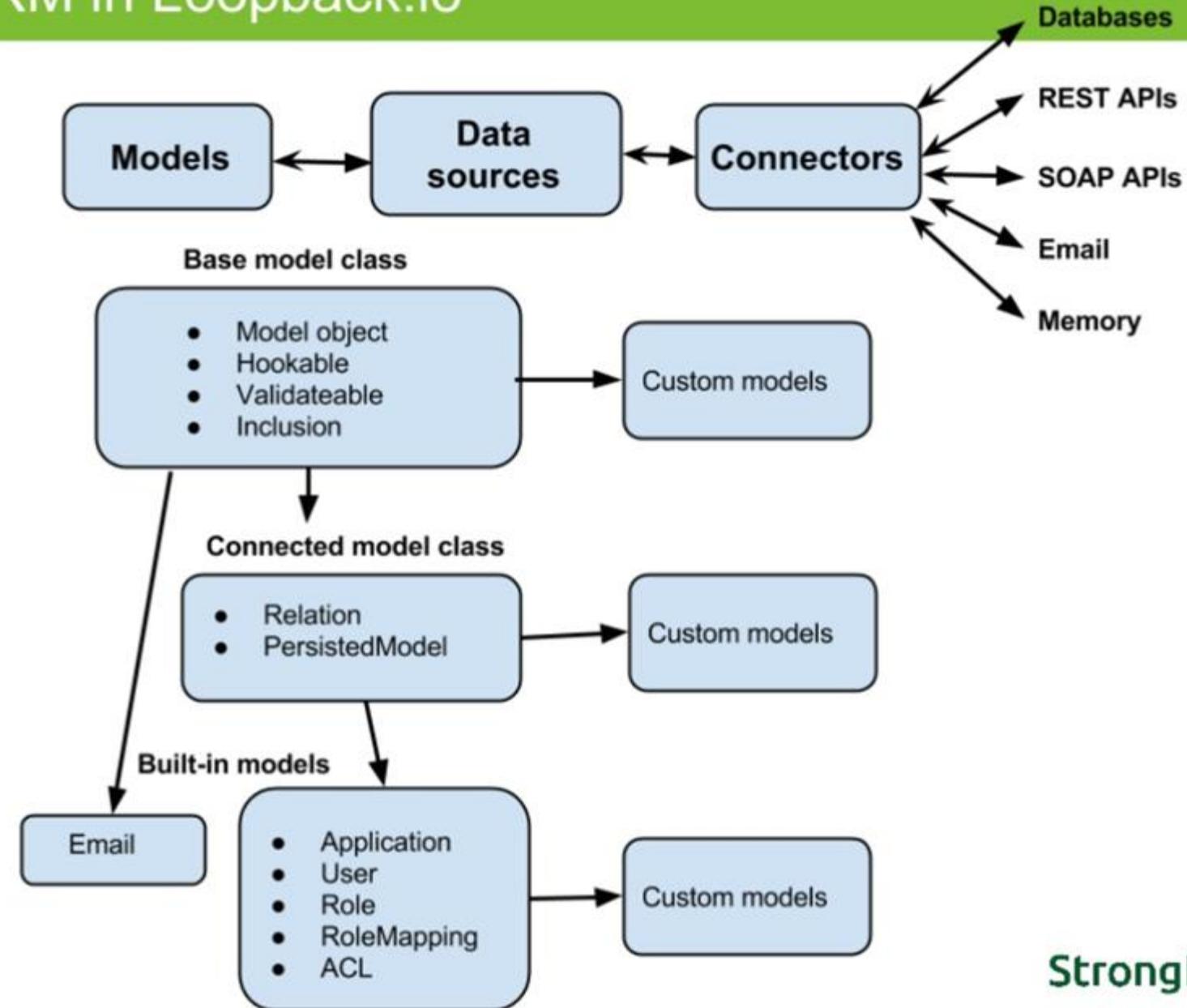


ORM in Loopback.io

Object

Please read terms of use for authorized access

Original Series

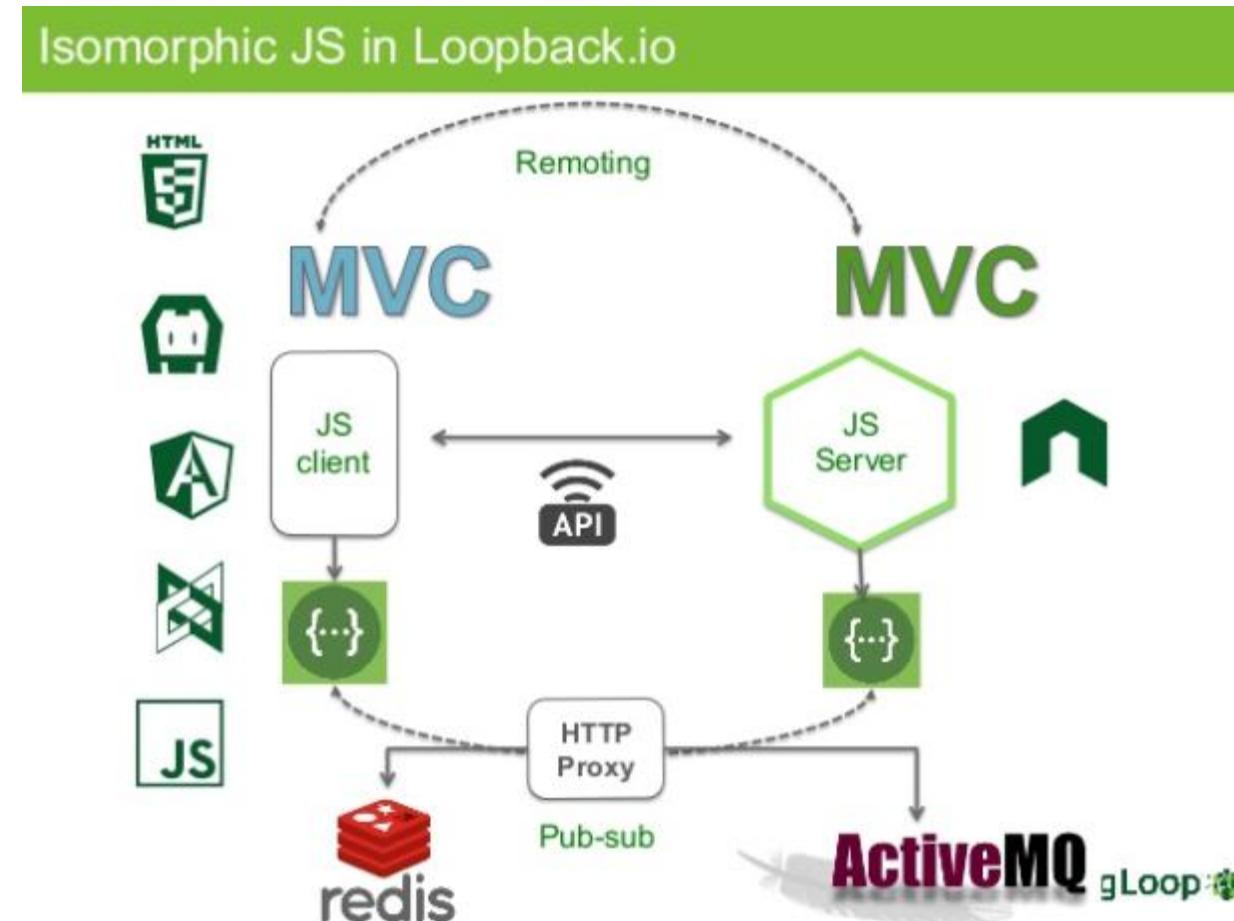
Ac
Go

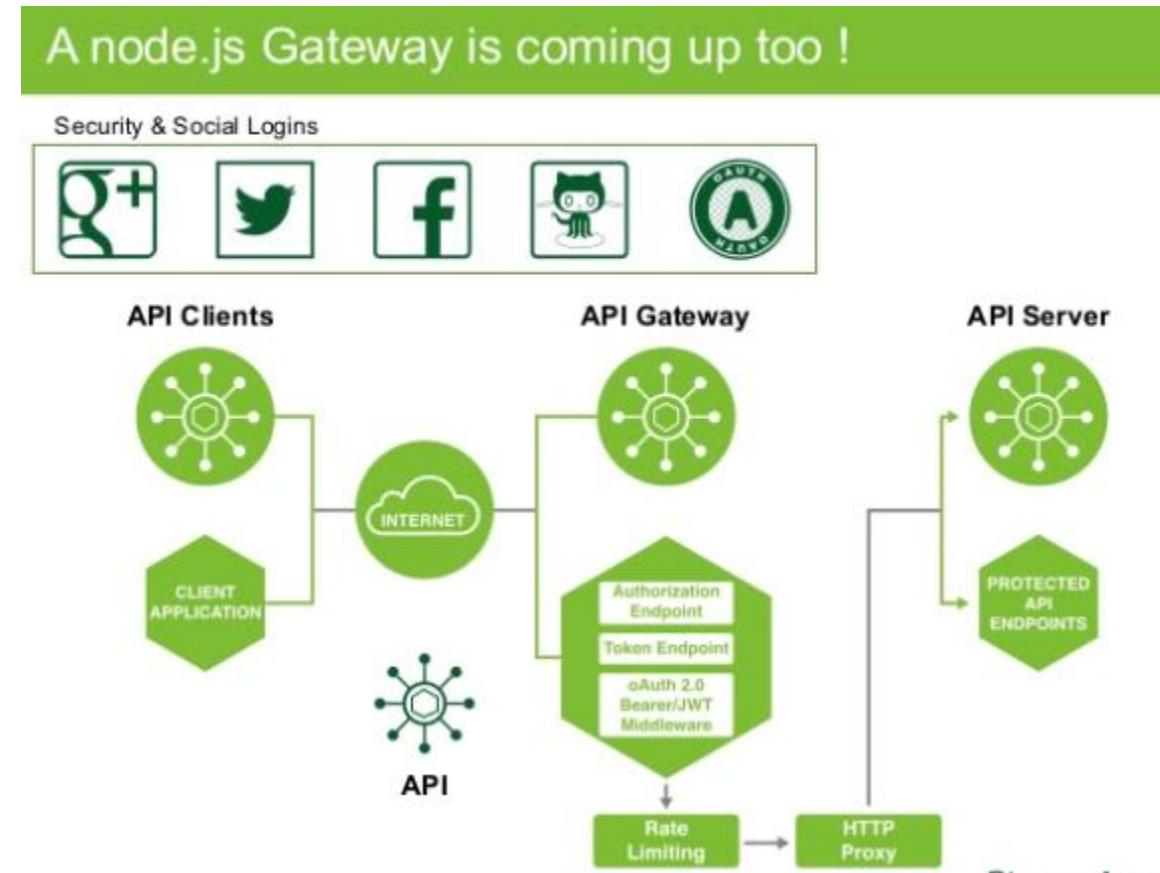
StrongLoo



Please read terms of use for authorized access

Original Series







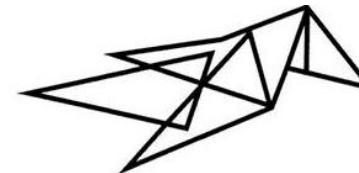


LoopBack Users

Please read terms of use for authorized access



devTango



huninnmesh



Original Series



Frameworks built on Express

- Feathers
<https://github.com/feathersjs/express>
- ItemsAPI
- KeystoneJS
- Kraken
- Lan-stack
- LoopBack
- MEAN
- SAILS



Express	Restify
DIY	DIY
HTTP ROUTING, MIDDLEWARE, TEMPLATING	SIMPLICITY, REST ROUTING
SIMPLE WEB APPS	SIMPLE REST API
COMMERCIAL SUPPORT-STRONGLOOP	JOYENT
CLI APP GENERATOR	YEOMAN GENERATOR
EXPORT API DEFINITION WITH STRONG-REMOTING	SPDY SUPPORT
CONNECT/EXPRESS MIDDLEWARE EXTENSIONS	Dtrace SUPPORT
STRONG LOOP DEVOPS TOOLING SUPPORT	NO EXPORT OF API DEFINITION
MANUAL CRUD ENDPOINTS	NO DATA SOURCE SUPPORT
MANUAL RECURSIVE REFACTORING/TESTING	NO EXTENSIONS OF SIGNIFICANCE
NO DATA SOURCE SUPPORT	MANUAL RECURSIVE REFACTORING/TESTING
NO CLIENT SDK	LIMITED ECOSYSTEM
NO ACL (FINE AUTHORIZATION)	NO ACL(FINE AUTHORIZATION)



HAPI	SAILS
OPINIONATED	OPINIONATED
MODULARITY,SECURITHY, HTTP SERVER CONTROL	WEB MVC FRAMEWORK
COMPLEX WEB APPS, APIS	WEBAPPS, API
COMMERCIAL-SUPPORT-NONE	RAILS FAMILIARITY, MVC
YEOMAN GENERATOR	COMMERCIAL SUPPORT –NONE
NO EXPORT OF API DEFINITION	YEOMAN GENERATOR
HAPI PLUGINS-JOI,BELL,BASSMASTER, ETC	STRONG LOOP DEVOPS TOOLING SUPPORT
LIMITED ECOSYSTEM AS COMPARD TO EXPRESS, SUPPORTS TEMPLATES WITH PLUGINS	NO EXPORT OF API DEFINITIONS
SWAGGER SUPPORT FOR API DOCUMENTATION	LIMITED DATA SOURCES – MYSQL, MONGODB, POSTGRESQL, MEMORY/FILE
VERY LIMITED DATA SOURCE SUPPORT-MONGODB, POSTGRES, LEVELDB	NO EXTENSIONS OF SIGNIFICANCE
NO CLIENT SDK	LIMITED ECOSYSTEM
NO ACL(FINE AUTHORIZATION)	NO ACL(FINE AUTHORIZATION)
	NO CLIENT SDKS

**LOOPBACK**

OPINIONATED

API FRAMEWORK

COMPLEX WEB APPS AND APIs

MODULARITY, SCAFFOLDING AND ENTERPRISE CONNECTIVITY

COMMERCIAL SUPPORT-STRONG LOOP

CLI CODE GENERATOR, VISUAL API COMPOSER, API EXPLORER

EXPORT API DEFINITIONS

EXTENSIONS LIKE PUS, FILE STORAGE, PASSPORT, Oauth 2.0, Express Middleware

VAST CONNECTOR ECOSYSTEM AND EXTENDED EXPRESS

AUTO CRUD

SWAGGER UI AND CODE GENERATOR

DATA SOURCES LIKE MONGO, MYSQL, ORACLE, SOAP, REST, SQL SERVER, MEMORY/FILE, POSTGRESQL, EMAIL, ATG, COUCHBASE, APACHE KAFKA, AND MANY MORE.

CLIENT SDK – ANGULAR, BROWSER, IOS, ANDROID, NODE.JS

MATURE ACL(FINE AUTHORIZATION), API GATEWAY

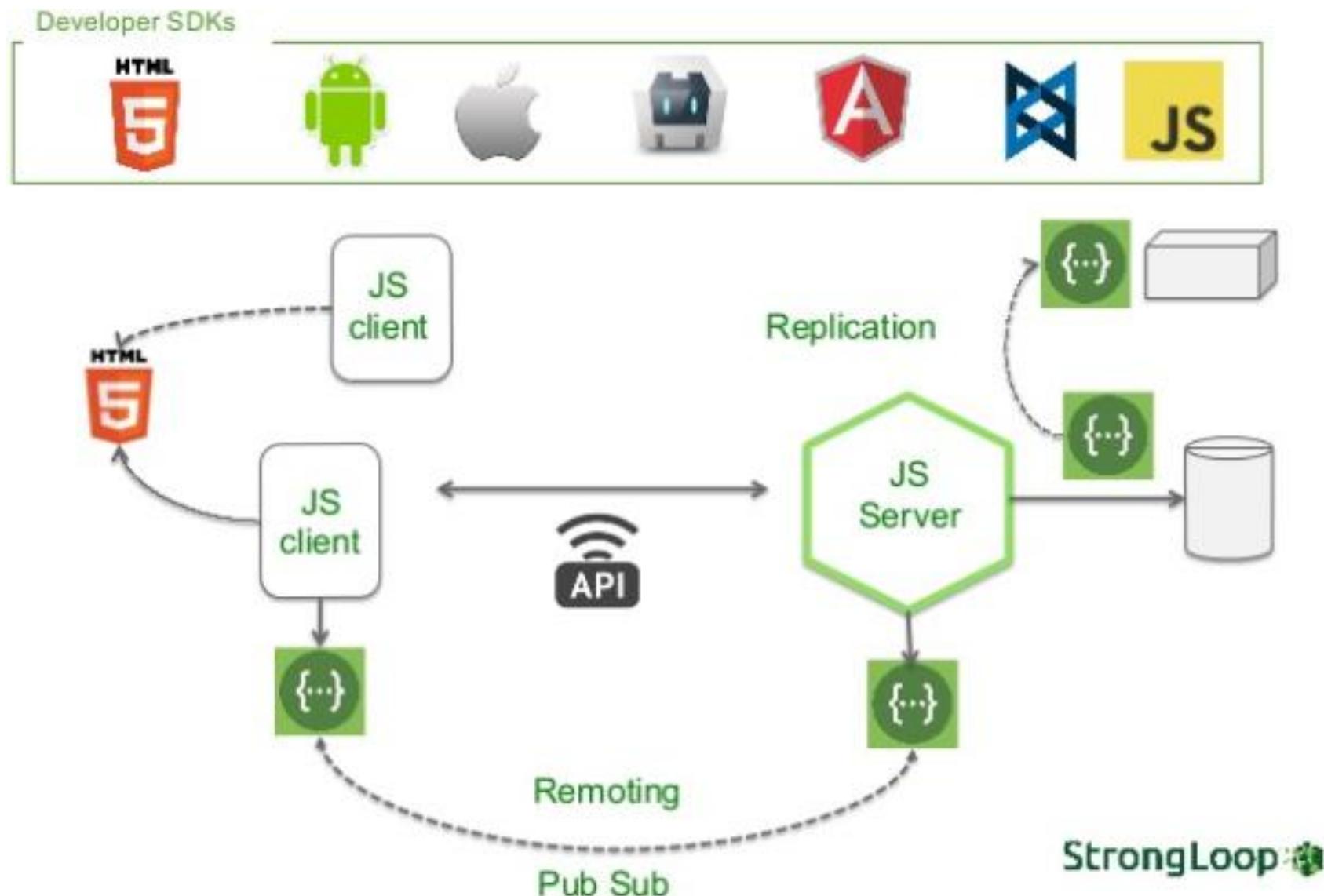


Express

- All end points need to be created manually, you end up doing a lot of the same code (or worse, start rolling your own libraries after a while)
- Every end point needs to be tested (or at the very least I recommend that you hit the end points with HTTP consumer to make sure they are actually there and don't throw 500s)
- Refactoring becomes painful because everything needs to be updated everywhere
- Doesn't come with anything "standard", have to figure out your own approach

LoopBack

- Very quick RESTful API development
- Convention over configuration
- Built in models ready to use
- RPC support
- Fully configurable when needed
- Extensive documentation
- Fulltime team working on the project
- Available commercial support

Original Series
Please read terms of use for authorized access



LoopBack Features

- Powerful ACLs
- Good core code quality
- Secure with baked in cross XSS
- Isomorphic model definitions
- Authentication, Authorization and Permissions.
- Clean separation between API server and client
- Baked in mobile and client SDKs
- Good Suite of surrounding enterprise tools like
 - StrongLoop Arc
 - Visual API composer
 - CLI code generators etc
- Third-party login
- Features like Storage Component, Push Notifications, Oauth Server



LoopBack

Please read terms of use for authorized access

Original Series



- Built on Top of Express
- Optimized for APIs
- Model Driven Architecture
- Enterprise Superglue
- Full Stack
- e2e SDLC Tooling



LoopBack Components

- LoopBack components are predefined packages that extend a basic loopback application.
- A component is related code bundled together as a unit to enable loopback applications for easy reuse.
- We can configure components declaratively in component-config.json.
- A loopback application is nothing more than a group of components with elements to tie them all together.

Component	Description
API Explorer	Enables the Swagger UI for the API
OAuth 2.0	Enables LoopBack applications to function as OAuth 2.0 providers to authenticate and authorize client applications and users to access protected API end points
Push Notifications	Adds push notification capabilities to your loopback application as a mobile backend service
Storage component	Adds an interface to abstract storage providers like S3, filesystem into general containers and files
Synchronization	Adds replication capability between LoopBack running in a browser or between LoopBack backend instances to enable offline synchronization and server-to-server data synchronization
Third-party login using Passport	Adds third-party login capabilities to your LoopBack application like Facebook, GitHub etc..



IBM API Connect

DaveYVR commented on May 16



In early 2016 the StrongLoop team announced that Arc would reach its end of life within a year. That year has passed, and Arc has been replaced by API Connect. New users cannot register for Arc, and we recommend interested users try out the Essentials edition of API Connect. If you have questions about API Connect, please email reachsl@us.ibm.com or visit this site:

https://console.ng.bluemix.net/catalog/services/api-connect?cm_mmc=Earned--IBM+Cloud+API+Economy--WW+WW--

StrongLoop+API+Connect+Page_ov49843&cm_mmca1=000000WT&cm_mmca2=10002833&=

You can also try LoopBack, our Open Source node framework:

<http://loopback.io/>

Thanks for reaching out.

https://console.ng.bluemix.net/catalog/services/api-connect?cm_mmc=Earned--IBM+Cloud+API+Economy--WW+WW--

StrongLoop+API+Connect+Page_ov49843&cm_mmca1=000000WT&cm_mmca2=10002833&=

The screenshot shows the IBM Cloud catalog page for the 'API Connect' service. The page header includes the IBM Cloud logo and navigation links for Catalog, Docs, and Log In. The main content area displays the product details for API Connect, highlighting its comprehensive end-to-end API lifecycle solution. It lists several key features: 'Securely unlock existing IT assets' (which allows for rapidly generating Swagger compliant APIs from backend databases), 'Graphical API assembly' (which enables graphical assembly of API invocation flows and applying policies for secure controlled access), 'Community & Subscription management' (which facilitates developer communities to publish and share APIs and engage with them through a self-service portal), and 'Gain insights about API consumption' (which provides built-in analytics and customized dashboards to empower businesses to make informed decisions). Below the features, there are sections for 'View all' and 'Images'.



SYED AWASE KHIRNI

LoopBack: Installation



COMMAND LINE INSTALLATION

<http://www.nodejs.org> install LTS Version

```
$ npm install -g loopback-cli
```

```
$ npm install -g strongloop
```

```
$ slc --help
```

```
$ slc loopback getting-started-node-loopback
```

Datastorage: MySQL,MongoDB,IBM UDB DB2, PostgreSQL,MySQLServer



LoopBack Application Structure

Please read terms of use for authorized access

Original Series

File Name	Description
Package.json	Development packages installed by npm
Common/models	Definition of basic models provided by loopback
Server/server.js	Main application file
Server/config.json	Machine-editable app configuration
Server/datasources.json	Definition of data sources
Server/model-config.json	Model configuration file
Server/middleware	Middleware configuration



Loopback Generators

Please read terms of use for authorized access

Original Series

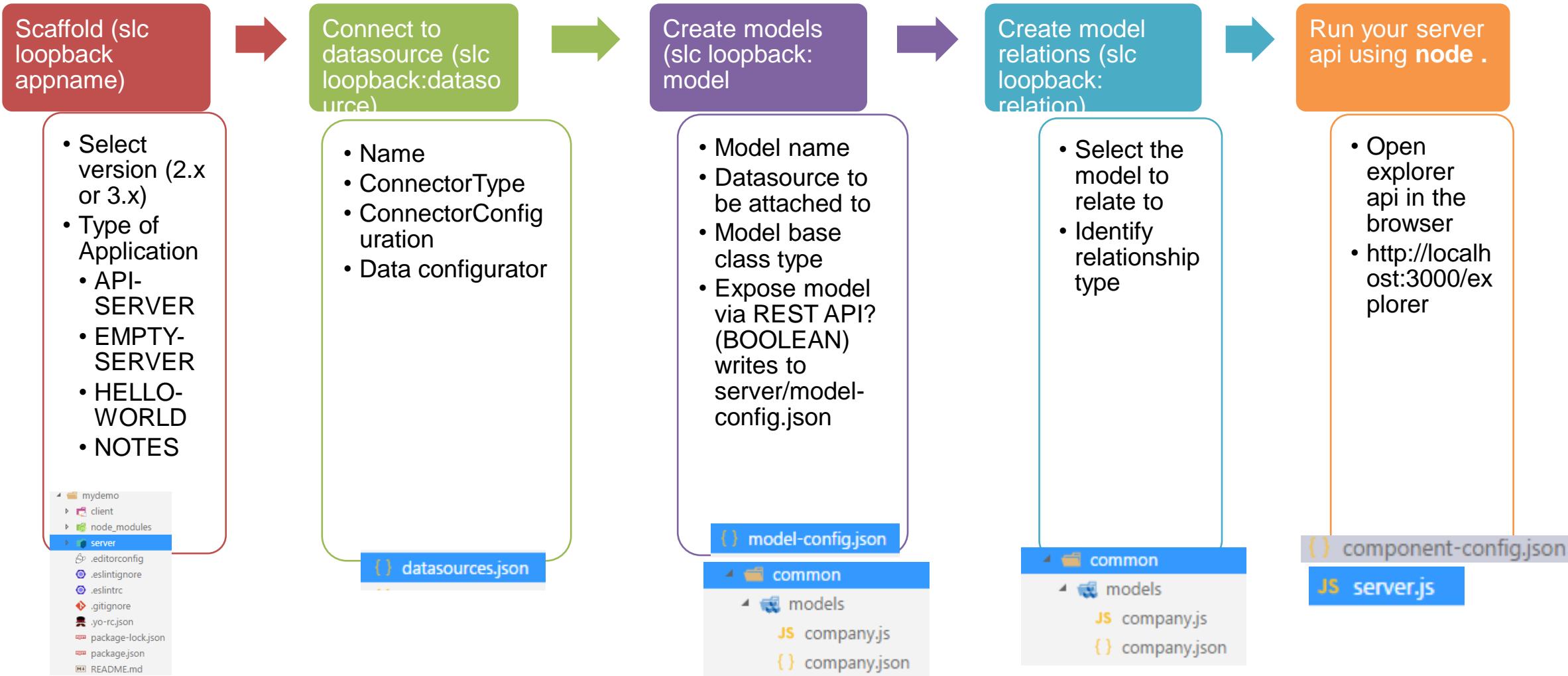
Command	Description
Slc loopback:acl	Access control list
Slc loopback:app	Creating a loopback application
Slc loopback:boot-script	Bootscript for your application
Slc loopback:datasource	Datasource configuration
Slc loopback:export-api-def	Export api definitions
Slc loopback:model	Model definitions, properties and property type
Slc loopback:property	Defining property to an existing loopback model
Slc loopback:remote-method	Defining custom remote methods
Slc loopback:swagger	



Loopback Workflow

Please read terms of use for authorized access

Original Series





LoopBack built-in models

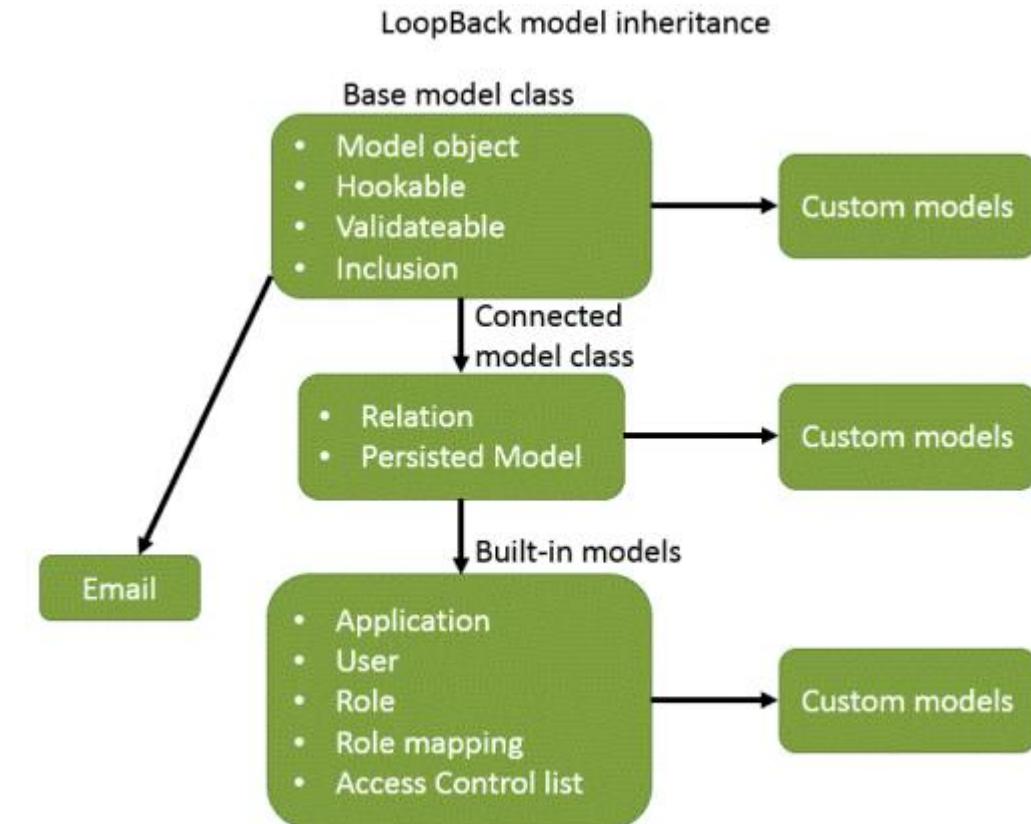
- Application model: contains metadata for a client application that has its own identity and associated configuration with the LoopBack Server.
- User model: used to register and authenticate users of your app locally or against third-party services.
- Access control models – ACL, access Token, Scope, Role and RoleMapping models for controlling access to applications, resources, and methods.
- Email model: send emails to app users using SMTP or third-party services.
- All models except for Email extend PersistedModel.
- Access Control Models
 - ACL
 - AccessToken
 - Scope
 - Role
 - RoleMapping



LoopBack Model Inheritance

Please read terms of use for authorized access

Original Series





Hello World Application

SLC LOOPBACK:DEMO-II



SLC Successful

- Upon on successful installation of loopback/strongloop

```
PS E:\CT\loopback\codeplay> slc loopback getting-started-node-loopback
```

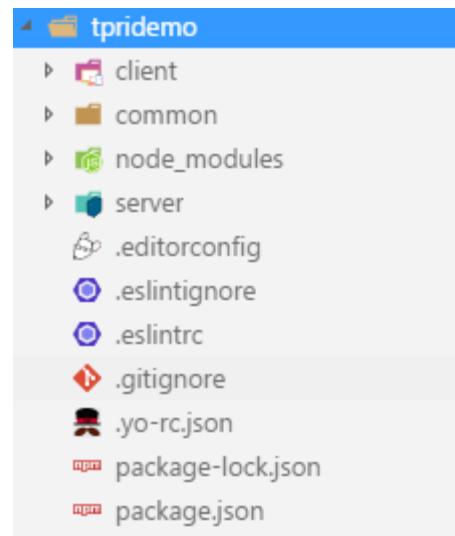


? What's the name of your application? (getting-started-node-loopback)

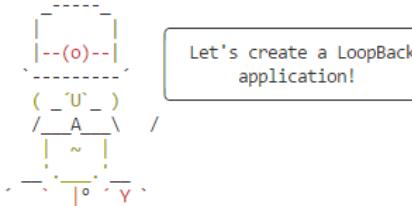


1. Basic working example

- Navigate through menu options on command line to create a basic working application as shown in the figure.



PS E:\CT\loopback\codeplay> `scl loopback getting-started-node-loopback`



```
? What's the name of your application? tpridemo
? Enter name of the directory to contain the project: tpridemo
  create tpridemo/
    info change the working directory to tpridemo

? Which version of LoopBack would you like to use? 2.x (long term support)
? What kind of application do you have in mind? notes (A project containing a basic working example, including a memory database)
Generating .yo-rc.json
```

I'm all done. Running `npm install` for you to install the required dependencies. If this fails, try running the command yourself.

```
create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create common\models\note.js
create common\models\note.json
create server\boot\authentication.js
create .gitignore
create client\README.md
```

Next steps:

Change directory to your app
`$ cd tpridemo`

Create a model in your app
`$ scl loopback:model`

Run the app
`$ node .`



2. Running your application.

- navigate to the directory where your application has been installed
- Run your application with **node**.

```
PS E:\CT\loopback\codeplay> cd tpridemo
PS E:\CT\loopback\codeplay\tpridemo> node .
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```



3. Browser view

- navigate to the directory where your application has been installed
- Run your application with **node .**

<http://localhost:3000/explorer/>

The screenshot shows the StrongLoop API Explorer interface. The URL in the address bar is `localhost:3000/explorer/#!/Note/Note_find`. The main title is `StrongLoop API Explorer`. The current model selected is `tpridemo`. The endpoint shown is `/Notes`. There are two tabs: `PATCH` and `GET`. The `PATCH` tab has a description: "Patch an existing model instance or insert a new one into the data source." The `GET` tab has a description: "Find all instances of the model matched by filter from the data source." Below the tabs, there is a "Response Class (Status 200)" section. It shows the `Model Schema` which contains a JSON object definition:

```
[  
  {  
    "title": "string",  
    "content": "string",  
    "id": 0  
  }  
]
```

Below the schema, the "Response Content Type" is set to `application/json`. Under the "Parameters" section, there is a table with the following columns: Parameter, Value, Description, Parameter Type, and Data Type. A single row is present for the parameter `filter`:

Parameter	Value	Description	Parameter Type	Data Type
<code>filter</code>	<input type="text"/>	Filter defining fields, where, include, order, offset, and limit - must be a JSON-encoded string ({"something":"value"})	query	string



HELLOWORLD APPLICATION

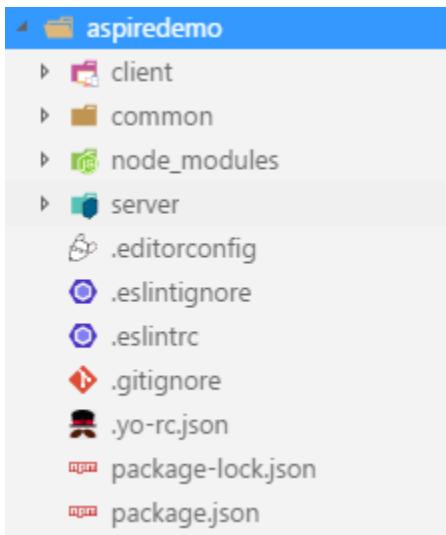
SLC LOOPBACK:DEMO-III



Step 1

Hello World Application

- Create another loopback application with current version 3.x
- Hello world application with controller.



```
PS E:\CT\loopback\codeplay> slc loopback
Let's create a LoopBack application!
? What's the name of your application? aspiredemo
? Enter name of the directory to contain the project: aspiredemo
  info change the working directory to aspiredemo

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? hello-world (A project containing a controller, including a single vanilla Message and a single remote method)
Generating .yo-rc.json

I'm all done. Running npm install for you to install the required dependencies. If this fails, try running the command yourself.

create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create common\models\message.js
create common\models\message.json
create server\boot\authentication.js
create .gitignore
create client\README.md
npm WARN deprecated nodemailer@2.7.2: All versions below 4.0.1 of Nodemailer are deprecated. See https://nodemailer.com/status/
```

Activate Window

Next steps:

Change directory to your app
`$ cd aspiredemo`

Create a model in your app
`$ slc loopback:model`

Run the app
`$ node .`



Step 2

Hello World Application

- change to the aspiredemo directory
- Run your application node .

```
PS E:\CT\loopback\codeplay\aspiredemo> node .
strong-remoting deprecated Remoting metadata "isStatic" is deprecated. Please specify "prototype.name" in method name instead for isStatic=false. <anonymous>;null:null
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

<http://localhost:3000/explorer/#/>

The screenshot shows the LoopBack API Explorer interface. At the top, there's a header bar with tabs like Java Topics & Books, Apps, Learn ASP.NET MVC, ViewData in ASP.NET, Stack Overflow Blog, Sean Lahman | Database, Retrosheet Event File, Youtube Multi Downloader, and Working with Geography. Below the header, the title 'LoopBack API Explorer' is displayed, along with a token input field ('Token Not Set') and a 'Set Access Token' button.

The main content area is titled 'aspiredemo'. It shows two sections: 'Message' and 'User'. The 'User' section is expanded, listing the following operations:

Method	Endpoint	Description
GET	/Messages/greet	
PATCH	/Users	Patch an existing model instance or insert a new one into the data source.
GET	/Users	Find all instances of the model matched by filter from the data source.
PUT	/Users	Replace an existing model instance or insert a new one into the data source.
POST	/Users	Create a new instance of the model and persist it into the data source.
PATCH	/Users/{id}	Patch attributes for a model instance and persist it into the data source.
GET	/Users/{id}	Find a model instance by {{id}} from the data source.
HEAD	/Users/{id}	Check whether a model instance exists in the data source.
PUT	/Users/{id}	Replace attributes for a model instance and persist it into the data source.
DELETE	/Users/{id}	Delete a model instance by {{id}} from the data source.
GET	/Users/{id}/accessTokens	Queries accessTokens of User.



EMPTY LOOPBACK APPLICATION

SLC LOOPBACK:DEMO-III



Step 1

Empty LoopBack Application

- create another loopback application with empty selection as shown in the figure with current version 3.x

PS E:\CT\loopback\codeplay> `scl` loopback



```
? What's the name of your application? truestate demo
? Enter name of the directory to contain the project: truestate demo
  create truestate demo/
    info change the working directory to truestate demo

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? empty-server (An empty LoopBack API, without any configured models or datasources)
Generating .yo-rc.json
```

I'm all done. Running `npm install` for you to install the required dependencies. If this fails, try running the command yourself.

```
create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create .gitignore
create client\README.md
```

Next steps:

Change directory to your app
`$ cd truestate demo`

Create a model in your app
`$ scl loopback:model`

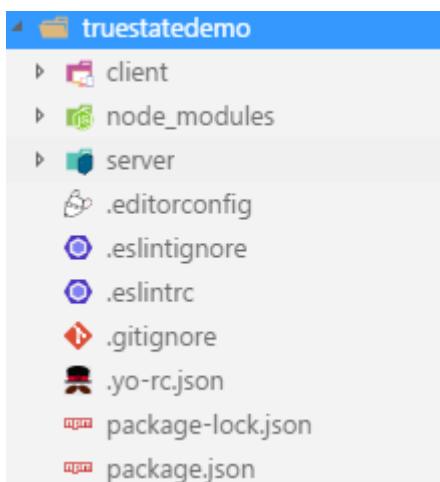
Run the app
`$ node .`



Step 2

Empty LoopBack Application

- change to the truestatedemo directory
- Run your application with node .



PS E:\CT\loopback\codeplay> `scl loopback`



```
? What's the name of your application? truestatedemo
? Enter name of the directory to contain the project: truestatedemo
create truestatedemo/
  info change the working directory to truestatedemo

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? empty-server (An empty LoopBack API, without any configured models or datasources)
Generating .yo-rc.json
```

I'm all done. Running `npm install` for you to install the required dependencies. If this fails, try running the command yourself.

```
create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create .gitignore
create client\README.md
```

Next steps:

Change directory to your app
`$ cd truestatedemo`

Create a model in your app
`$ scl loopback:model`

Run the app
`$ node .`



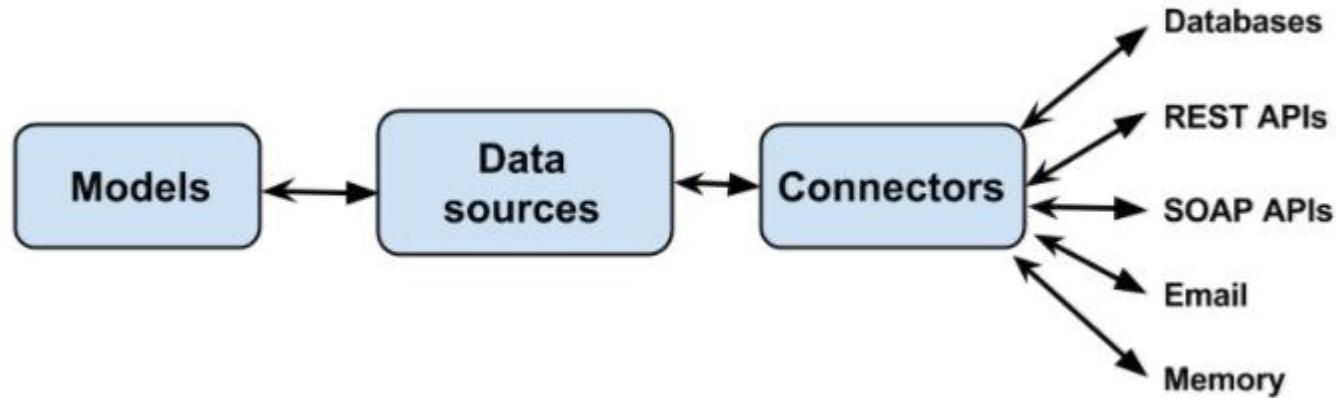
Step 3 Empty LoopBack Application

- open browser view
<http://localhost:3000/explorer/>

```
PS E:\CT\loopback\codeplay> cd .\truestatedemo\  
PS E:\CT\loopback\codeplay\truestatedemo> node .  
Web server listening at: http://localhost:3000  
Browse your REST API at http://localhost:3000/explorer
```

<http://localhost:3000/explorer/>

The screenshot shows a web browser window with the URL localhost:3000/explorer/. The page title is "LoopBack API Explorer". The main header reads "truestatedemo" and includes the text "[BASE URL: /api , API VERSION: 1.0.0]". At the top right, there is a search bar labeled "accessToken" and a button labeled "Set Access Token". The browser's address bar also shows "localhost:3000/explorer/". The page has a dark green header and a white body with some placeholder text.



BANK DEMO WITH MONGODB

SLC LOOPBACK:DEMO-IV

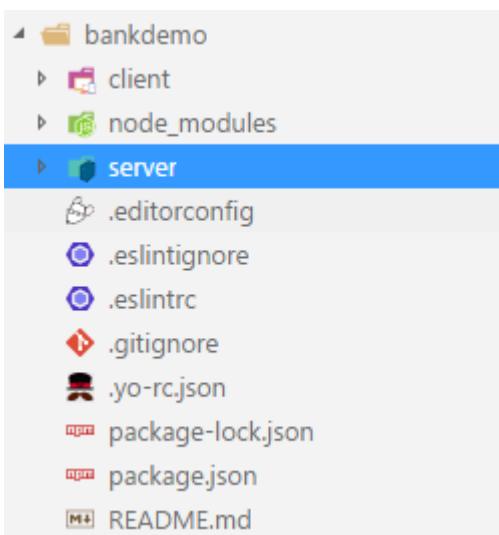


BANK DEMO

Step 1

Create API server project

- create API server project using `slc loopback` as shown in the figure using current version 3.x



PS E:\CT\loopback\codeplay> `slc loopback`



```
? What's the name of your application? bankdemo
? Enter name of the directory to contain the project: bankdemo
  create bankdemo/
    info change the working directory to bankdemo

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? api-server (A LoopBack API server with local User auth)
Generating .yo-rc.json
```

I'm all done. Running `npm install` for you to install the required dependencies. If this fails, try running the command yourself.

```
create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
```

Next steps:

Change directory to your app
\$ `cd bankdemo`

Create a model in your app
\$ `slc loopback:model`

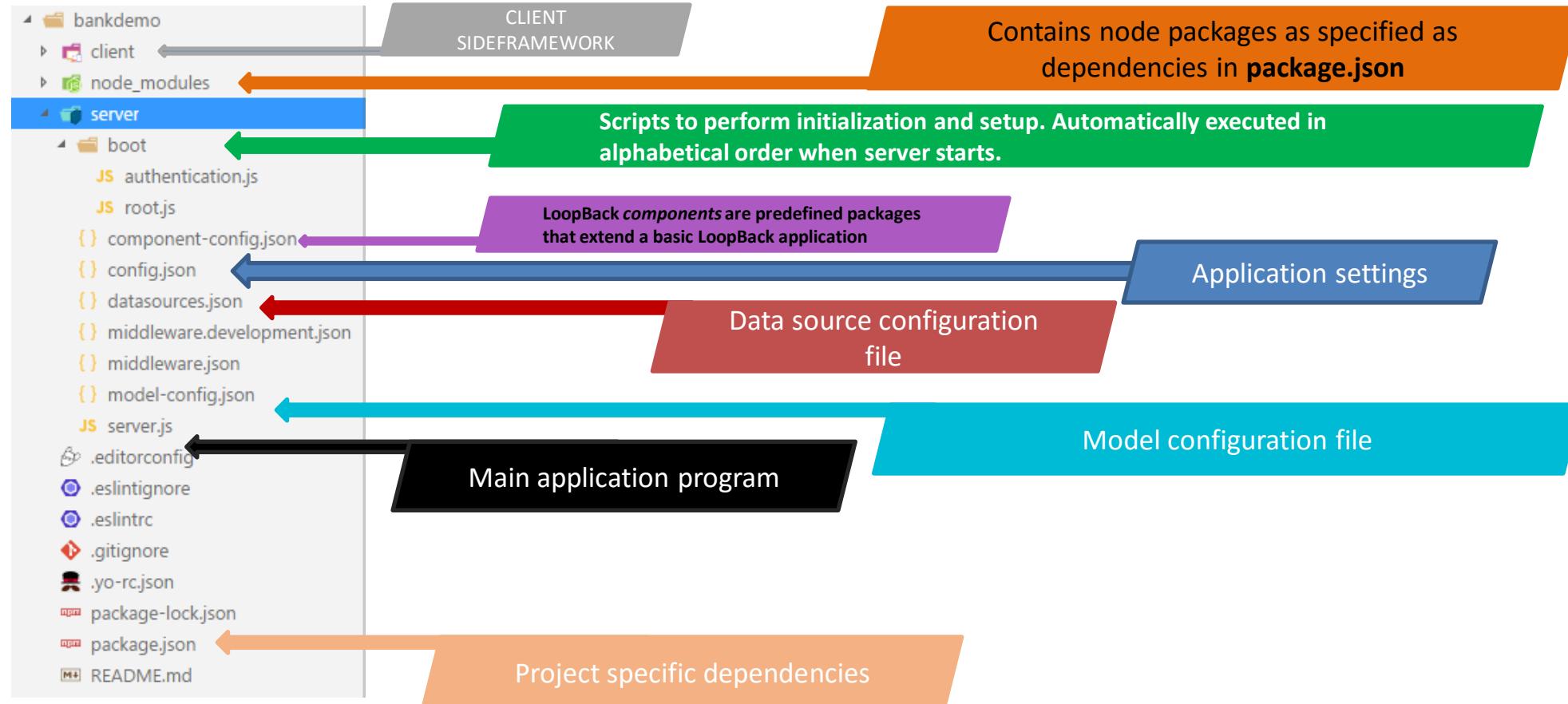
Run the app
\$ `node .`



API Server Project Folder

Please read terms of use for authorized access

Original Series



**Step 2**

```
$ npm install --save loopback-connector-mongodb
```

Install slc loopback:datasource

- configure your loopback to MongoDB as datasource for persistent storage.

```
PS E:\CT\loopback\codeplay> cd .\bankdemo\  
PS E:\CT\loopback\codeplay\bankdemo> slc loopback:datasource  
? Enter the data-source name: bankdemo  
? Select the connector for bankdemo: MongoDB (supported by StrongLoop)  
Connector-specific configuration:  
? Connection String url to override other settings (eg: mongodb://username:password@hostname:port/database): mongodb://@localhost:27017/bankloop  
? host: localhost  
? port: 27017  
? user:  
? password:  
? database: bankloop  
? Install loopback-connector-mongodb@^1.4 Yes  
+ loopback-connector-mongodb@1.18.1  
added 14 packages in 8.835s
```



Step 3

slc loopback:model

- create required models for rendering the API.
- customer model and address model

```
PS E:\CT\loopback\codeplay\bankdemo> slc loopback:model
? Enter the model name: Customer
? Select the data-source to attach Customer to: bankdemo (mongodb)
? Select model's base class PersistedModel
? Expose Customer via the REST API? Yes
? Custom plural form (used to build REST URL): customers
? Common model or server only? common
Let's add some Customer properties now.
```

Enter an empty property name when done.

```
? Property name: name
  invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```

Let's add another Customer property.

Enter an empty property name when done.

```
? Property name: age
  invoke loopback:property
? Property type: number
? Required? Yes
? Default value[leave blank for none]:
```

Let's add another Customer property.

Enter an empty property name when done.

```
? Property name: email
  invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```

Let's add another Customer property.

Enter an empty property name when done.

```
? Property name: [REDACTED]
```

```
PS E:\CT\loopback\codeplay\bankdemo> slc loopback:model
? Enter the model name: Address
? Select the data-source to attach Address to: bankdemo (mongodb)
? Select model's base class PersistedModel
? Expose Address via the REST API? Yes
? Custom plural form (used to build REST URL): addresses
? Common model or server only? common
Let's add some Address properties now.
```

Enter an empty property name when done.

```
? Property name: streetname
  invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```

Let's add another Address property.

Enter an empty property name when done.

```
? Property name: city
  invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```

Let's add another Address property.

Enter an empty property name when done.

```
? Property name: state
  invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```

Let's add another Address property.

Enter an empty property name when done.

```
? Property name: country
```

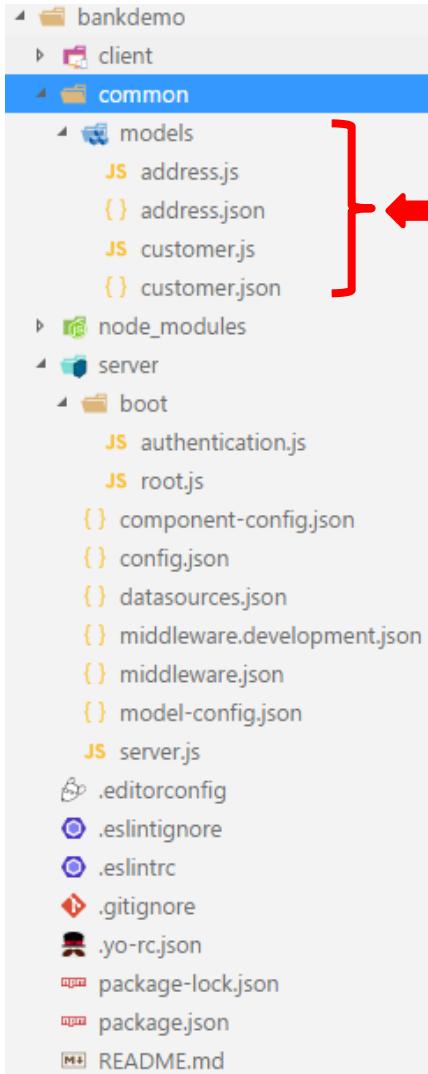
PersistedModel is the base object for all models connected to a persistent data source such as a database



BankDemo verifying common models created

Please read terms of use for authorized access

Original Series



- create **common/models** folder inside bankdemo

Stores Custom model files:

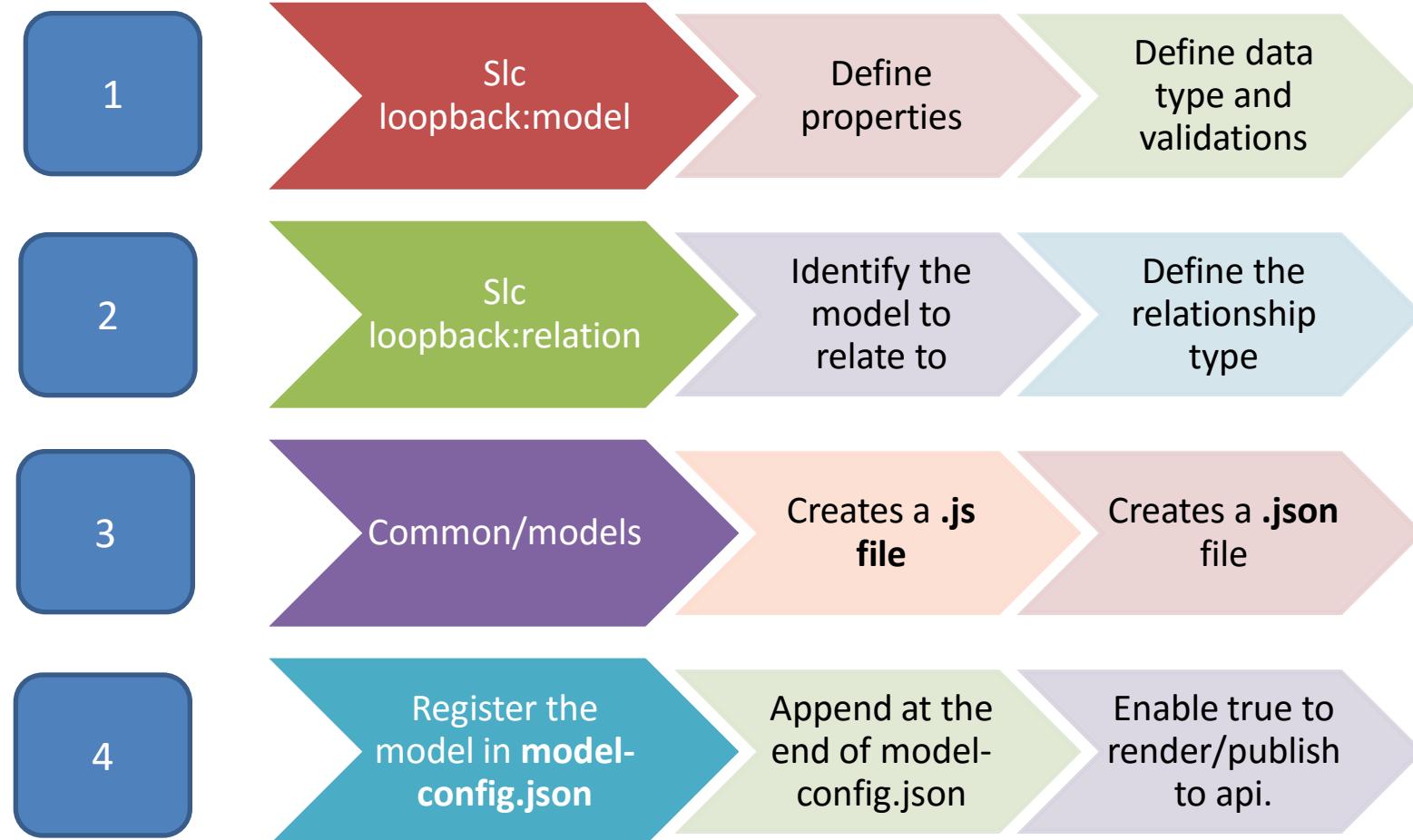
- Model definition JSON files, by convention named *model-name.json*; for example customer.json.
- Custom model scripts by convention named *model-name.js*; for example, customer.js.



LoopBack models workflow

Please read terms of use for authorized access

Original Series





Step 4

slc loopback:relation

- A customer “**has many**” addresses.
- loopback enables us to define relationship between two entities using slc loopback:relation as shown in the figure.

```
PS E:\CT\loopback\codeplay\bankdemo> slc loopback:relation
? Select the model to create the relationship from: Customer
? Relation type: has many
? Choose a model to create a relationship with: Address
? Enter the property name for the relation: addresses
? Optionally enter a custom foreign key:
? Require a through model? No
```



Step 5

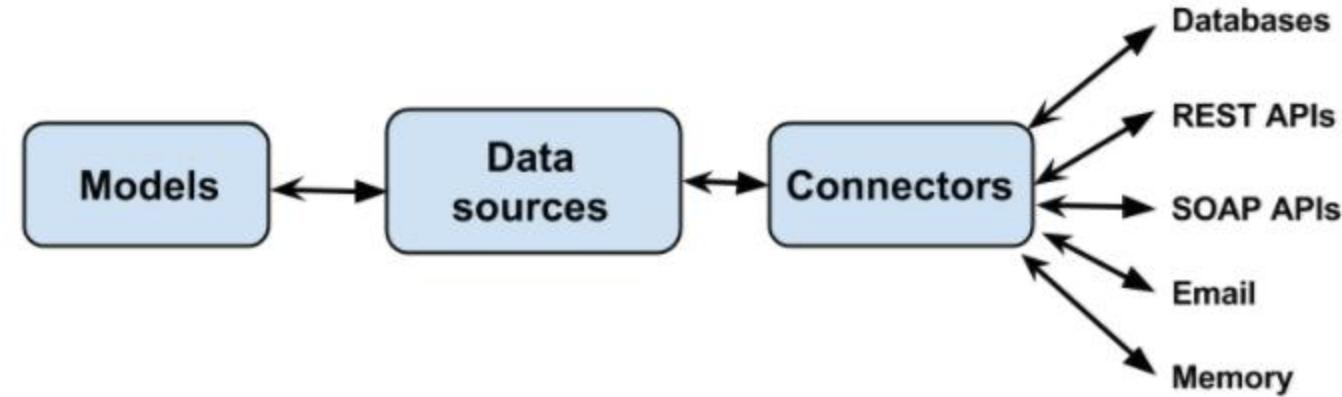
Running your application

- Now run your application
- cd bankdemo
- Node .

```
PS E:\CT\loopback\codeplay\bankdemo> node .
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

<http://localhost:3000/explorer/#/>

The screenshot shows the LoopBack API Explorer interface. At the top, there is a navigation bar with links like Java Topics & Books, Apps, Learn ASP.NET MVC, ViewData in ASP.NET, Stack Overflow Blog, Sean Lahman | Database, Retrosheet Event File, Youtube Multi Downloader, and Working with Geography. Below the navigation bar, the title "LoopBack API Explorer" is displayed, along with a "Token Not Set" message and a "accessToken" input field with a "Set Access Token" button. The main content area is titled "bankdemo". It lists three models: "Address", "Customer", and "User". Each model has a "Show/Hide" link, a "List Operations" link, and an "Expand Operations" link. At the bottom of the page, it says "[BASE URL: /api , API VERSION: 1.0.0]".



ONLINE BANKING WITH MYSQL AND LOOPBACK-DISCOVERY

SLC LOOPBACK:DEMO-V



ONLINE BANKING DEMO

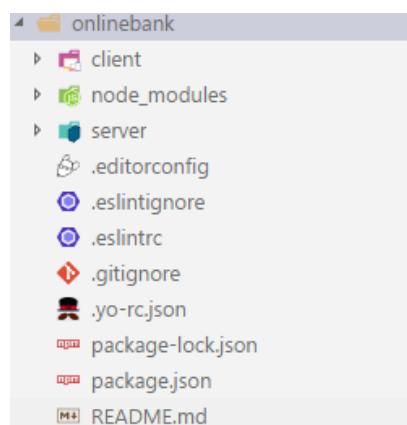
Please read terms of use for authorized access

Original Series

Step 1

Create API server project

- create API server project using `scl loopback` as shown in the figure using current version 3.x



Next steps:

Change directory to your app
`$ cd onlinebank`

Create a model in your app
`$ scl loopback:model`

Run the app
`$ node .`

```
PS E:\CT\loopback\codeplay> scl loopback
```



```
? What's the name of your application? onlinebank
? Enter name of the directory to contain the project: onlinebank
  create onlinebank/
    info change the working directory to onlinebank

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? api-server (A LoopBack API server with local User auth)
Generating .yo-rc.json
```

I'm all done. Running `npm install` for you to install the required dependencies. If this fails, try running the command yourself.

```
create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create README.md
create server\boot\authentication.js
create .gitignore
create client\README.md
npm WARN deprecated nodemailer@2.7.2: All versions below 4.0.1 of Nodemailer are deprecated. See https://nodemailer.com/status/
[ ..... ] | fetchMetadata: http fetch GET 304 https://registry.npmjs.org/arrify 232ms (from cache)
```

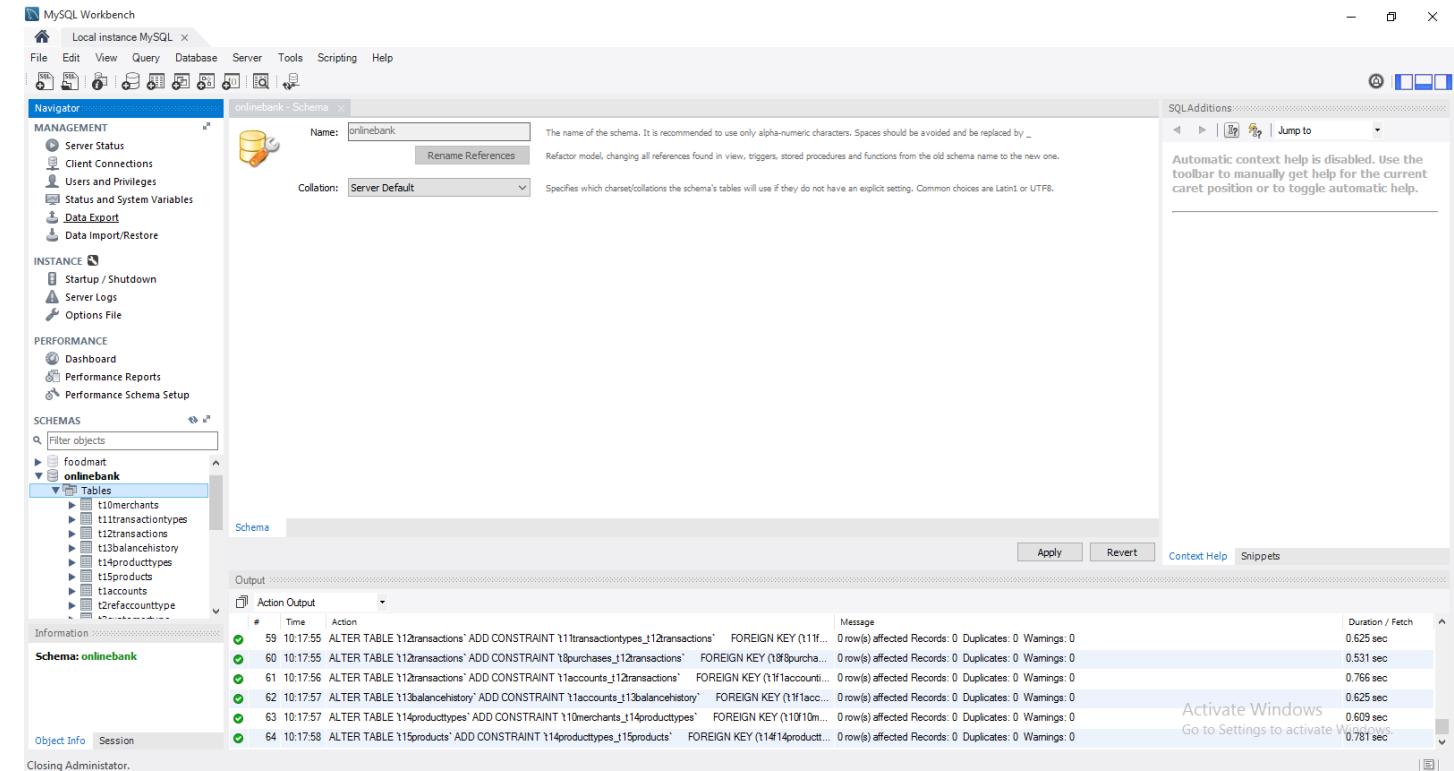


ONLINE BANKING DEMO

Step 2

Create MySQL database

- create a database onlinebank in MySQL5 as shown in the figure.



Schema Script:

<https://github.com/awasekhirni/dbmodels/blob/master/mysql/onlinebank-mysql5-create.sql>



ONLINE BANKING DEMO

Step 3

Configure datasource

- `slc loopback:datasource`
- To install and configure the connector for mysql using loopback.

```
PS E:\CT\loopback\codeplay\onlinebank> slc loopback:datasource
? Enter the data-source name: onlinebank
? Select the connector for onlinebank: MySQL (supported by StrongLoop)
Connector-specific configuration:
? Connection String url to override other settings (eg: mysql://user:pass@host/db): mysql://mysqlusr:lny786@localhost/onlinebank
ql://user:pass@host/db): mysql://mysqlusr:lny786@localhost/
onlinebank
? host: localhost
? port: 3306
? user: mysqlusr
? password: *****
? database: onlinebank
? Install loopback-connector-mysql@^2.2 Yes
```

```
1  {
2    "db": {
3      "name": "db",
4      "connector": "memory"
5    },
6    "onlinebank": {
7      "host": "localhost",
8      "port": 3306,
9      "url": "mysql://mysqlusr:test123@localhost/onlinebank",
10     "database": "onlinebank",
11     "password": "test123",
12     "name": "onlinebank",
13     "user": "mysqlusr",
14     "connector": "mysql"
15   }
16 }
17 }
```



Step 4

Install loopback-discovery

- globally install loopback-discovery npm packages.
- Post installation use `loopback-discovery -a -v -d datasourcename` to navigate and select the tables, views which would be needed to generate API

```
npm install -g loopback-discovery
```

```
PS E:\CT\loopback\codeplay\onlinebank> loopback-discovery -a -v -d onlinebank
Save definition for model: T10merchants
Save definition for model: T11transactiontypes
Save definition for model: T12transactions
Save definition for model: T13balancehistory
Save definition for model: T14producttypes
Save definition for model: T15products
Save definition for model: T1accounts
Save definition for model: T2refaccounttype
Save definition for model: T3customertype
Save definition for model: T4customers
.....
```



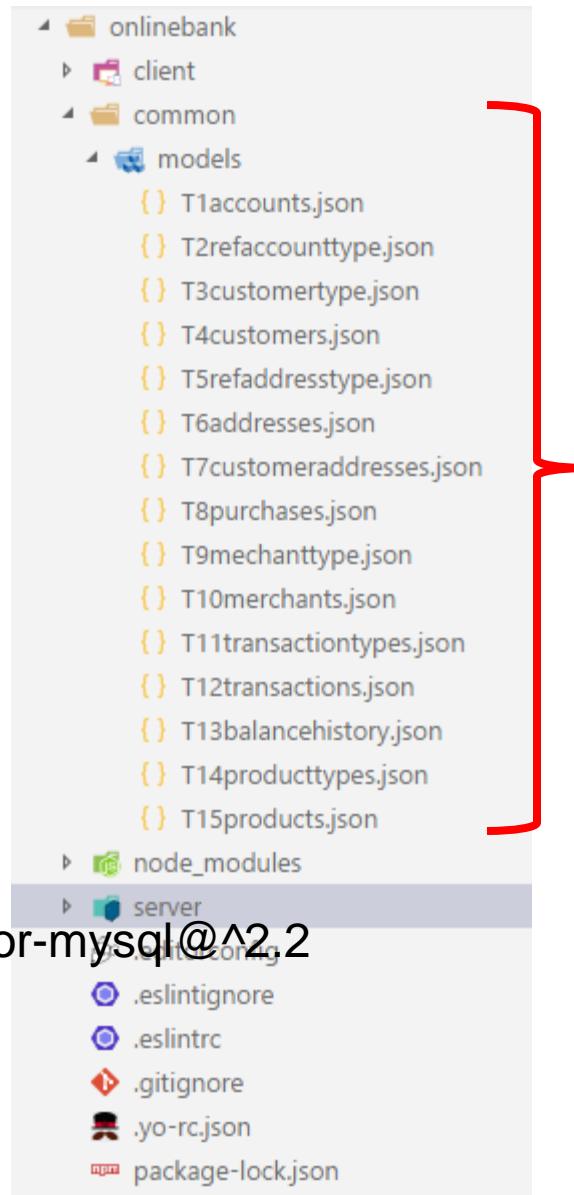
ONLINE BANKING DEMO

Step 5

Verify the models generated

- Now let us verify the models that have been generated using loopback-discovery as shown in the figure.

```
npm install --save loopback-connector-mysql@^2.2
```





ONLINE BANKING DEMO

Step 6

Running LoopBack Application

- run your loopback application using node .

<http://localhost:3000/explorer/#/>

The screenshot shows the LoopBack API Explorer interface at the URL <http://localhost:3000/explorer/#/>. The title bar says "LoopBack API Explorer". A search bar contains "accessToken". The main area displays the "onlinebank" namespace with the following models and their operations:

Model	Show/Hide	List Operations	Expand Operations
T10merchants	Show/Hide	List Operations	Expand Operations
T11transactiontypes	Show/Hide	List Operations	Expand Operations
T12transactions	Show/Hide	List Operations	Expand Operations
T13balancehistory	Show/Hide	List Operations	Expand Operations
T14producttypes	Show/Hide	List Operations	Expand Operations
T15products	Show/Hide	List Operations	Expand Operations
T1accounts	Show/Hide	List Operations	Expand Operations
T2refaccounttype	Show/Hide	List Operations	Expand Operations
T3customertype	Show/Hide	List Operations	Expand Operations
T4customers	Show/Hide	List Operations	Expand Operations
T5refaddresstype	Show/Hide	List Operations	Expand Operations
T6addresses	Show/Hide	List Operations	Expand Operations
T7customeraddresses	Show/Hide	List Operations	Expand Operations



POINT OF SALES APPLICATION WITH POSTGRESQL

SLC LOOPBACK:DEMO-VI



ONLINE POINT OF SALE(POS)

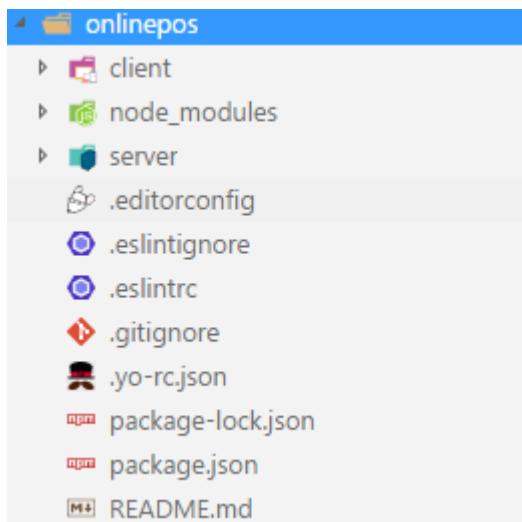
Please read terms of use for authorized access

Original Series

Step 1

Create API server project

- create API server project using slc loopback as shown in the figure using current version 3.x



PS E:\CT\loopback\codeplay> `slc loopback`



```
? What's the name of your application? onlinepos
? Enter name of the directory to contain the project: onlinepos
  create onlinepos/
    info change the working directory to onlinepos

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? api-server (A LoopBack API server with local User auth)
Generating .yo-rc.json
```

I'm all done. Running `npm install` for you to install the required dependencies. If this fails, try running the command yourself.

```
create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create README.md
create server\boot\authentication.js
create .gitignore
create client\README.md
[ ..... ] | fetchMetadata: sill pacote range manifest for strong-globalize@^2.6.2 fetched in 150ms
```

Next steps:

Change directory to your app
\$ `cd onlinepos`

Create a model in your app
\$ `slc loopback:model`

Run the app
\$ `node .`

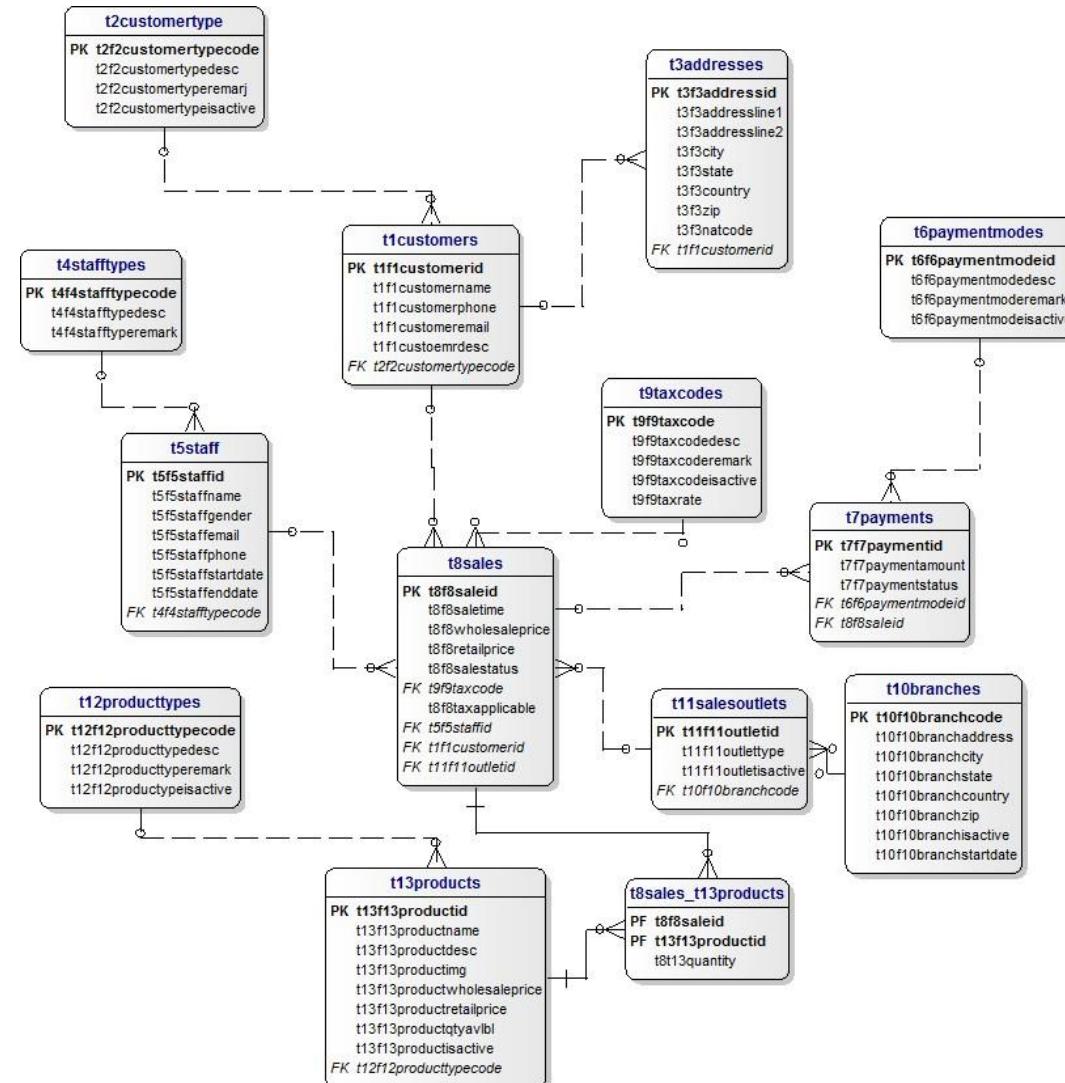


ONLINE POINT OF SALE(POS)

Step 2

Create PostgreSQL database

- create schema for onlinepost for postgresql



<https://github.com/awasekhirni/dbmodels/blob/master/postgres/pos-postgres83.jpg>

Please read terms of use for authorized access

Original Series



ONLINE POINT OF SALE(POS)

Step 3

Create PostgreSQL database

- create a database based on the erd diagram shown in the earlier step.

Properties

Property	Value
Name	onlinepos
OID	16384
Owner	postgres
ACL	pg_default
Tablespace	pg_default
Encoding	UTF8
Collation	English_United States.1252
Character type	English_United States.1252
Default schema	public
Default table ACL	
Default sequence ACL	
Default function ACL	
Default type ACL	
Allow connections?	Yes
Connected?	Yes
Connection limit	-1
System database?	No
Comment	

SQL pane

```
-- Database: onlinepos
-- DROP DATABASE onlinepos;

CREATE DATABASE onlinepos
WITH OWNER = postgres
ENCODING = 'UTF8'
TABLESPACE = pg_default
LC_COLLATE = 'English_United States.1252'
LC_CTYPE = 'English_United States.1252'
CONNECTION LIMIT = -1;
```

<https://github.com/awasekhirni/dbmodels/blob/master/postgres/pos-postgres83-create.sql>



ONLINE POINT OF SALE(POS)

Please read terms of use for authorized access

Original Series

Step 4

Configure your datasource

- configure and install the loopback connector for postgresql as shown in the figure.

```
PS E:\CT\loopback\codeplay\onlinepos> slc loopback:datasource
? Enter the data-source name: onlinepos
? Select the connector for onlinepos: PostgreSQL (supported by StrongLoop)
Connector-specific configuration:
? Connection String url to override other settings (eg: postgres://username:password@localhost/database): postgres://postgres:@localhost/onlinepos
? host: localhost
? port: 5432
? user: postgres
? password:
? database: onlinepos
? Install loopback-connector-postgresql@^2.4 (Y/n) Y
```

```
{
  "db": {
    "name": "db",
    "connector": "memory"
  },
  "onlinepos": {
    "host": "localhost",
    "port": 5432,
    "url": "postgres://postgres:@localhost/onlinepos",
    "database": "onlinepos",
    "password": "",
    "name": "onlinepos",
    "user": "postgres",
    "connector": "postgresql"
  }
}
```



ONLINE POINT OF SALE(POS)

Step 5

Write custom discover and build models

- Create directory /server/bin/
- Write the file discover-and-build.js to discover the schema from postgresql database

```

OPEN EDITORS
  [] datasources.json onlinepos\server
  JS discovery-and-build.js onlinepos\server...
CODEPLAY
  MODELS
    JS t1customers.js
    () t1customers.json
    JS t2customertype.js
    () t2customertype.json
    JS t3addresses.js
    () t3addresses.json
  node_modules
  SERVER
    bin
      JS discovery-and-build.js
      boot
        () component-config.json
        () config.json
        () datasources.json
        () middleware-development.json
        () middleware.json
        () model-config.json
        JS server.js
        .editorconfig
        .eslintignore
        .eslintrc
        .gitignore
        .yo-rc.json
        package-lock.json
        package.json
        README.md
      testdemo
      tridemo
      truestartdemo
PROJECTS
datasources.json JS discovery-and-build.js x
1 var path = require('path');
2 var fs = require('fs');
3 var app = require('loopback');
4 var output_directory = path.resolve(__dirname, '.', '..', 'common', 'models');

5
6 function callback(err, schema) {
7   if (err) {
8     console.error(err);
9     return;
10 }
11 if (typeof schema != 'object') {
12   throw 'schema object not defined';
13 }
14 console.log("Auto discovery for schema " + schema.name);
15 /*
16 * Convert schema name from CamelCase to dashed lowercase (loopback format
17 * for json files describing models), for example: CamelCase -> camel-case.
18 */
19 //var model_name = schema.name.replace(/([a-z])([A-Z])/g, '$1-$2').toLowerCase();
20 var model_name = modelName(null, schema.name);

21
22 console.log('Writing model JSON file..');
23 // write model definition JSON file
24 var now_ms = Date.now();
25 var model_JSON_file = path.join(output_directory, model_name + '.json');
26 // if JSON file exists
27 if (fs.existsSync(model_JSON_file)) {
28   // save a backup copy of the JSON file
  
```

Activate Windows
Go to Settings to activate Windows.

▶ Launch Program ⌂ Scanning... ▶ 00:00 235 Python 3.6.0 00:25:00 (1/8) Ln 22, Col 44 Spaces: 2 UTF-8 LF JavaScript 😊

<https://github.com/awasekhirni/dw/blob/master/loopback/discover-and-build.js>



ONLINE POINT OF SALE(POS)

Please read terms of use for authorized access

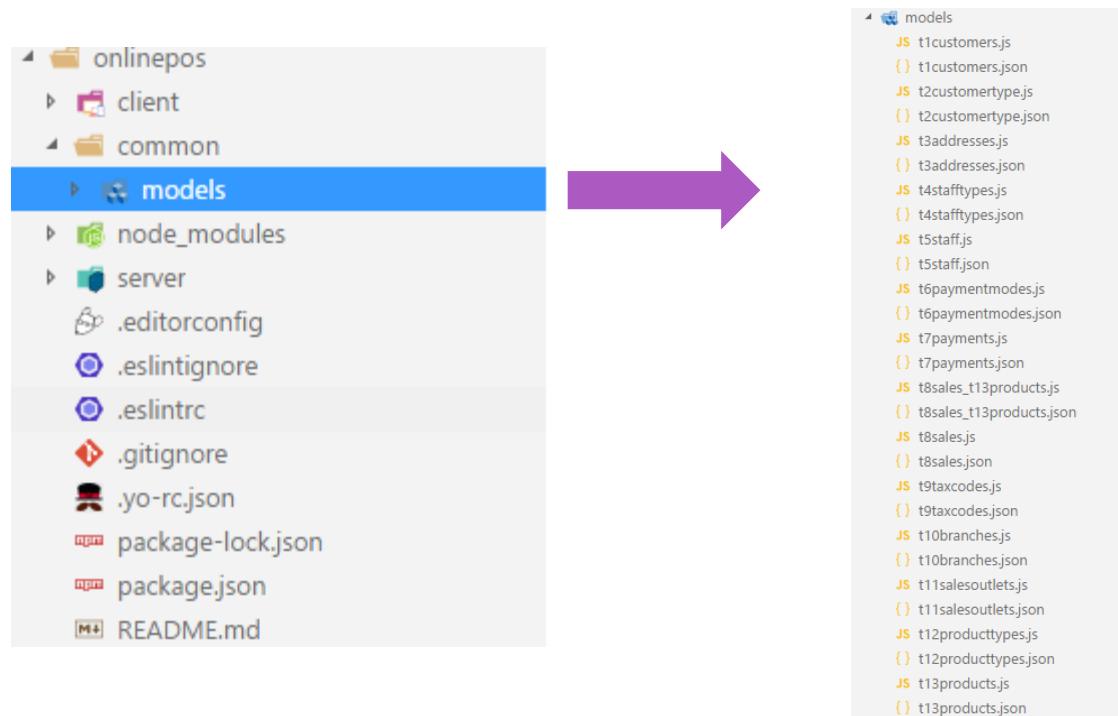
Original Series

Step 6

Discover and build individual models.

- Run discover and build individual models to create models from the database schema => onlinepos
- Repeat the process for all tables in schema

```
PS E:\CT\loopback\codeplay\onlinepos> node .\server\bin\discovery-and-build.js -ds onlinepos -sn t1customers
Auto discovery for schema t1customers
Writing model JSON file..
JSON saved to E:\CT\loopback\codeplay\onlinepos\common\models\t1customers.json
Writing model JS file..
JSON saved to E:\CT\loopback\codeplay\onlinepos\common\models\t1customers.json
```



`node .\server\bin\discovery-and-build.js -ds onlinepos -sn t1customers`



ONLINE POINT OF SALE(POS)

Step 7

Configure model-config.json

- configure model-config.json, by toggling the models generated by **discovery-and-build** process from 'false' to 'true' to render in the swagger API.

```
"t13products": {
  "dataSource": "onlinepos",
  "public": false
}
```

➡

```
"t13products": {
  "dataSource": "onlinepos",
  "public": true
}
```

```
EXPLORER
OPEN EDITORS
codeplay-notes.txt
model-config.json onlinepos\server
CODEPLAY
  t13salesoutlets.js
  t11salesoutlets.json
  t12producttypes.js
  t12producttypes.json
  t13products.js
  t13products.json
  node_modules
    server
      bin
        discover-schema-basic.js
        discovery-and-build.js
      boot
        component-config.json
        config.json
        datasources.json
        middleware.development.json
        middleware.json
        model-config.json
        server.js
      .editorconfig
      .eslintignore
      .eslintrc
      .gitignore
      .yo-rc.json
      package-lock.json
      package.json
      README.md
    testdemo
    tpridemo
    truestatedemo
PROJECTS
```

```
datasource : onlinepos ,
"public": true
},
"t7payments": {
  "dataSource": "onlinepos",
  "public": true
},
"t8sales": {
  "dataSource": "onlinepos",
  "public": true
},
"t8sales_t13products": {
  "dataSource": "onlinepos",
  "public": true
},
"t9taxcodes": {
  "dataSource": "onlinepos",
  "public": true
},
"t10branches": {
  "dataSource": "onlinepos",
  "public": true
},
"t11salesoutlets": {
  "dataSource": "onlinepos",
  "public": true
},
"t12producttypes": {
  "dataSource": "onlinepos"
}
```



ONLINE POINT OF SALE(POS)

Step 7

Run your loopback application

- Run your loopback application with **node** .

```
PS E:\CT\loopback\codeplay\onlinepos> node .
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

<http://localhost:3000/explorer/#/>

The screenshot shows the LoopBack API Explorer interface. At the top, it displays the URL `localhost:3000/explorer/#/t13products`. Below the header, there's a navigation bar with links like Java Topics & Books, Apps, Learn ASP.NET MVC, ViewData in ASP.NET, Stack Overflow Blog, Sean Lahman | Database, Retrosheet Event File, Youtube Multi Downloader, Working with Geography, and a star icon. The main content area is titled "onlinepos". It lists several models: t10branches, t11salesoutlets, t12producttypes, and t13products. For each model, it provides "Show/Hide", "List Operations", and "Expand Operations" buttons. The t13products section is expanded, showing various HTTP methods and their descriptions:

Method	Endpoint	Description
PATCH	/t13products	Patch an existing model instance or insert a new one into the data source.
GET	/t13products	Find all instances of the model matched by filter from the data source.
PUT	/t13products	Replace an existing model instance or insert a new one into the data source.
POST	/t13products	Create a new instance of the model and persist it into the data source.
PATCH	/t13products/{id}	Patch attributes for a model instance and persist it into the data source.
GET	/t13products/{id}	Find a model instance by {{id}} from the data source.
HEAD	/t13products/{id}	Check whether a model instance exists in the data source.
PUT	/t13products/{id}	Replace attributes for a model instance and persist it into the data source.



UBER CAR INVOICE WITH MONGODB

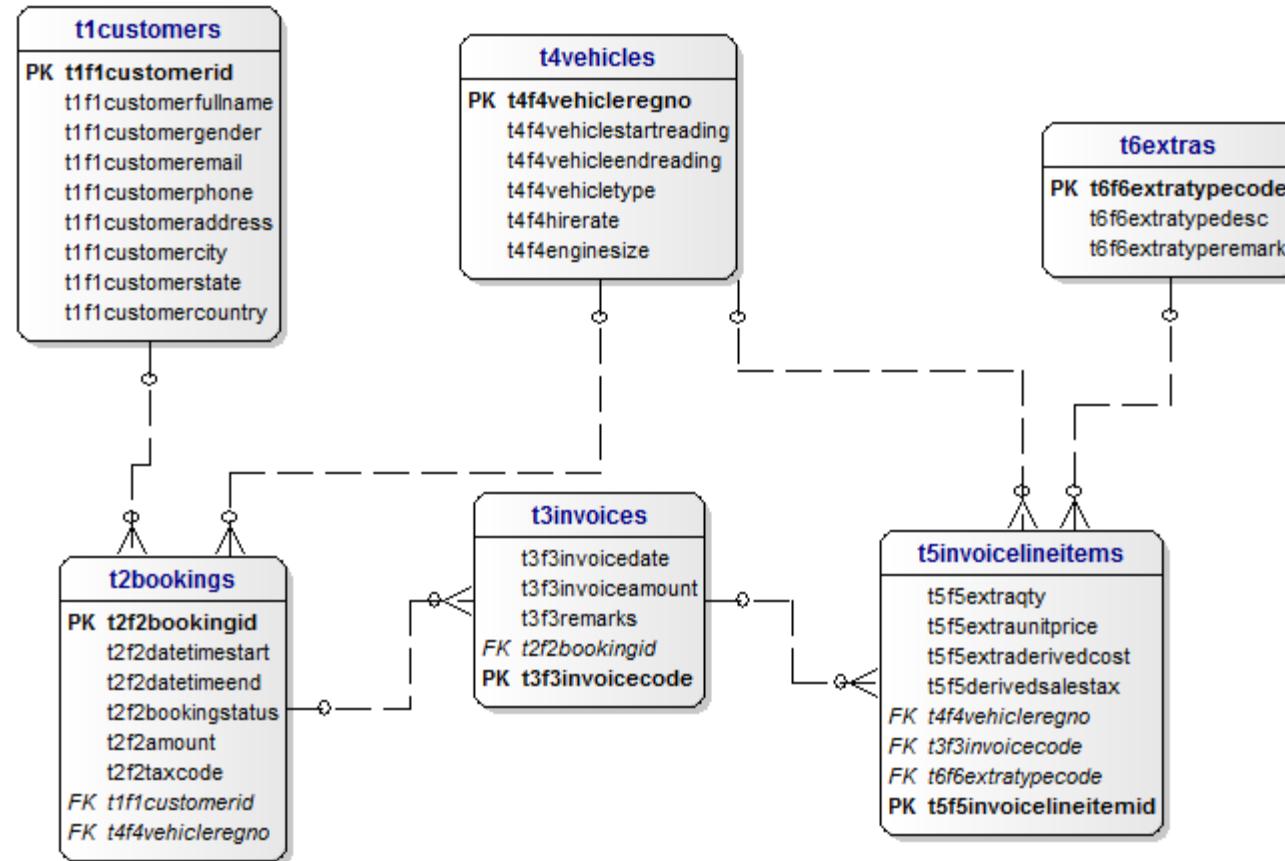
SLC LOOPBACK:DEMO-VII



UBER TRAVEL INVOICE

Please read terms of use for authorized access

Original Series





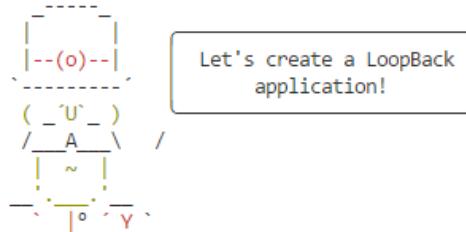
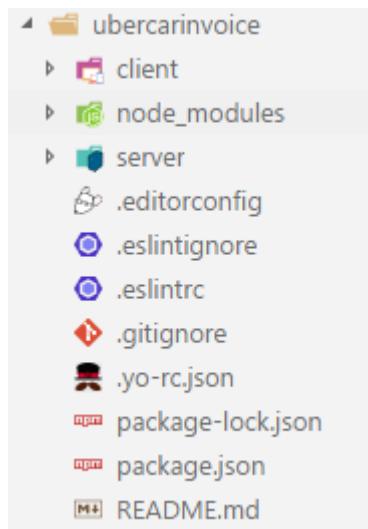
UBER CAR INVOICE(MONGODB)

PS E:\CT\loopback\codeplay> `scl loopback ubercarinvoice`

Step 1

Create API server project

- create API server project using `scl loopback` as shown in the figure using current version 3.x



```
? What's the name of your application? ubercarinvoice
? Enter name of the directory to contain the project: ubercarinvoice
  create ubercarinvoice/
    info change the working directory to ubercarinvoice

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? api-server (A LoopBack API server with local User auth)
Generating .yo-rc.json

I'm all done. Running npm install for you to install the required dependencies. If this fails, try running the command yourself.
```

```
create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create README.md
create server\boot\authentication.js
create .gitignore
create client\README.md
```

Next steps:

Change directory to your app
\$ `cd ubercarinvoice`

Create a model in your app
\$ `scl loopback:model`

Run the app
\$ `node .`



UBER CAR INVOICE(MONGODB)

Step 2

Configure datasource

- create datasource connector for mongodb as shown in the figure.

```
PS E:\CT\loopback\codeplay\ubercarinvoice> slc loopback:datasource
? Enter the data-source name: mongoconnector
? Select the connector for mongoconnector: MongoDB (supported by StrongLoop)
Connector-specific configuration:
? Connection String url to override other settings (eg: mongodb://username:password@hostname:port/database): mongodb://@localhost:27017/ubercardb
? host: localhost
? port: 27017
? user:
? password:
? database: ubercardb
? Install loopback-connector-mongodb@^1.4 Yes
```

```
1  {
2   "db": {
3     "name": "db",
4     "connector": "memory"
5   },
6   "mongoconnector": {
7     "host": "localhost",
8     "port": 27017,
9     "url": "mongodb://@localhost:27017/ubercardb",
10    "database": "ubercardb",
11    "password": "",
12    "name": "mongoconnector",
13    "user": "",
14    "connector": "mongodb"
15  }
16 }
```

Run the app
\$ node .

```
PS E:\CT\loopback\codeplay> cd ..\ubercarinvoice
PS E:\CT\loopback\codeplay\ubercarinvoice> slc loopback:datasource
? Enter the data-source name: mongoconnector
? Select the connector for mongoconnector: MongoDB (supported by StrongLoop)
Connector-specific configuration:
? Connection String url to override other settings (eg: mongodb://username:password@hostname:port/database): mongodb://@localhost:27017/ubercardb
? host: localhost
? port: 27017
? user:
? password:
? database: ubercardb
? Install loopback-connector-mongodb@^1.4 Yes
```

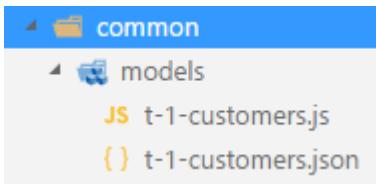


UBER CAR INVOICE(MONGODB)

Step 3

Create models

- create models based on the erd diagram for usedcar invoice.



codeplay-notes.txt JS t-1-customers.js {} t-1-customers.json

```

1  'use strict';
2  module.exports = function(T1customers) {
3    //remote methods come here
4  };
    
```

```

PS E:\CT\loopback\codeplay\ubercarinvoice> slc loopback:model
del
? Enter the model name: t1customers
? Select the data-source to attach t1customers to: mongoc
onnection (mongodb)
? Select model's base class PersistedModel
? Expose t1customers via the REST API? Yes
? Custom plural form (used to build REST URL): t1customers
? Common model or server only? common
Let's add some t1customers properties now.
    
```

Enter an empty property name when done.

```

? Property name: t1f1customerid
  invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
    
```

```

codeplay-notes.txt   JS t-1-customers.js   {} t-1-customers.json
34   "t1f1customercity": {
35     "type": "string",
36     "required": true
37   },
38   "t1f1customerstate": {
39     "type": "string",
40     "required": true
41   },
42   "t1f1customercountry": {
43     "type": "string",
44     "required": true
45   }
46 },
47 "validations": [],
48 "relations": {},
49 "acls": [],
50 "methods": {}
51 }
52 
```

t1customers
PK t1f1customerid
t1f1customerfullname
t1f1customergender
t1f1customeremail
t1f1customerphone
t1f1customeraddress
t1f1customercity
t1f1customerstate
t1f1customercountry

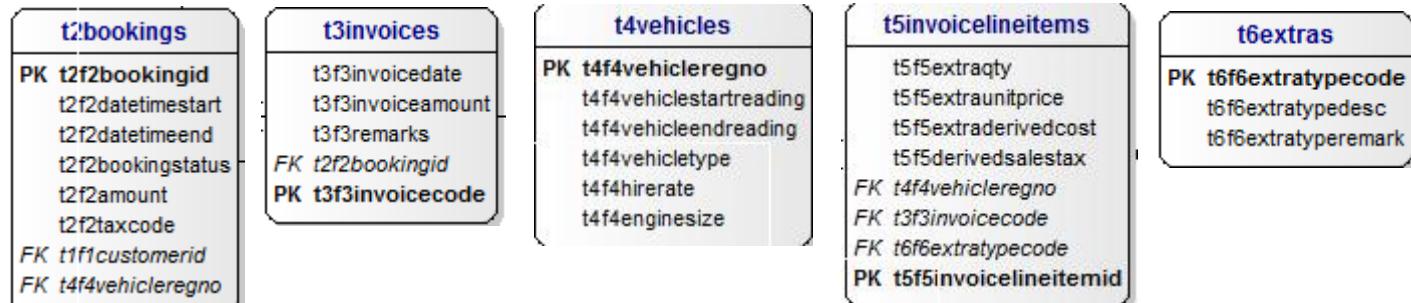
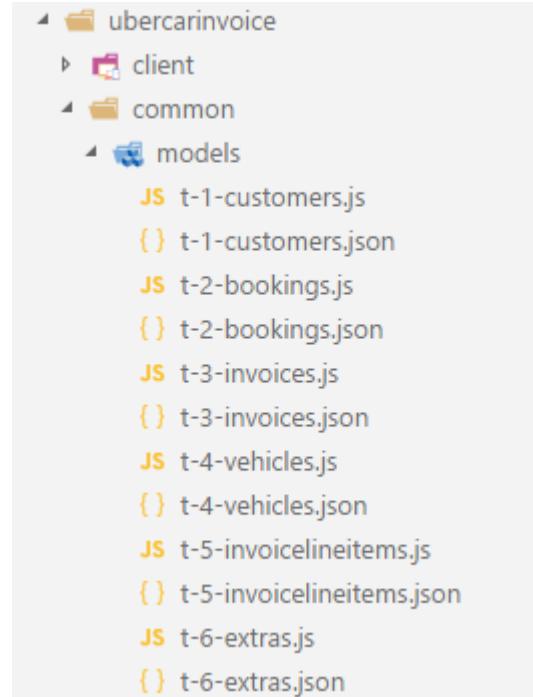


UBER CAR INVOICE(MONGODB)

Step 4

Create models

- create models using slc
loopback:model



`model-config.json`

```

32     },
33     },
34     "Role": {
35         "dataSource": "db",
36         "public": false
37     },
38     "t1customers": {
39         "dataSource": "mongoconnector",
40         "public": true
41     },
42     "t2bookings": {
43         "dataSource": "mongoconnector",
44         "public": true
45     },
46     "t3invoices": {
47         "dataSource": "mongoconnector",
48         "public": true
49     },
50     "t4vehicles": {
51         "dataSource": "mongoconnector",
52         "public": true
53     },
54     "t6extras": {
55         "dataSource": "mongoconnector",
56         "public": true
57     },
58     "t5invoicelineitems": {
59         "dataSource": "mongoconnector",

```

Models registered in
model-config.json



UBER CAR INVOICE(MONGODB)

Step 5

Establish relationship

- create relationship between models using slc loopback: relation

```
PS E:\CT\loopback\codeplay\ubercarinvoice> slc loopback:relation
? Select the model to create the relationship from: t1customers
? Relation type: has many
? Choose a model to create a relationship with: t2bookings
? Enter the property name for the relation: t2bookings
? Optionally enter a custom foreign key:
? Require a through model? No
```

Please read terms of use for authorized access

Original Series

```
{
  "t1customers": {
    "customerCountry": {
      "type": "string",
      "required": true
    },
    "relations": {
      "t2bookings": {
        "type": "hasMany",
        "model": "t2bookings",
        "foreignKey": ""
      }
    },
    "acls": [],
    "methods": {}
  }
}
```

```
PS E:\CT\loopback\codeplay\ubercarinvoice> slc loopback:relation
? Select the model to create the relationship from: t4vehicles
? Relation type: has many
? Choose a model to create a relationship with: t2bookings
? Enter the property name for the relation: t2bookings
? Optionally enter a custom foreign key:
? Require a through model? No
```



UBER CAR INVOICE(MONGODB)

Please read terms of use for authorized access

Original Series

Step 6

Establish relationship

- create relationship between models using `slc loopback: relation`

```
PS E:\CT\loopback\codeplay\ubercarinvoice> slc loopback:relation
? Select the model to create the relationship from: t2bookings
? Relation type: has many
? Choose a model to create a relationship with: t3invoices
? Enter the property name for the relation: t3invoices
? Optionally enter a custom foreign key:
? Require a through model? No
```

```
PS E:\CT\loopback\codeplay\ubercarinvoice> slc loopback:relation
? Select the model to create the relationship from: t3invoices
? Relation type: has many
? Choose a model to create a relationship with: t5invoicelineitems
? Enter the property name for the relation: t5invoicelineitems
? Optionally enter a custom foreign key:
? Require a through model? No
```

```
PS E:\CT\loopback\codeplay\ubercarinvoice> slc loopback:relation
? Select the model to create the relationship from: t4vehicles
? Relation type: has many
? Choose a model to create a relationship with: t5invoicelineitems
? Enter the property name for the relation: t5invoicelineitems
? Optionally enter a custom foreign key:
? Require a through model? No
```



UBER CAR INVOICE(MONGODB)

Step 7

Create Account Model for User Authentication

- create account model by extending user model's base class.

Please read terms of use for authorized access

Original Series

```
PS E:\CT\loopback\codeplay\ubercarinvoice> slc loopback:model
? Enter the model name: Account
? Select the data-source to attach Account to: mongoconnector (mongodb)
? Select model's base class User
? Expose Account via the REST API? Yes
? Custom plural form (used to build REST URL): accounts
? Common model or server only? common
Let's add some Account properties now.
```

Enter an empty property name when done.

? Property name:

```

EXPLORER
OPEN EDITORS
account.json ubercarinvoice\common\m...
CODEPLAY
onlinepos
testdemo
tpridemo
truestatedemo
ubercarinvoice
client
common
models
account.js
account.json
t-1-customers.js
t-1-customers.json
t-2-bookings.js
t-2-bookings.json
t-3-invoices.js

account.json
1 {
2   "name": "Account",
3   "plural": "accounts",
4   "base": "User",
5   "idInjection": true,
6   "options": {
7     "validateUpsert": true
8   },
9   "properties": {},
10  "validations": [],
11  "relations": {},
12  "acls": [],
13  "methods": {}
14 }
```



UBER CAR INVOICE(MONGODB)

Step 8

ACL Enabling and Authentication

- Check boot/authentication.js
- Call slc loopback:acl generator for generators for access control list.
- Acls are added to all the existing models.

```
PS E:\CT\loopback\codeplay\ubercarinvoice> slc loopback:acl
? Select the model to apply the ACL entry to: (all existing models)
? Select the ACL scope: All methods and properties
? Select the access type: All (match all types)
? Select the role Any authenticated user
? Select the permission to apply Explicitly_deny access
```

{ account.json x

```
10  "validations": [],
11  "relations": {},
12  "acls": [
13    {
14      "accessType": "*",
15      "principalType": "ROLE",
16      "principalId": "$authenticated",
17      "permission": "DENY"
18    }
19  ],
20  "methods": {}
21 }
```



UBER CAR INVOICE(MONGODB)

Step 9

Create a new account

- access the loopback api explorer and using post method register a new account as shown in the figure.

The screenshot shows the LoopBack API Explorer interface. The URL is `localhost:3000/explorer#!/Account/Account_login`. The main title is "LoopBack API Explorer". The top navigation bar includes links for Java Topics & Books, Apps, Learn ASP.NET MVC, ViewData in ASP.NET, Stack Overflow Blog, Sean Lahman | Database, Retrosheet Event File, Youtube Multi Downloader, Working with, and a search bar for "accessToken" with a "Set Access Token" button. A message at the top right says "Token Not Set". Below the title, it says "POST /accounts" and "Create a new instance of the model and persist it into the data source." The "Model" tab is selected, showing the JSON schema for the account model:

```
{  
  "realm": "string",  
  "username": "string",  
  "email": "string",  
  "emailVerified": true,  
  "id": "string"  
}
```

The "Parameters" section shows a table with columns: Parameter, Value, Description, Parameter Type, and Data Type. There is one row for the "data" parameter, which is set to a JSON object:

```
{  
  "realm": "string",  
  "username": "syedawase",  
  "email": "awasekhirni@gmail.com",  
  "emailVerified": true,  
  "password": "test123"  
}
```

The "Parameter content type:" dropdown is set to "application/json". To the right of the table, there is another JSON object with the same schema:

```
{  
  "realm": "string",  
  "username": "string",  
  "email": "string",  
  "emailVerified": true,  
  "id": "string"  
}
```

Please read terms of use for authorized access

Original Series



UBER CAR INVOICE(MONGODB)

Step 10

Login to loopback Explorer

- login to loopback explorer using the newly created credentials.

Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \"email\":\"awasekhirni%4@gmail.com\", \"password\":\"test123\" }' http://localhost:3000/api/accounts/login
```

Request URL

http://localhost:3000/api/accounts/login

Response Body

```
{
  "id": "Nin5MHleZm4TK1Y1MCJaUDFbBbg5zOiRWHfX2BEu4qQLPDxMfGcKxbvmpgDff4ds",
  "ttl": 1209600,
  "created": "2017-12-10T10:38:36.148Z",
  "userId": "5a2d0da6ce15833b6c3a0aa7"
}
```

Response Code

200

The screenshot shows the LoopBack API Explorer interface. At the top, it displays the URL `localhost:3000/explorer/#!/Account/Account_login`. Below the header, there are two main sections: one for a GET request to `/accounts/findOne` and another for a POST request to `/accounts/login`. The POST section is highlighted. It shows the response class as Status 200 and indicates the request was successful. The parameters section shows a parameter named `credentials` with a value of `{"email":"awasekhirni@gmail.com", "password":"test123"}`. The response content type is set to `application/json`.



UBER CAR INVOICE(MONGODB)

Step 11

Applying access token

- Now apply the access token post creation of an account.
- You can now be able to create a new customer record.

localhost:3000/explorer/#!/t1customers/t1customers_create

Core Java Topics & Books Apps Learn ASP.NET MVC ViewData in ASP.NET Stack Overflow Blog Sean Lahman | Database Retrosheet Event File Youtube Multi Downloader Working with CodeIgniter

LoopBack API Explorer

Token Set: Nin5MHleZm4TK1Y1MCJaUDFbBbg5zO1RWhfx2BEu4qQLPDxMfGcKxbvmpgDff4dS | Set Access Token

Request URL
http://localhost:3000/api/t1customers?access_token=Nin5MHleZm4TK1Y1MCJaUDFbBbg5zO1RWhfx2BEu4qQLPDxMfGcKxbvmpgDff4dS

Response Body

```
{
  "t1fcustomerid": "a1",
  "t1fcustomerfullname": "Syed Rayyan",
  "t1fcustomergender": "male",
  "t1fcustomeremail": "rayyan@gmail.com",
  "t1fcustomerphone": "9999191912",
  "t1fcustomeraddress": "227 hrbr",
  "t1fcustomercity": "bangalore",
  "t1fcustomerstate": "karnataka",
  "t1fcustomercountry": "india",
  "id": "5a2d13ccdc037125bc06b877"
}
```

Response Code
200

Response Headers

```
{
  "date": "Sun, 10 Dec 2017 11:00:28 GMT",
  "x-content-type-options": "nosniff",
  "etag": "W/\"13e-GnFNkPBtlDcra47BNBoKhDzfys\"",
  "x-download-options": "nopopen",
  "x-frame-options": "DENY",
  "x-xss-filter": "Enabled"
}
```

Please read terms of use for authorized access

Original Series



UBER CAR INVOICE(MONGODB)

Step 12

Verify records in MongoDB

- Now verifying the records are updated in mongodb

MongoChef Professional - 3T Software Labs - Non-Commercial License

File Edit Database Collection Index Document GridFS View Help

Connect IntelliShell Aggregate Map-Reduce Export Import Users Roles Feedback

MongoChef is now [Studio 3T](#), and version 5.6.4 is out! Download the update [here](#) to experience the latest features!

New Connection - imported on 08-Mar-2017

Account t1customers

localhost:27017 ubercardb t1customers

Query { } Sort { }

Projection { } Skip { }

Documents 1 to 1 | 50 | 1 2 3 4 5 6 7 8 9 10 11 12 13

```
1 {
2   "_id" : ObjectId("5a2d13ccdc037125bc06b877"),
3   "t1f1customerid" : "a1",
4   "t1f1customerfullname" : "Syed Rayyan",
5   "t1f1customergender" : "male",
6   "t1f1customeremail" : "rayyan@gmail.com",
7   "t1f1customerphone" : "9999191912",
8   "t1f1customeraddress" : "227 hrbr",
9   "t1f1customercity" : "bangalore",
10  "t1f1customerstate" : "karnataka",
11  "t1f1customercountry" : "india"
12 }
13
```



MONGODB DEMO : COMPANY MANY TO MANY RELATIONSHIP

SLC LOOPBACK:DEMO-VIII



MONGODB MANY2MANY

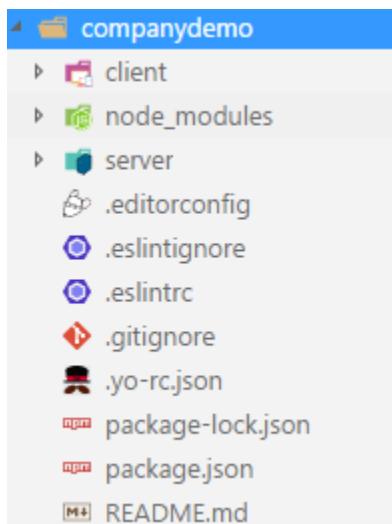
Please read terms of use for authorized access

Original Series

Step 1

Create API server project

- create API server project using `slc loopback` as shown in the figure using current version 3.x



```
PS E:\CT\loopback\codeplay> slc loopback companydemo
```



```
? What's the name of your application? companydemo
? Enter name of the directory to contain the project: companydemo
  create companydemo/
    info change the working directory to companydemo

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? api-server (A LoopBack API server with local User auth)
Generating .yo-rc.json
```

I'm all done. Running `npm install` for you to install the required dependencies. If this fails, try running the command yourself.

```
create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create README.md
create server\boot\authentication.js
create .gitignore
create client\README.md
```

Next steps:

Change directory to your app
`$ cd companydemo`

Create a model in your app
`$ slc loopback:model`

Run the app
`$ node .`

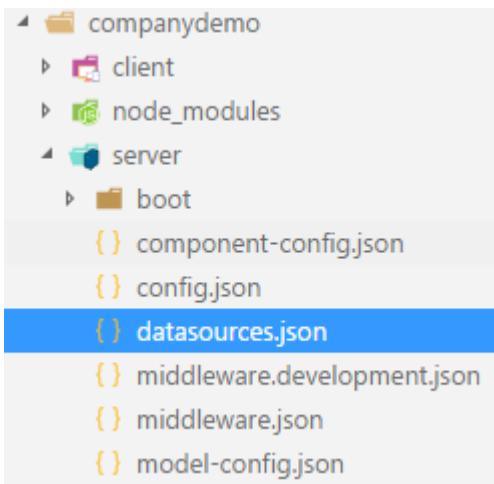


MONGODB MANY2MANY

Step 2

Create a datasource

- create a datasource using `scl loopback:datasource`



```
PS E:\CT\loopback\codeplay\companydemo> scl loopback:datasource
? Enter the data-source name: mongoconnect
? Select the connector for mongoconnect: MongoDB (supported by StrongLoop)
Connector-specific configuration:
? Connection String url to override other settings (eg: mongodb://username:password@hostname:port/database): mongodb://@localhost:27017/companydb
? host: localhost
? port: 27017
? user:
? password:
? database: companydb
? Install loopback-connector-mongodb@^1.4 Yes
```

```
{
  "db": {
    "name": "db",
    "connector": "memory"
  },
  "mongoconnect": {
    "host": "localhost",
    "port": 27017,
    "url": "mongodb://@localhost:27017/companydb",
    "database": "companydb",
    "password": "",
    "name": "mongoconnect",
    "user": "",
    "connector": "mongodb"
  }
}
```



MONGODB MANY2MANY

Please read terms of use for authorized access

Original Series

Step 3

Create a models

- create model for company using slc loopback:model as shown in the figure.
- It creates common/models and models inside it.

```
PS E:\CT\loopback\codeplay\companydemo> slc loopback:model
? Enter the model name: t1company
? Select the data-source to attach t1company to: mssqlconnectdb (mssql)
? Select model's base class PersistedModel
? Expose t1company via the REST API? Yes
? Custom plural form (used to build REST URL): t1companies
? Common model or server only? common
Let's add some t1company properties now.
```

Enter an empty property name when done.

```
? Property name: t1f1companyid
  invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```

Let's add another t1company property.

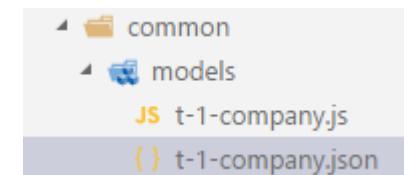
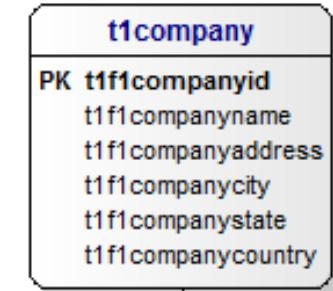
Enter an empty property name when done.

```
? Property name: t1f1companyname
  invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```

Let's add another t1company property.

Enter an empty property name when done.

```
? Property name: t1f1companyaddress
  invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```



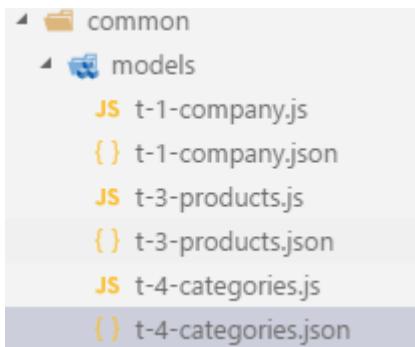


MONGODB MANY2MANY

Step 4

Create a models

- create products and categories model similarly as done in step 3.



Please read terms of use for authorized access

Original Series

```

PS E:\CT\loopback\codeplay\companydemo> slc loopback:model
? Enter the model name: t3products
? Select the data-source to attach t3products to: mssqlconnectdb (mssql)
? Select model's base class PersistedModel
? Expose t3products via the REST API? Yes
? Custom plural form (used to build REST URL): t3products
? Common model or server only? common
Let's add some t3products properties now.

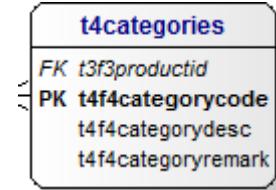
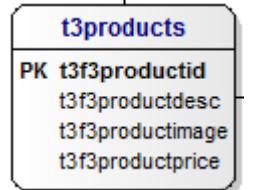
Enter an empty property name when done.
? Property name: t3f3productid
  invoke  loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:  
  

Let's add another t3products property.
Enter an empty property name when done.
? Property name: t3f3productname
  invoke  loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:  
  

Let's add another t3products property.
Enter an empty property name when done.
? Property name: t3f3productdesc
  invoke  loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:  
  

Let's add another t3products property.
Enter an empty property name when done.
? Property name: t3f3productprice
  invoke  loopback:property
? Property type: number
? Required? Yes
? Default value[leave blank for none]:

```



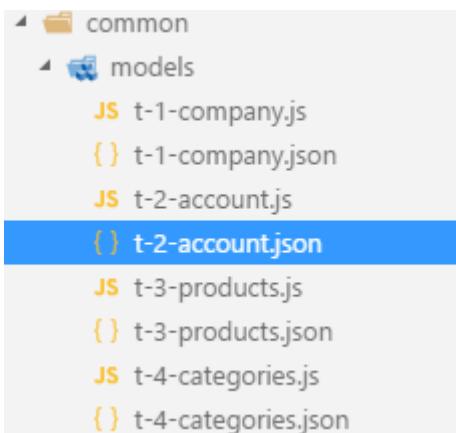


MONGODB MANY2MANY

Step 5

Create a ACCOUNT MODEL

- create account model by deriving from base model “user”.



```
PS E:\CT\loopback\codeplay\companydemo> slc loopback:model
? Enter the model name: t2account
? Select the data-source to attach t2account to: mssqlconnectdb (mssql)
? Select model's base class User ←
? Expose t2account via the REST API? Yes
? Custom plural form (used to build REST URL): t2accounts
? Common model or server only? common
Let's add some t2account properties now.
```

```
{
  "name": "t2account",
  "plural": "t2accounts",
  "base": "User",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {},
  "validations": [],
  "relations": {},
  "acls": [],
  "methods": {}
}
```



Step 5

Establish relationships

- establishing one to one relationship between company and account models.

```
PS E:\CT\loopback\codeplay\companydemo> slc loopback:relation
? Select the model to create the relationship from: t1company
? Relation type:
  has many
  belongs to
  has and belongs to many
> has one
```

```
PS E:\CT\loopback\codeplay\companydemo> slc loopback:relation
? Select the model to create the relationship from: t1company
? Relation type: has one
? Choose a model to create a relationship with: t2account
? Enter the property name for the relation: account
? Optionally enter a custom foreign key: _t1f1companyid
```

```
"relations": {
  "account": {
    "type": "hasOne",
    "model": "t2account",
    "foreignKey": "_t1f1companyid"
  }
},
```



Step 7

Automigrate function

- We would like automigrate all the database tables

```
'use strict';
module.exports = function(app){
  app.dataSources.mongodb.automigrate(['t1company',
  't2account','t3products','t4categories'],err=>{
    if(err) throw err;
    console.log('All the Models are synced with companydb');
  })
}
```

```
'use strict';
module.exports = function(app){
  app.dataSources.mongodb.automigrate(['t1company',
  't2account','t3products','t4categories'],err=>{
    if(err) throw err;
    console.log('All the Models are synced with
  companydb');
  })
}
```



MONGODB MANY2MANY

Step 8

Establish relationships

- establishing one to many relationship between categories and products.
- Each category **has many** products

```
PS E:\CT\loopback\codeplay\companydemo> slc loopback:relation
? Select the model to create the relationship from: t4categories
? Relation type: has many
? Choose a model to create a relationship with: t3products
? Enter the property name for the relation: t3products
? Optionally enter a custom foreign key: t3f3categoryid
? Require a through model? No
```

{ } t-4-categories.json

```
"relations": {
  "t3products": {
    "type": "hasMany",
    "model": "t3products",
    "foreignKey": "t3f3categoryid"
  }
},
```



Step 9

Establish relationships

- establishing many to many relationship between company and products.
- Many companies have many products.
- Achieved by creating a pivot table /junction table

SUB-STEP: A

```
PS E:\CT\loopback\codeplay\companydemo> slc loopback:relation
? Select the model to create the relationship from: t1company
? Relation type: has many
? Choose a model to create a relationship with: t3products
? Enter the property name for the relation: t3products
? Optionally enter a custom foreign key:
? Require a through model? Yes
? Choose a through model: (other)
? Enter the model name: CompanyProduct
```

PIVOT_TABLE

SUB-STEP: B

```
PS E:\CT\loopback\codeplay\companydemo> slc loopback:relation
? Select the model to create the relationship from: t3products
? Relation type: has many
? Choose a model to create a relationship with: t1company
? Enter the property name for the relation: t1companies
? Optionally enter a custom foreign key:
? Require a through model? Yes
? Choose a through model: (other)
? Enter the model name: CompanyProduct
```

PIVOT_TABLE



MONGODB MANY2MANY

Step 10

Create PIVOT TABLE MODEL

- create **pivot table** “**companyproduct**” with persisted model, but REST API is not enabled to render.
- CompanyProduct model **belongs to both** Product and Company model, as shown in sub-step: B.

SUB-STEP: A

```
PS E:\CT\loopback\codeplay\companydemo> slc loopback:model
? Enter the model name: CompanyProduct
? Select the data-source to attach CompanyProduct to: mongoconnect (mongodb)
? Select model's base class PersistedModel
? Expose CompanyProduct via the REST API? No ←
? Common model or server only? common
Let's add some CompanyProduct properties now.
```

Enter an empty property name when done.

? Property name:



SUB-STEP: B

```
PS E:\CT\loopback\codeplay\companydemo> slc loopback:relation
? Select the model to create the relationship from: CompanyProduct
? Relation type: belongs to
? Choose a model to create a relationship with: t3products
? Enter the property name for the relation: t3products
? Optionally enter a custom foreign key:
PS E:\CT\loopback\codeplay\companydemo> slc loopback:relation
? Select the model to create the relationship from: CompanyProduct
? Relation type: belongs to
? Choose a model to create a relationship with: t1company
? Enter the property name for the relation: t1company
? Optionally enter a custom foreign key:
```



MONGODB MANY2MANY

Step 11

Run your application

- run your application to render Explorer API as shown in the figure.
- Run => node .

```
PS E:\CT\loopback\codeplay\companydemo> node .
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
All the Models are synced with companydb
```

http://localhost:3000/explorer/#!/t2account/t2account_create

localhost:3000/explorer/#!/t2account/t2account_create

LoopBack API Explorer

Token Set: Nin5MHleZm4TKIY1MCJa | Set Access Token

companydemo

t1company		Show/Hide List Operations Expand Operations

t2account		Show/Hide List Operations Expand Operations
PATCH	/t2accounts	Patch an existing model instance or insert a new one into the data source.
GET	/t2accounts	Find all instances of the model matched by filter from the data source.
PUT	/t2accounts	Replace an existing model instance or insert a new one into the data source.
POST	/t2accounts	Create a new instance of the model and persist it into the data source.

Response Class (Status 200)
Request was successful

Model Example Value

```
{
  "realm": "string",
  "username": "string",
  "email": "string",
}
```



MONGODB MANY2MANY

Step 12

Verify data in MongoDB

- verify the data inside mongodb using mongochef.

The screenshot shows the MongoChef Professional interface. On the left, a sidebar lists various MongoDB databases and collections. In the main pane, a specific document from the 't1company' collection in the 'companydb' database is displayed in JSON format. The document contains fields such as '_id', 't1f1companyid', 't1f1companynamne', 't1f1companyaddress', 't1f1companycity', 't1f1companystate', and 't1f1companycountry'. The JSON code is as follows:

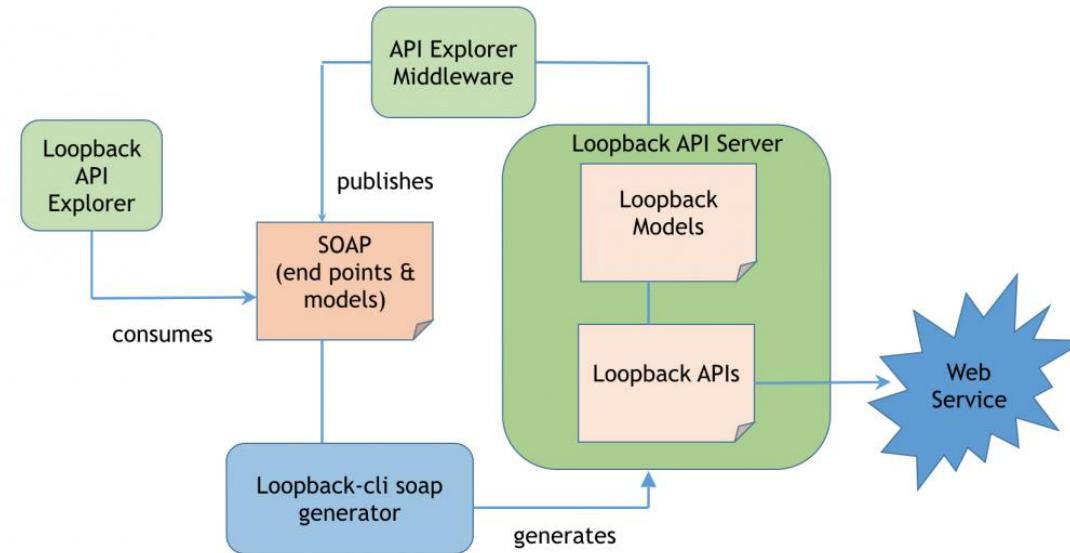
```
1
2   "_id" : ObjectId("5a2e9016eae13e0b8468d228"),
3   "t1f1companyid" : "aa281821",
4   "t1f1companynamne" : "sycliq",
5   "t1f1companyaddress" : "cessna business park",
6   "t1f1companycity" : "bangalore",
7   "t1f1companystate" : "karnataka",
8   "t1f1companycountry" : "india"
9 }
10
```

Please read terms of use for authorized access

Original Series

Starting MongoDB:

```
mongod --dbpath E:\mongodata\data --port 27017 --logpath E:\mongodata\mongod.log
```



SIMPLE OBJECT ACCESS PROTOCOL (SOAP) CONNECTOR DEMO

SLC LOOPBACK:DEMO-IX



Playbook for creating soap connector

Please read terms of use for authorized access

Original Series

- 1 • Bootstrap the application
• Slc loopback weathersoap
- 2 • Create a soap data source
• Slc loopback:datasource
- 3 • Create a weatherReport model
• Slc loopback:model
- 4 • Create loopback soap service connection and select appropriate service and bindings.
- 5 • Run your application
• Node .



SOAP CONNECTOR DEMO

Please read terms of use for authorized access

Original Series

Step 1

Create API server project

- create API server project using slc loopback as shown in the figure using current version 3.x

```
PS E:\CT\loopback\codeplay> slc loopback weathersoap
```



```
? What's the name of your application? weathersoap
? Enter name of the directory to contain the project: weathersoap
  create weathersoap/
    info change the working directory to weathersoap

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? api-server (A LoopBack API server with local User auth)
Generating .yo-rc.json
```

I'm all done. Running `npm install` for you to install the required dependencies. If this fails, try running the command yourself.

```
create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create README.md
create server\boot\authentication.js
create .gitignore
  .gitattributes
```

Next steps:

Change directory to your app
`$ cd weathersoap`

Create a model in your app
`$ slc loopback:model`

Run the app
`$ node .`

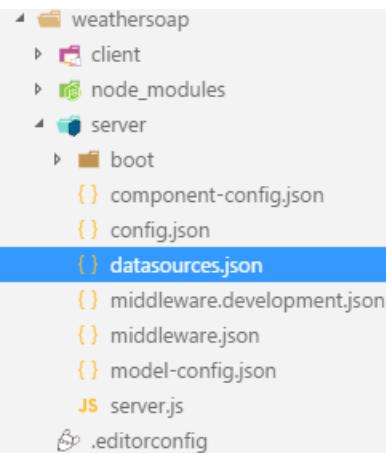


SOAP CONNECTOR DEMO

Step 2

Create a SOAP datasource

- create a soap datasource connector.



C ⓘ www.webservicex.com/globalweather.asmx

Core Java Topics & Ba Apps Learn ASP.NET

[GetCitiesByCountry](#)
Get all major cities by country name(full / part).

[GetWeather](#)
Get weather report for all major cities around the world.

- <http://www.webservicex.com/globalweather.asmx?WSDL>

```

PS E:\CT\loopback\codeplay\weathersoap> slc loopback:datasource
? Enter the data-source name: soapconnect
? Select the connector for soapconnect: SOAP webservices (supported by StrongLoop)
Connector-specific configuration:
? URL to the SOAP web service endpoint: http://www.webservicex.com/globalweather.asmx
? HTTP URL or local file system path to the WSDL file: http://www.webservicex.com/globalweather.asmx?WSDL
? Expose operations as REST APIs: Yes
? Maps WSDL binding operations to Node.js methods:
? Install loopback-connector-soap@^3.0 Yes
[ ..... ] | diffTrees: sill install generateActionsToTake
  
```

```

"operations": {
  "GetCitiesByCountry": {
    "service": "GlobalWeather",
    "port": "GlobalWeatherSoap",
    "operation": "GetCitiesByCountry"
  },
  "GetWeather": {
    "service": "GlobalWeather",
    "port": "GlobalWeatherSoap",
    "operation": "GetWeather"
  }
}
  
```

Other examples of WSDL files:

- <http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl>
- <http://webservices.amazon.com/AWSECommerceService/2013-08-01/AWSECommerceService.wsdl>
- <http://webservices.gama-system.com/stockquotes.asmx?wsdl>
- <http://www.webservicex.net/periodictable.asmx?WSDL>
- <http://www.webservicex.com/globalweather.asmx?WSDL>



Step 2

- Created data source with operations

```
{  
    "db": {  
        "name": "db",  
        "connector": "memory"  
    },  
    "soapconnect": {  
        "url": "http://www.webservicex.com/globalweather.asmx",  
        "name": "soapconnect",  
        "wsdl": "http://www.webservicex.com/globalweather.asmx?WSDL",  
        "remotingEnabled": true,  
        "operations":{  
            "GetCitiesByCountry": {  
                "service": "GlobalWeather",  
                "port": "GlobalWeatherSoap",  
                "operation": "GetCitiesByCountry"  
            },  
            "GetWeather": {  
                "service": "GlobalWeather",  
                "port": "GlobalWeatherSoap",  
                "operation": "GetWeather"  
            }  
        },  
        "connector": "soap"  
    }  
}
```



SOAP CONNECTOR DEMO

Step 3

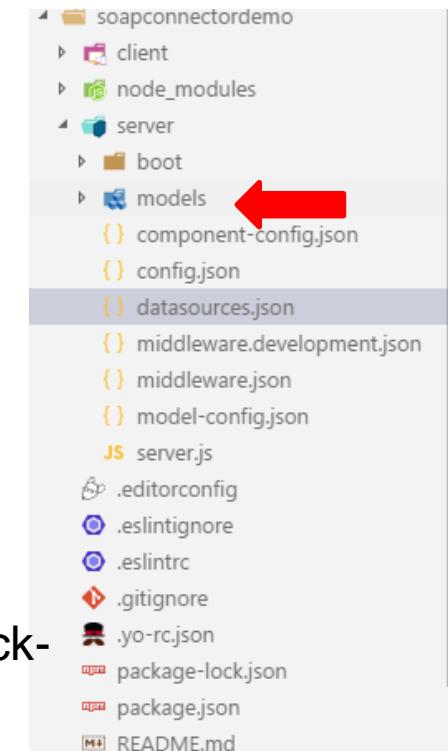
Create a cart model

- create cart model using `scl loopback:model`
- The model is render in the **server only** and it **extends model base class**
- **Notice that server/models directory is created with cartmodel**

```
PS E:\CT\loopback\codeplay\soapconnectordemo> scl loopback:model
? Enter the model name: cartmodel
? Select the data-source to attach cartmodel to: soapconnector (soap)
? Select model's base class Model ←
? Expose cartmodel via the REST API? Yes ←
? Custom plural form (used to build REST URL): no ←
? Common model or server only? server ←
Let's add some cartmodel properties now.
```

Enter an empty property name when done.

? Property name: █



<https://github.com/strongloop-community/loopback-example-connector/tree/soap#scaffold-the-app>



SOAP CONNECTOR DEMO

Step 3

Create models

- Create a model for weathermodel using slc loopback:model

```
PS E:\CT\loopback\codeplay\weathersoap> slc loopback:model
? Enter the model name: weathermodel
? Select the data-source to attach weathermodel to: soapconnect (soap)
? Select model's base class Model
? Expose weathermodel via the REST API? Yes
? Custom plural form (used to build REST URL): NO
? Common model or server only? server
Let's add some weathermodel properties now.

Enter an empty property name when done.
? Property name: 
```



SOAP CONNECTOR DEMO

Step 4

Generate APIs from SOAP web services datasource

- lb soap => select the soapconnect datasource
- Identify the service
- Identify the binding.
- Select operations to be generated.

```
PS E:\CT\loopback\codeplay\weathersoap> lb soap
? Select the datasource for SOAP discovery soapconnect
WSDL for datasource soapconnect: http://www.webservicex.com/globalweather.asmx?WSDL
? Select the service: GlobalWeather
? Select the binding: GlobalWeatherSoap
? Select operations to be generated: (Press <space> to select, <a> to toggle all, <i> to inverse selection)GetWeather, GetCitiesByCountry
Creating model definition for soap_GlobalWeatherSoap...
Creating model definition for GetWeather...
Creating model definition for GetWeatherResponse...
Creating model definition for GetCitiesByCountry...
Creating model definition for GetCitiesByCountryResponse...
Model definition created/updated for soap_GlobalWeatherSoap.
Model definition created/updated for GetWeatherResponse.
Model definition created/updated for GetCitiesByCountryResponse.
Model definition created/updated for GetCitiesByCountry.
Model definition created/updated for GetWeather.
Creating model config for soap_GlobalWeatherSoap...
Model config created for soap_GlobalWeatherSoap.
Creating model config for GetWeather...
Model config created for GetWeather.
Creating model config for GetWeatherResponse...
Model config created for GetWeatherResponse.
Creating model config for GetCitiesByCountry...
Model config created for GetCitiesByCountry.
Creating model config for GetCitiesByCountryResponse...
Model config created for GetCitiesByCountryResponse.
Generating E:\CT\loopback\codeplay\weathersoap\server\models\soap-global-weather-soap.js
Models are successfully generated from WSDL.
```

A



SOAP CONNECTOR DEMO

Step 5

Run your application

- Let us run our API explorer => node .

Original Series

Please read terms of use for authorized access

```
CURL
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \
  "CountryName": "india" \
}' "http://localhost:3000/api/GlobalWeatherSoap/GetCitiesByCountry?access_token=Nin5MHleZm4TK1Y1MCJaUDFbBbg5zO1RWhfx2BEu4qQLPDxHfgCxb"

Request URL
http://localhost:3000/api/GlobalWeatherSoap/GetCitiesByCountry?access_token=Nin5MHleZm4TK1Y1MCJaUDFbBbg5zO1RWhfx2BEu4qQLPDxHfgCxb

Response Body
</Table><r/n <Table><r/n <Country>India</Country><r/n <City>Thiruvananthapuram</City><r/n </Table><r/n <Table><r/n

Response Code
200
```

```
PS E:\CT\loopback\codeplay\weathersoap> node .
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

localhost:3000/explorer#!/soap95GlobalWeatherSoap/soap_GlobalWeatherSoap_GetCitiesByCountry

LoopBack API Explorer

Token Set: Nin5MHleZm4TK1Y1MCJaUDFbBbg5zO1RWhfx2BEu4qQLPDxHfgCxb | Set Access Token

soap_GlobalWeatherSoap

POST /GlobalWeatherSoap/GetCitiesByCountry

Response Class (Status 200)
Request was successful

Model Example Value

```
{
  "GetCitiesByCountryResult": "string"
}
```

Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
GetCitiesByCountry	{ "CountryName": "india" }	GetCitiesByCountry	body	Model Example Value
				{ "CountryName": "string" }



SOAP CONNECTOR DEMO WITH REMOTE METHODS

SLC LOOPBACK:DEMO-X



Playbook for creating soap connect with remote methods

Please read terms of use for authorized access

Original Series

- 1 • Bootstrap the application
• Slc loopback soapstockapp
- 2 • Create a soap data source
• Slc loopback:datasource
- 3 • Create a soapstock model
• Slc loopback:model
- 4 • Add static remote methods to server models
• Add custom logic to javascript (model)
- 5 • Run your application
• Node .

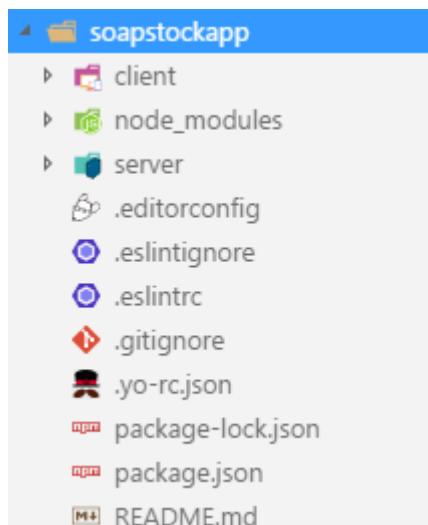


SOAPSTOCK +REMOTE METHODS

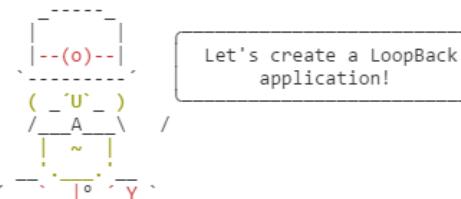
Step 1

Create API server project

- create API server project using slc loopback as shown in the figure using current version 3.x



```
PS E:\CT\loopback\codeplay> slc loopback soapstockapp
```



```
? What's the name of your application? soapstockapp
? Enter name of the directory to contain the project: soapstockapp
  create soapstockapp/
    info change the working directory to soapstockapp

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? api-server (A LoopBack API server with local User auth)
Generating .yo-rc.json
```

I'm all done. Running `npm install` for you to install the required dependencies. If this fails, try running the command yourself.

```
create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create README.md
create server\boot\authentication.js
create .gitignore
create client\README.md
```

Next steps:

Change directory to your app
`$ cd soapstockapp`

Create a model in your app
`$ slc loopback:model`

Run the app
`$ node .`

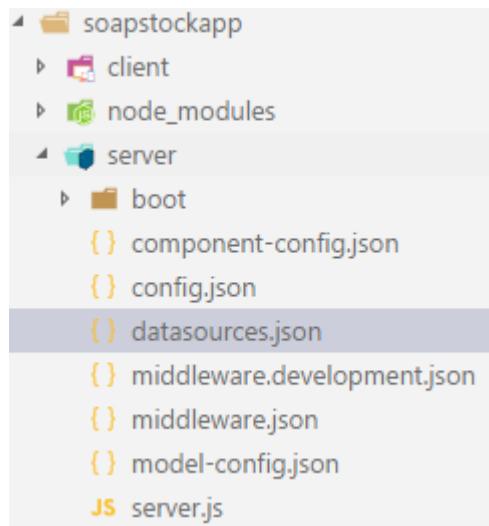


SOAPSTOCK +REMOTE METHODS

Step 2

Create a SOAP datasource

- create a soap datasource connector.



```
PS E:\CT\loopback\codeplay\soapstockapp> slc loopback:datasource
? Enter the data-source name: soapriconnect
? Select the connector for soapriconnect: SOAP webservices (supported by StrongLoop)
Connector-specific configuration:
? URL to the SOAP web service endpoint: http://www.webservicex.net/stockquote.asmx
? HTTP URL or local file system path to the WSDL file: http://www.webservicex.net/stockquote.asmx?WSDL
? Expose operations as REST APIs: Yes
? Maps WSDL binding operations to Node.js methods:
? Install loopback-connector-soap@^3.0 Yes
```

```
{
  "db": {
    "name": "db",
    "connector": "memory"
  },
  "soapriconnect": {
    "url": "http://www.webservicex.net/stockquote.asmx",
    "name": "soapriconnect",
    "wsdl": "http://www.webservicex.net/stockquote.asmx?WSDL",
    "remotingEnabled": true,
    // Map SOAP service/port/operation to Node.js methods
    "operations": {
      // The key is the method name
      "stockQuote": {
        "service": "StockQuote", // The WSDL service name
        "port": "StockQuoteSoap", // The WSDL port name
        "operation": "GetQuote" // The WSDL operation name
      },
      // The key is the method name
      "stockQuote12": {
        "service": "StockQuote", // The WSDL service name
        "port": "StockQuoteSoap12", // The WSDL port name
        "operation": "GetQuote" // The WSDL operation name
      }
    },
    "connector": "soap"
  }
}
```

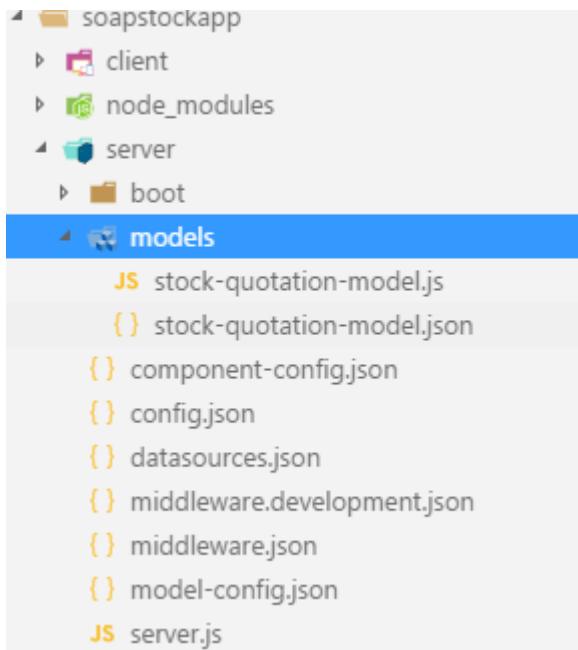


SOAPSTOCK +REMOTE METHODS

Step 3

Create a SOAPQuotationModel

- create a soap quotation model



```
PS E:\CT\loopback\codeplay\soapstockapp> slc loopback:model
? Enter the model name: stockQuotationModel
? Select the data-source to attach stockQuotationModel to: soaprmconnect (soap)
? Select model's base class Model
? Expose stockQuotationModel via the REST API? Yes
? Custom plural form (used to build REST URL): NO
? Common model or server only? server
Let's add some stockQuotationModel properties now.
```

Enter an empty property name when done.
? Property name:



Adding application logic

- When building an application, we generally need to **implement custom logic to process data and perform other operations before responding to client requests.**
 - Adding logic to models
 - Remote methods
 - Remote hooks
 - Operation hooks
 - Defining custom boot scripts, which run on application boot.
 - Defining custom middleware



Remote Methods

- It is a static method of a model, exposed over a **custom REST endpoint**.

1

- Create a function in the model extension file. The method name determines whether a remote method is static or an instance method.
- A method name starting with **prototype**. Indicates an instance method. **By default, a remote method is static.**

2

- Register using JSON in the model definition JSON file.
- Registering using Javascript code in the model extension file calling `remoteMethod()`.

<https://loopback.io/doc/en/lb3/Remote-methods.html#example>



SOAPSTOCK +REMOTE METHODS

Step 3

Add Custom Remote Methods

- Custom remote methods to StockQuotationModel

```
PS E:\CT\loopback\codeplay\soapstockapp> slc loopback:model
? Enter the model name: stockQuotationModel
? Select the data-source to attach stockQuotationModel to: soaprmconnect (soap)
? Select model's base class Model
? Expose stockQuotationModel via the REST API? Yes
? Custom plural form (used to build REST URL): NO
? Common model or server only? server
Let's add some stockQuotationModel properties now.

Enter an empty property name when done.
? Property name: -
```



JS stock-quotation-model.js

```
'use strict';
module.exports = function (Stockquotationmodel) {
  //Remote Methods
  Stockquotationmodel.getStockQuote = function (symbol, cb) {
    Stockquotationmodel.stockQuote({
      ElementName: symbol || 'FB'
    }, function (err, response) {
      var result = response;
      cb(err, result);
    });
  };

  Stockquotationmodel.getStockQuote12 = function (symbol, callback) {
    Stockquotationmodel.stockQuote12({
      ElementName: symbol || 'IBM'
    }, function (err, response) {
      var result = response;
      callback(err, result);
    });
  };
}

// Map to REST/HTTP
Stockquotationmodel.remoteMethod(
  'getStockQuote', {
    accepts: [
      {
        arg: 'symbol', type: 'string', required: true,
        http: { source: 'query' }
      }],
    returns: { arg: 'result', type: 'object', root: true },
    http: {
      verb: 'get',
      path: '/StockQuote'
    }
  );
// Map to REST/HTTP
Stockquotationmodel.remoteMethod(
  'getStockQuote12', {
    accepts: [
      {
        arg: 'symbol', type: 'string', required: true,
        http: { source: 'query' }
      }],
    returns: {
      arg: 'result', type: 'object', root: true
    },
    http: { verb: 'get', path: '/StockQuote12' }
  );
};
```



SOAPSTOCK +REMOTE METHODS

Step 4

Run your application

- Run your application to render API explorer using node .

The screenshot shows the LoopBack API Explorer interface. At the top, the URL is localhost:3000/explorer/#!/stockQuotationModel/stockQuotationModel_getStockQuote. The title bar says "LoopBack API Explorer". A search bar contains "accessToken" and a button says "Set Access Token". Below the title bar, there are tabs for "soapstockapp" and "stockQuotationModel". Under "stockQuotationModel", there is a "GET /NO/StockQuote" operation. The response class is "Response Class (Status 200)" and the message is "Request was successful". There are tabs for "Model" and "Example Value", both of which show an empty JSON object "{}". Below the response area, there is a "Response Content Type" dropdown set to "application/json" and a "Parameters" section.

Please read terms of use for authorized access

Original Series



Hooks

Please read terms of use for authorized access

Original Series

Remote Hooks

- Hooks that execute before or after calling a remote method, either a custom remote method or a standard create, retrieve, update and delete method inherited from `PersistedModel`.
- A remote hook enables you to execute a function before or after a remote method is called by a client.
 - `beforeRemote()`
 - `afterRemote()`
 - `afterRemoteError()`

Operation Hooks

- Hooks that execute when models perform create, retrieve, update and delete operations.
- They replace deprecated model hooks.
- They are triggered by all methods that execute a particular high-level create, read, update or delete operation.



Operation Hook

Please read terms of use for authorized access

Original Series

Method	find	exists	count	create	upsert	findOrCreate	deleteAll	updateAll	prototype	prototype	prototype
Operation hook	findOne	findById					deleteById		.save	.delete	.updateAttributes
access	X	X	X	X	X	X	X	X			
before save				X	X	X		X	X		X
after save				X	X	X		X	X		X
before delete							X			X	
after delete							X			X	
loaded	X	X	X	X	X	X		X	X		X
persist				X	X	X		X	X		X



Connector hook

- Connector hooks are triggered by actions of connectors
- Connectors are responsible for interacting with the backend systems on behalf of model methods.
- Connector hooks enable applications to intercept the connector execution.
 - Before execute
 - After execute

Please read terms of use for authorized access

Original Series



REMOTE METHODS ON MODELS AND VALIDATIONS

SLC LOOPBACK:DEMO-XI



Playbook for creating soap connect with remote methods

Please read terms of use for authorized access

Original Series

- 1 • Bootstrap the application
• Slc loopback remmetapp
- 2 • Create a soap data source
• Slc loopback:datasource
- 3 • Create a shoppingcart model
• Slc loopback:model
- 4 • Add static remote methods to server models
• Add custom logic to javascript (model)
- 5 • Run your application
• Node .

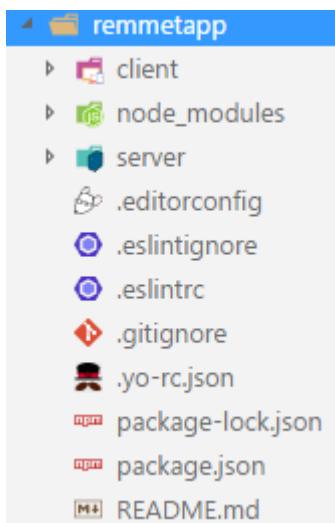


REMOTE METHODS ON MODELS

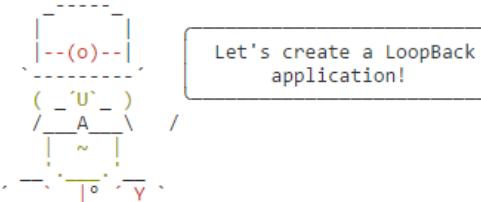
Step 1

Create API server project

- create API server project using `sfc loopback` as shown in the figure using current version 3.x



PS E:\CT\loopback\codeplay> `sfc loopback remmetapp`



```

? What's the name of your application? remmetapp
? Enter name of the directory to contain the project: remmetapp
p
  create remmetapp/
    info change the working directory to remmetapp

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? api-server (A
LoopBack API server with local User auth)
Generating .yo-rc.json

```

I'm all done. Running `npm install` for you to install the required dependencies. If this fails, try running the command yourself.

```

create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create README.md
create server\boot\authentication.js
create .gitignore
create client\README.md

```

Next steps:

Change directory to your app
\$ `cd remmetapp`

Create a model in your app
\$ `sfc loopback:model`

Run the app
\$ `node .`



REMOTE METHODS ON MODELS

Step 2

Create datasource for mongodb

- create mongodb connector datasource using `slc loopback:datasource`

{ } **datasources.json**

```
PS E:\CT\loopback\codeplay> cd .\remmetapp\
PS E:\CT\loopback\codeplay\remmetapp> slc loopback:datasource
? Enter the data-source name: mongoconnect
? Select the connector for mongoconnect: MongoDB (supported by StrongLoop)
Connector-specific configuration:
? Connection String url to override other settings (eg: mongodb://username:password@hostname:port/database):
? host: localhost
? port: 27017
? user:
? password:
? database: loopcommercedb
? Install loopback-connector-mongodb@^1.4 Yes
[.....] / fetchMetadata: sill resolveWithNewModule loopback-connect

{
  "db": {
    "name": "db",
    "connector": "memory"
  },
  "mongoconnect": {
    "host": "localhost",
    "port": 27017,
    "url": "mongodb://@localhost:27017/loopcommercedb",
    "database": "loopcommercedb",
    "password": "",
    "name": "mongoconnect",
    "user": "",
    "connector": "mongodb"
  }
}
```

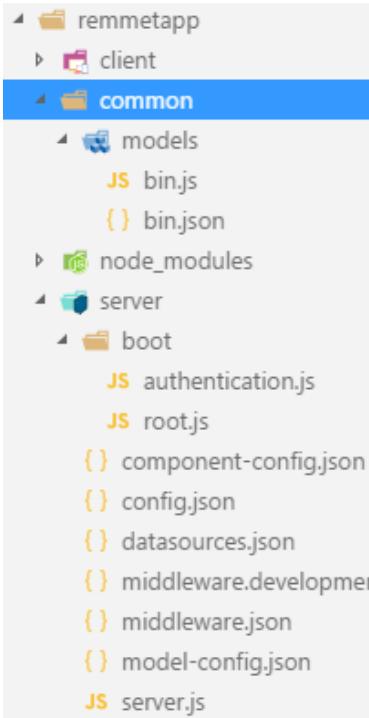


REMOTE METHODS ON MODELS

Step 3

Create a model returnbins

- create a model **bin** using `sfc loopback:model`



```
PS E:\CT\loopback\codeplay\remmetapp> sfc loopback:model
? Enter the model name: bin
? Select the data-source to attach bin to: mongoconnect (mongodb)
? Select model's base class PersistedModel
? Expose bin via the REST API? Yes
? Custom plural form (used to build REST URL): bins
? Common model or server only? common
Let's add some bin properties now.
```

```
Enter an empty property name when done.
? Property name: binid
    invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```

```
Let's add another bin property.
Enter an empty property name when done.
? Property name: binlocation
    invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```

```
Let's add another bin property.
Enter an empty property name when done.
? Property name: -
```



REMOTE METHODS ON MODELS

Step 4

Create remote-methods on model

- create a model **bin** using **scl loopback:remote-method**

```
PS E:\CT\loopback\codeplay\remmetapp> scl loopback:remote-method
? Select the model: bin
? Enter the remote method name: refurbish
? Is Static? No
? Description for method: refurbish items returned by customers
```

Let's configure where to expose your new method in the public REST API.
You can provide multiple HTTP endpoints, enter an empty path when you are done.

```
? Enter the path of this endpoint: /refurbish
? HTTP verb: post
```

Let's add another endpoint, enter an empty name when done.
? Enter the path of this endpoint:

Describe the input ("accepts") arguments of your remote method.
You can define multiple input arguments.
Enter an empty name when you've defined all input arguments.

```
? What is the name of this argument? count
? Select argument's type: number
? Is this argument required? Yes
? Please describe the argument: no of items to refurbish
? Where to get the value from? (auto)
```

Let's add another accept argument, enter an empty name when done.
? What is the name of this argument?

Describe the output ("returns") arguments to the remote method's callback function.
You can define multiple output arguments.
Enter an empty name when you've defined all output arguments.

```
? What is the name of this argument? result
? Select argument's type: object
? Is this argument a full response body (root)? Yes
? Please describe the argument: result of number of items refurbished
```

Let's add another return argument. Enter empty name when done.
? What is the name of this argument?

We added strong-remoting metadata for your new method to **common\models\bin.json**.
You must implement the method in **common\models\bin.js**. For example:
Here is sample code to get you started:

```
/**
 * refurbish items returned by customers
 * @param {number} count no of items to refurbish
 * @param {Function(Error, object)} callback
 */

bin.prototype.refurbish = function(count, callback) {
  var result;
  // TODO
  callback(null, result);
};
```

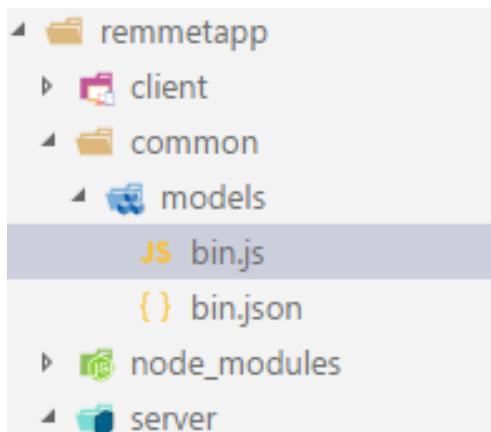


REMOTE METHODS ON MODELS

Step 5

Create remote-methods on model

- create a model bin using slc loopback:remote-method



```
'use strict';

module.exports = function(Bin) {
  Bin.prototype.refurbish = function (count, callback) {
    if(count<=0){
      var err={
        statusCode:"400",
        message:"We have very few items which are refurbished, add more numbers"
      };
      callback(err);
    }
    var result={
      status:`This Store has ${count} product items refurbished!!`
    };
    // TODO
    callback(null, result);
    //store it to the database permanently
  };
};
```



REMOTE METHODS ON MODELS

Step 6

Run your application

- run your application to render the custom remote method created.
- Run **node .**
- Now post/create a new refurbish bin

```
PS E:\CT\loopback\codeplay\remmetapp> node .
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

LoopBack API Explorer

Token Not Set accessToken [Set Access Token](#)

Find all instances of the model matched by filter from the data source.

POST /bins Create a new instance of the model and persist it into the data source.

Response Class (Status 200)
Request was successful

Model Example Value

```
{
  "binid": "string",
  "binlocation": "string",
  "id": "string"
}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
data	{ "binid": "u12313123123", "binlocation": "mannheim" }	Model instance data	body	Model Example Value

Decompile content type:

```
{
  "binid": "string",
  "binlocation": "string",
  "id": "string"
}
```



REMOTE METHODS ON MODELS

Step 7

Using remote-method

- select the refurbish bin id and add the number of items refurbished as 212.

Please read terms of use for authorized access

Original Series

Try it out | Hide Response

Curl

```
curl -X POST --header 'Content-Type: application/x-www-form-urlencoded' --header 'Accept: application/json' -d 'count=212' 'http://localhost:3000/api/bins/5a2fb5a8e6dc010cca90592/refurbish'
```

Request URL

<http://localhost:3000/api/bins/5a2fb5a8e6dc010cca90592/refurbish>

Response Body

```
{
  "status": "This Store has 212 product items refurbished!!"
}
```

Response Code

200

```
PS E:\CT\loopback\codeplay\remmetapp> node .
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

localhost:3000/explorer#!/bin/bin_prototype_refurbish

Java Topics & Books Apps Learn ASP.NET MVC ViewData in ASP.NET Stack Overflow Blog Sean Lahman | Data Retrosheet Event File Youtube Multi Downloader Working with

LoopBack API Explorer Token Not Set accessToken Set Access Token

Check whether a model instance exists in the data source.

GET /bins/{id}/exists

POST /bins/{id}/refurbish refurbish items returned by customers

Response Class (Status 200)
Request was successful

Model Example Value

```
{}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	5a2fb5a8e6dc010cca90592	bin id	path	string
count	212	no of items to refurbish	formData	double

Try it out! Hide Response

Curl

```
curl -X POST --header 'Content-Type: application/x-www-form-urlencoded' --header 'Accept: application/json' -d 'count=212' 'http://localhost:3000/api/bins/5a2fb5a8e6dc010cca90592/refurbish'
```



Model Validations

- A model can validate data before passing it on to a datastore such as a database to ensure that it conforms to the backend schema.
- Schema imposes restrictions on the model, to ensure that the model will save data that matches the corresponding database table.
- Loopback ensures that the data conforms to that schema definition

Validation method	Description
validatesAbsenceOf	Checks absence of one or more specified properties
validatesExclusionOf	Validate exclusion
validatesFormatOf	Validate format
validatesInclusionOf	Validate inclusion in set
validatesLengthOf	Validate length (min, max, is)
validatesNumericalityOf	Validate numericality (int or num)
validatesPresenceOf	Validate presence of one or more specified properties
validatesUniquenessOf	Validate uniqueness.

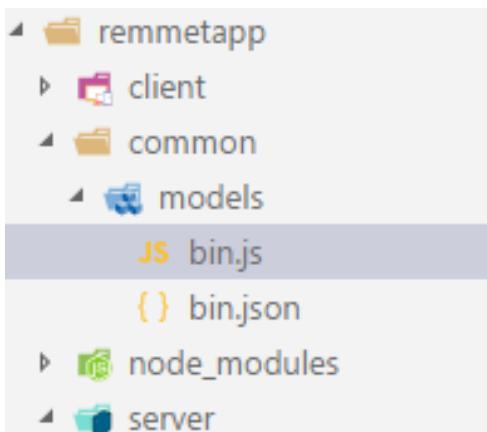


REMOTE METHODS ON MODELS

Step 8

Create remote-methods on model

- adding model validations



```
module.exports = function(Bin) {
  //model validations
  Bin.validatesLengthOf("binlocation",{
    min:4,
    message:{min:"location of store is defined short"}
  });

  //remote method
  Bin.prototype.refurbish = function (count, callback) {
    if(count<=0){
      var err={
        statusCode:"400",
        message:"We have very few items which are refurbished, add more numbers"
      };
      callback(err);
    }
    var result={
      status:`This Store has ${count} product items refurbished!!`
    };
    // TODO
    callback(null, result);
    //store it to the database permanently
  };
};
```

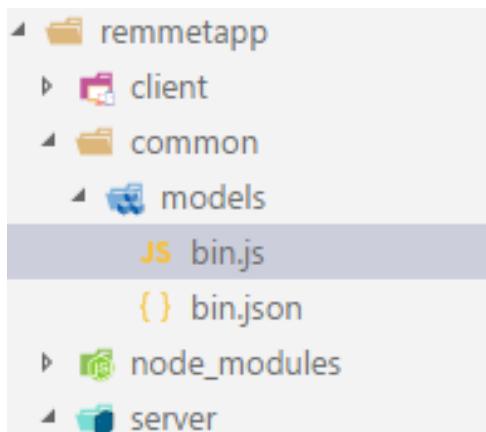


REMOTE METHODS ON MODELS

Step 8

Adding validations to model

- adding model validations



```
module.exports = function(Bin) {
  //model validations
  Bin.validatesLengthOf("binlocation",{
    min:4,
    message:{min:"location of store is defined short"}
  });

  //remote method
  Bin.prototype.refurbish = function (count, callback) {
    if(count<=0){
      var err={
        statusCode:"400",
        message:"We have very few items which are refurbished, add more numbers"
      };
      callback(err);
    }
    var result={
      status:`This Store has ${count} product items refurbished!!`
    };
    // TODO
    callback(null, result);
    //store it to the database permanently
  };
};
```



REMOTE METHODS ON MODELS

Step 9

Verifying validation

- upon adding a new binlocation with a single character, we receive a validation error as shown in the figure.

The screenshot shows the LoopBack API Explorer interface. In the 'Try it out!' section, a POST request is being made to `http://localhost:3000/api/bins` with the following JSON body:

```
{
  "binid": "s12314663434",
  "binlocation": "a"
}
```

The 'binlocation' field is highlighted in red, indicating a validation error. The response body shows the following error message:

```
{
  "statusCode": 422,
  "name": "ValidationError",
  "message": "The `bin` instance is not valid. Details: `binlocation` location of store is defined short (value: \"a\").",
  "details": {
    "context": "bin",
    "path": "binlocation"
  }
}
```

<https://loopback.io/doc/en/lb2/Validating-model-data.html>



COMPONENT-STORAGE AND UPLOADING SLC LOOPBACK:DEMO-XII

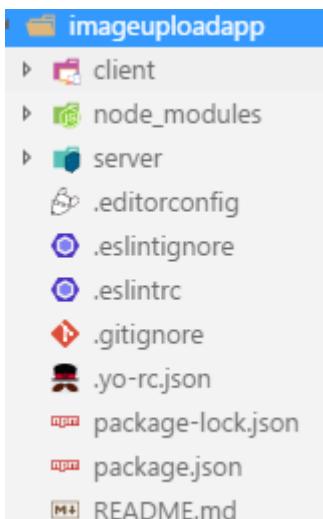


COMPONENT-STORAGE

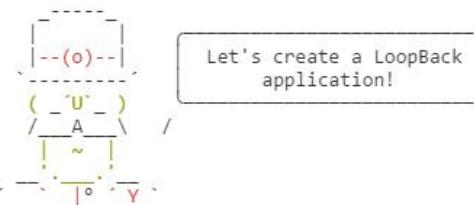
Step 1

Create API server project

- create API server project using slc loopback as shown in the figure using current version 3.x



PS E:\CT\loopback\codeplay> **slc** loopback imageuploadapp



```

? What's the name of your application? imageuploadapp
? Enter name of the directory to contain the project: imageuploadapp
  create imageuploadapp/
    info change the working directory to imageuploadapp

? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? api-server (A LoopBack API server with local User auth)
Generating .yo-rc.json
  
```

I'm all done. Running **npm install** for you to install the required dependencies. If this fails, try running the command yourself

```

create .editorconfig
create .eslintignore
create .eslintrc
create server\boot\root.js
create server\middleware.development.json
create server\middleware.json
create server\server.js
create README.md
create server\boot\authentication.js
create .gitignore
create client\README.md
  
```

Next steps:

Change directory to your app
\$ cd imageuploadapp

Create a model in your app
\$ slc loopback:model

Run the app
\$ node .



COMPONENT-STORAGE

Step 2

Create a new datasource

- install loopback-component-storage –save package
- Create a new data source

```
PS E:\CT\loopback\codeplay\imageuploadapp> npm install loopback-component-storage --save
[.....] / rollbackFailedOptional: verb npm-session 456a9d6a00060eab[.....] / rollback
npm WARN deprecated gcloud@0.10.0: gcloud has been renamed to google-cloud. To get new features and bug fixes,
you must use the new package.
npm WARN deprecated node-uuid@1.4.8: Use uuid module instead
+ loopback-component-storage@3.3.1
added 69 packages in 25.518s
```

Please read terms of use for authorized access

Original Series

```
PS E:\CT\loopback\codeplay\imageuploadapp> s1c loopback:datasource
? Enter the data-source name: container
? Select the connector for container: other
? Enter the connector's module name loopback-component-storage
? Install loopback-component-storage Yes
Please manually add config for your custom connector loopback-component-storage in server/datasources.json
npm WARN deprecated gcloud@0.10.0: gcloud has been renamed to google-cloud. To get new features and bug fixes, you must use the new package.
npm WARN deprecated node-uuid@1.4.8: Use uuid module instead
+ loopback-component-storage@3.3.1
updated 1 package in 7.005s
```



COMPONENT-STORAGE

Step 3

Create a new datasource

- update the datasource.json file

{ } datasources.json

```
{  
  "db": {  
    "name": "db",  
    "connector": "memory"  
  },  
  "container": {  
    "name": "container",  
    "connector": "loopback-component-storage",  
    "provider": "filesystem",  
    "root": "./uploads"  
  }  
}
```



COMPONENT-STORAGE

Please read terms of use for authorized access

Original Series

Step 4

Create a model

- Create a container model for the data source.

```
PS E:\CT\loopback\codeplay\imageuploadapp> slc loopback:model
? Enter the model name: Container
? Select the data-source to attach Container to: container (loopback-component-storage)
? Select model's base class PersistedModel
? Expose Container via the REST API? Yes
? Custom plural form (used to build REST URL):
? Common model or server only? common
Let's add some Container properties now.

Enter an empty property name when done.
? Property name:
```

<https://strongloop.com/strongblog/working-with-file-storage-and-loopback/>



COMPONENT-STORAGE

Step 5

Run your application

- Run your application using node .

localhost:3000/explorer/#!/Container/Container_upload

LoopBack API Explorer

Token Not Set accessToken

imageuploadapp

Container

Show/Hide | List Operations | Expand Operations

PATCH	/Containers	Patch an existing model instance or insert a new one into the data source.
GET	/Containers	Find all instances of the model matched by filter from the data source.
PUT	/Containers	Replace an existing model instance or insert a new one into the data source.
POST	/Containers	Create a new instance of the model and persist it into the data source.

Response Class (Status 200)
Request was successful

Please read terms of use for authorized access

Original Series



ANGULAR SDK IBM OPEN API



ANGULAR SDK

SLC LOOPBACK:DEMO-



Installing SDK and Running Angular

- Installing the LoopBack Angular SDK
 - Npm install –g loopback-sdk-angular-cli
- Generate the services
 - lb-ng ./server/server.js client/www/js/services/lbservices.js
 - Dependency
 - Bower install –save angular-resource (for \$resource/ngResource builtin service)



Resources



IBM API Connect

<https://developer.ibm.com/apiconnect/>

The screenshot shows the homepage of the IBM API Connect developer portal. At the top, there's a navigation bar with links for 'Secure' (HTTPS), 'Marketplace', 'Awase Khirni Syed', and a search icon. Below the navigation is a main banner with the text 'Create. Run. Manage. Secure.' and 'Take control of your API lifecycle with IBM API Connect'. In the center of the banner is a network graph icon. Below the banner, there's a call-to-action button with the text 'Try API Connect Free on Bluemix' and a smaller link 'Manage APIs on the IBM cloud'. The background has a blue and teal gradient design.

Please read terms of use for authorized access

Original Series



<https://swagger.io/>

Swagger: Pre-requisite for LoopBack

SYED AWASE KHIRNI



Swagger

Please read terms of use for authorized access

<https://swagger.io/>

The screenshot shows the official Swagger website at https://swagger.io/. The header features the "SWAGGER" logo with a green icon and the word "SWAGGER" in bold capital letters. Below the logo is the tagline "THE WORLD'S MOST POPULAR API TOOLING". A descriptive paragraph explains that Swagger is the world's largest framework for API developer tools, supporting the OpenAPI Specification (OAS) and enabling development across the entire API lifecycle. Two prominent buttons are visible: "OPEN SOURCE TOOLS" and "TRY SWAGGERHUB". At the bottom, there is a screenshot of the Swagger IDE interface, showing a "DESIGN" tab, a "BUILD" tab, and a "DOC" tab, with a preview window displaying a "PUT" request with parameters and responses.

Why Swagger?

- Integration with REST APIs is troublesome and inconsistent
- Different vendors have different REST semantics.
- Client libraries vary wildly vendor, language.
- Poor documentation standards
- Input parameters, allowable values, models, responses are explored through trial and error.
- Ad-hoc attributes



Swagger

How it works?

- Each API declares itself
 - Operations that can be performed
 - Parameters
 - Type(path,query, body)
 - Allowable values/data types
 - Input/Output models
 - Error responses with descriptions

What is Swagger?

- Swagger Spec is language agnostic
- Any swagger-compliant server can
 - Use the swagger client lib generator
 - Use the test framework
 - Use the sandbox UI



Swagger

- It is a technology framework for **producing, consuming , visualizing** RESTful APIs
- It is a methodology for **describing, documenting** RESTful APIs
- Swagger **does not support multiple API versions.**
- **It does not tell how to write the API**
- It's a JSON to specify metadata, to specify API structure and JSON schema for the model specification.
- It is machine readable and language agnostic.
- Apiary.io, masheryIO-docs are alternatives to swagger.



Swagger Toolset

<https://swagger.io/tools/>

Please read terms of use for authorized access

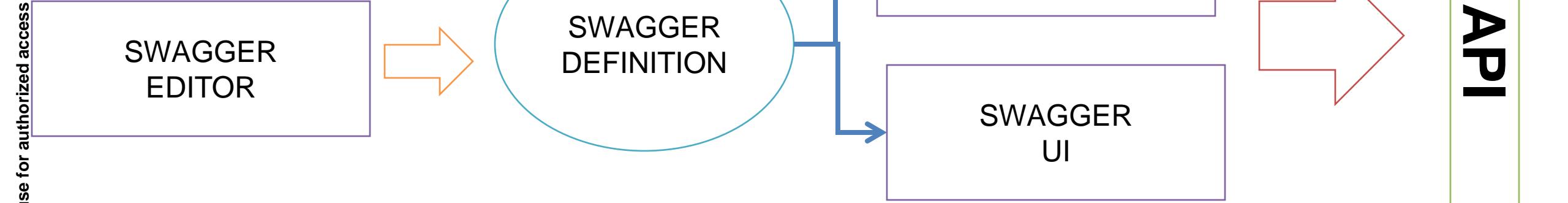
Original Series

Tool	Description
Swagger Core	Java-related libraries for creating, consuming, and working with Swagger definitions
Swagger Codegen	A code generation framework for building Client SDKs, servers, and documentation from Swagger Definitions
Swagger UI	An HTML5 based UI for exploring and interacting with a Swagger defined API
Swagger Editor	Browser based editor for authoring Swagger definitions using YAML

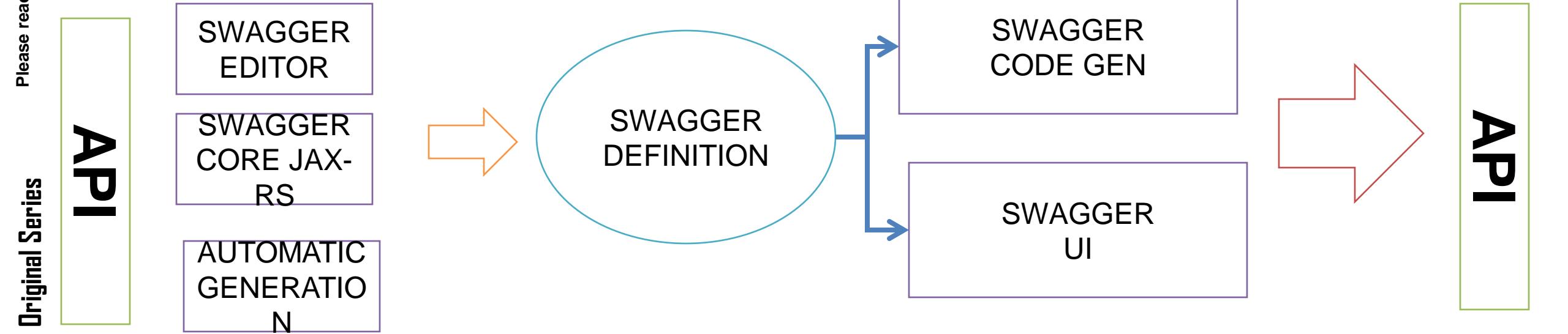
Tool	Description
Swagger JS	Javascript library for connecting to Swagger-defined APIs from browser and node.js applications
Swagger Node	Design-driven server implementation for node.js
Swagger Parser	Standalone library for parsing Swagger definitions from Java
Validator Badge	Standalone web service which validates swagger definitions dynamically



TOP DOWN APPROACH



BOTTOM UP APPROACH





SWAGGER OBJECTS

<https://swagger.io/docs/specification/basic-structure/>

Info

API metadata

definitions

Data types produced and consumed by operations

responses

Responses that can be used across operations

SecurityDefinitions

Security scheme definitions

ExternalDocs

Additional external documentation

paths

Available paths and operations

parameters

Parameters that can be used across operations

security

Alternative security schemes that can be used

tags

List of tags with additional metadata



SYED AWASE KHIRNI

SWAGGER: INSTALLATION



Swagger Installation

```
PS E:\CT\loopback> cd .\swagger-codeplay\  
PS E:\CT\loopback\swagger-codeplay> npm install -g swagger
```

Swagger Editor Installation

```
http://editor.swagger.io/?_ga=2.183538982.1851927523.1513093142-  
1425743669.1513093142#/  
npm install -g http-server  
wget https://github.com/swagger-api/swagger-editor/releases/download/v2.10.4/swagger-editor.zip  
unzip swagger-editor.zip  
http-server swagger-editor
```

```
PS E:\CT\loopback\swagger-codeplay> npm install -g swagger-cli  
C:\Users\SyedAwase\AppData\Roaming\npm\swagger-cli -> C:\Users\SyedAwase\AppData\Roaming\npm\node_modules\swagger-c  
+ swagger-cli@1.0.1  
added 77 packages in 14.187s
```



Swagger usage

```
PS E:\CT\loopback\swagger-codeplay> swagger --help
```

Usage: swagger [options] [command]

Options:

- V, --version output the version number
- h, --help output usage information

Commands:

project <action>	project actions
docs	open Swagger documentation
validate [options] [swaggerFile]	validate a Swagger document (supports unix piping)
convert [options] <swaggerFile> [apiDeclarations...]	Converts Swagger 1.2 documents to a Swagger 2.0 document
help [cmd]	display help for [cmd]

-



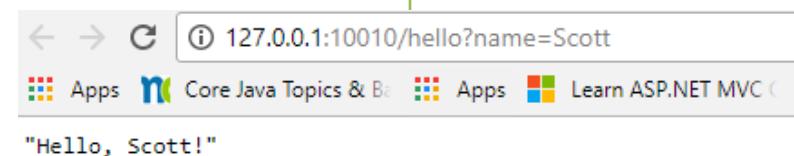
Creating a swagger project

Creating swagger project

- Swagger project create pomodotask
- Select “express”

Starting swagger project

- Swagger project start pomodotask





Running Swagger Project Editor

PS E:\CT\loopback\swagger-codeplay\pomodotask> **swagger** project edit

Starting Swagger Editor.

Opening browser to: <http://127.0.0.1:29124/#/edit>

Do not terminate this process or close this window until finished editing.

http://127.0.0.1:29124/#/!

```

swagger: "2.0"
info:
  version: "0.0.1"
  title: Hello World App
  # during dev, should point to your local machine
  host: localhost:10910
  # basePath prefixes all resource paths
  basePath: /
  #
schemes:
  # tip: remove http to make production-grade
  - http
  - https
  # format of bodies a client can send (Content-Type)
  consumes:
    - application/json
  # format of the responses to the client (Accepts)
  produces:
    - application/json
paths:
  /hello:
    get:
      description: Returns 'Hello' to the caller
      # used as the method name of the controller
      operationId: hello
      parameters:
        - name: name
          in: query
          description: The name of the person to whom to say hello
          required: false
          type: string
      responses:
        "200":
          description: Success
          schema:
            # a pointer to a definition
            $ref: "#/definitions/HelloworldResponse"
        # responses may fall through to errors
        default:
          description: Error
          schema:
            $ref: "#/definitions/ErrorResponse"
      /swagger:
        x-swagger-pipe: swagger_raw
      # complex objects have schema definitions
  
```

Hello World App
Version 0.0.1
Paths
/hello
GET /hello
Description
Returns 'Hello' to the caller
Parameters
Name Located in Description Required Schema
name query The name of the person to whom to say hello No string
Responses
Code Description Schema
200 Success `*HelloWorldResponse {
 message: string *
}`
default Error `*ErrorResponse {
 message: string *
}`
Try this operation



Creating our PomodoroTaskAPI

Please read terms of use for authorized access

```
swagger: "2.0"
info:
  version: "0.0.1"
  title: Pomodo Task API
# during dev, should point to your local machine
host: localhost:10010
# basePath prefixes all resource paths
basePath: /
#
schemes:
  # tip: remove http to make production-grade
  - http
  - https
# format of bodies a client can send (Content-Type)
consumes:
  - application/json
# format of the responses to the client (Accepts)
produces:
  - application/json
paths:
  /:
    get:
      description: "This endpoint returns all pomodoro tasks available in the database"
      operationId: "GetAllPomodoroTasks"
      parameters: []
      responses:
        200:
          description: "An array of Pomodoro Tasks"
          schema:
            type: "array"
            items:
              $ref: "#/definitions/PomodoroTask"
      x-swagger-router-controller:
        "GetAllPomodoroTasks"
```

Original Series

```
# complex objects have schema definitions
definitions:
  PomodoroTask:
    type: "object"
    properties:
      task_id:
        type: "integer"
        description: "taskid for pomodoro task"
      pomodoro_task:
        type: "string"
        description: "description of pomodorotask"
      pt_starttime:
        type: "integer"
        description: "time allocated to complete the task"
      pt_datecreated:
        type: "string"
        format: "date-time"
        description: "creation date of the task"
      pt_author:
        type: "string"
        description: "created by"
      pt_duedate:
        type: "string"
        format: "date-time"
        description: "task needs to be completed by due date"
      pt_taskstatus:
        type: "boolean"
        description: "status of the pomodoro task"
```



Adding a POST endpoint

```
post:
  description: "This endpoint allows us to create a new pomodoro task in the database"
  operationId: "Add PomodoroTask"
  produces:
    - "application/json"
  parameters:
    - in: "body"
      name: "pomodorotask"
      description: "the pomodoro task to be added"
      required: true
      schema:
        $ref: "#/definitions/PomodoroTask"
  responses:
    200:
      description: "Successful PomodoroTask added"
  x-swagger-router-controller:
    "AddPomodoroTask"
```



FetchbyId

```
/todo/{task_id}:
  get:
    description: "Retrieve a single pomodoro task by task_id"
    operationId: "FindById"
    produces:
      - "application/json"
    parameters:
      - name: "task_id"
        in : "path"
        description: "ID of the todo to fetch"
        required: true
        type: "integer"
        format: "int64"
    responses:
      200:
        description: "fetched a pomodoro task"
        schema:
          $ref: "#/definitions/PomodoroTask"
  x-swagger-router-controller:
    "FindById"
```



DeleteById

```
delete:  
  description: "delete a pomodoro task"  
  operationId: "DeleteTaskById"  
  parameters:  
    - name: "task_id"  
      in: "path"  
      description: "id of the task to be deleted"  
      required: true  
      type: "integer"  
      format: "int64"  
  responses:  
    204:  
      description: "todo deleted"  
  x-swagger-router-controller:  
    "DeleteTaskbyId"
```



Put

```
put:  
  description: "update a single to task by id"  
  operationId: "UpdateTaskById"  
  parameters:  
    - name: "id"  
      in: "path"  
      description: " id of the task to be updated"  
      required: true  
      type: "integer"  
      format: "int64"  
    - name: "updated_task"  
      in: "body"  
      description: "the updated todo"  
      required: true  
      schema:  
        $ref: "#/definitions/PomodoroTask"  
  responses:  
    200:  
      description: "Task Updated"  
    400:  
      description: "OOPS! there was an error!"
```



Swagger mock mode

Swagger project
start -m

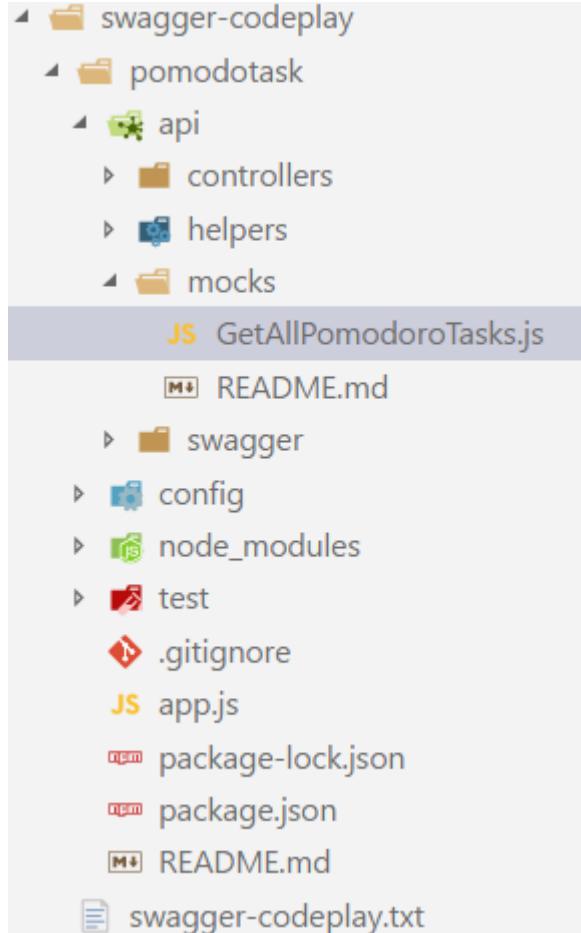
- To render api in mock up mode



Node.js function for getAll

Please read terms of use for authorized access

Original Series



```
'use strict';
module.exports={
    GetAllPomodoroTasks: GetAllPomodoroTasks,
}

function GetAllPomodoroTasks(req,res,next){
    res.json([
        {task_id:0,
            pomodoro_task:"Complete IBM OpenAPI v5.0",
            pt_starttime:"1400",
            pt_datecreated:"2016-01-01T20:12:12:00.000Z",
            pt_author:"Syed Awase",
            pt_duedate: "2016-01-08T20:12:12:00.000Z",
            pt_taskstatus:false
        }
    ]);
}
```



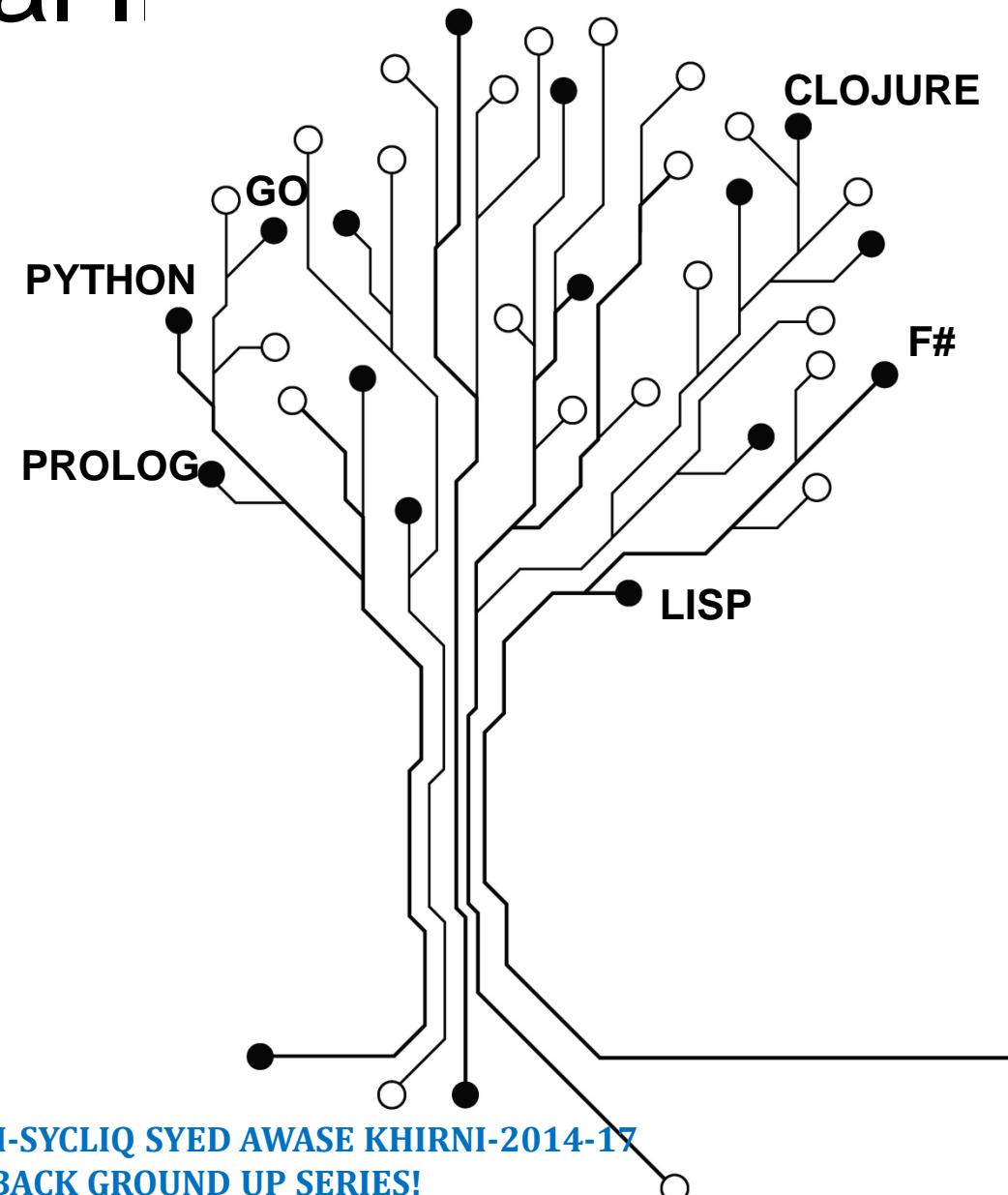
Empowering You

TPRI-SYCLIQ PROGRAMS OVERVIEW



We also train on AI Stack
Reach out to us sak@syqliq.com or
sak@territorialprescience.com
www.territorialprescience.com
www.syqliq.com
+91.9035433124

Artificial Intelligence

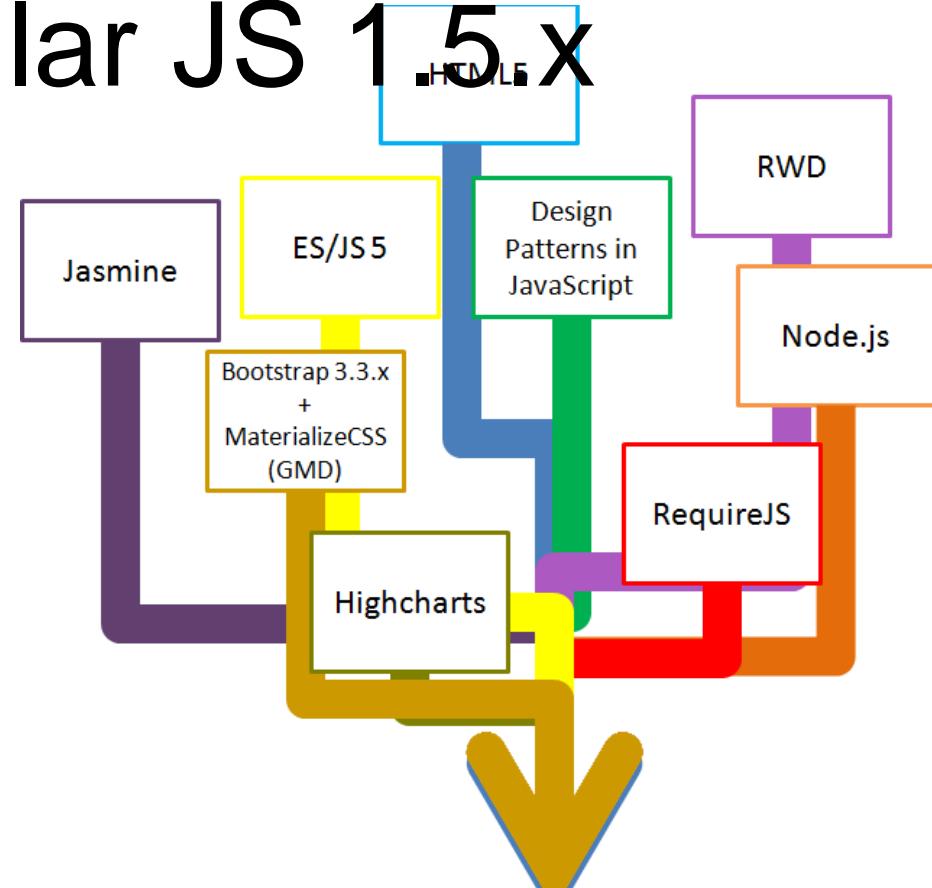




Angular 4.x/Angular JS 1.5.x

Dr. Syed Awase 2016 Session Feedbacks: <http://bit.ly/2hhNg58>

Reach out to sak@territorialprescience.com/+91.9035433124



Original Series

Dr. Syed Awase also offers Machine learning Stack, R Statistical Stack, .NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of performance from here
<http://bit.ly/2hhNg58>

© COPYRIGHT TPRI-SYCLIQ SYED AWASE KHIRNI-2014-17
 LOOPBACK GROUND UP SERIES!



NOW OFFERING!

Code Focused Training

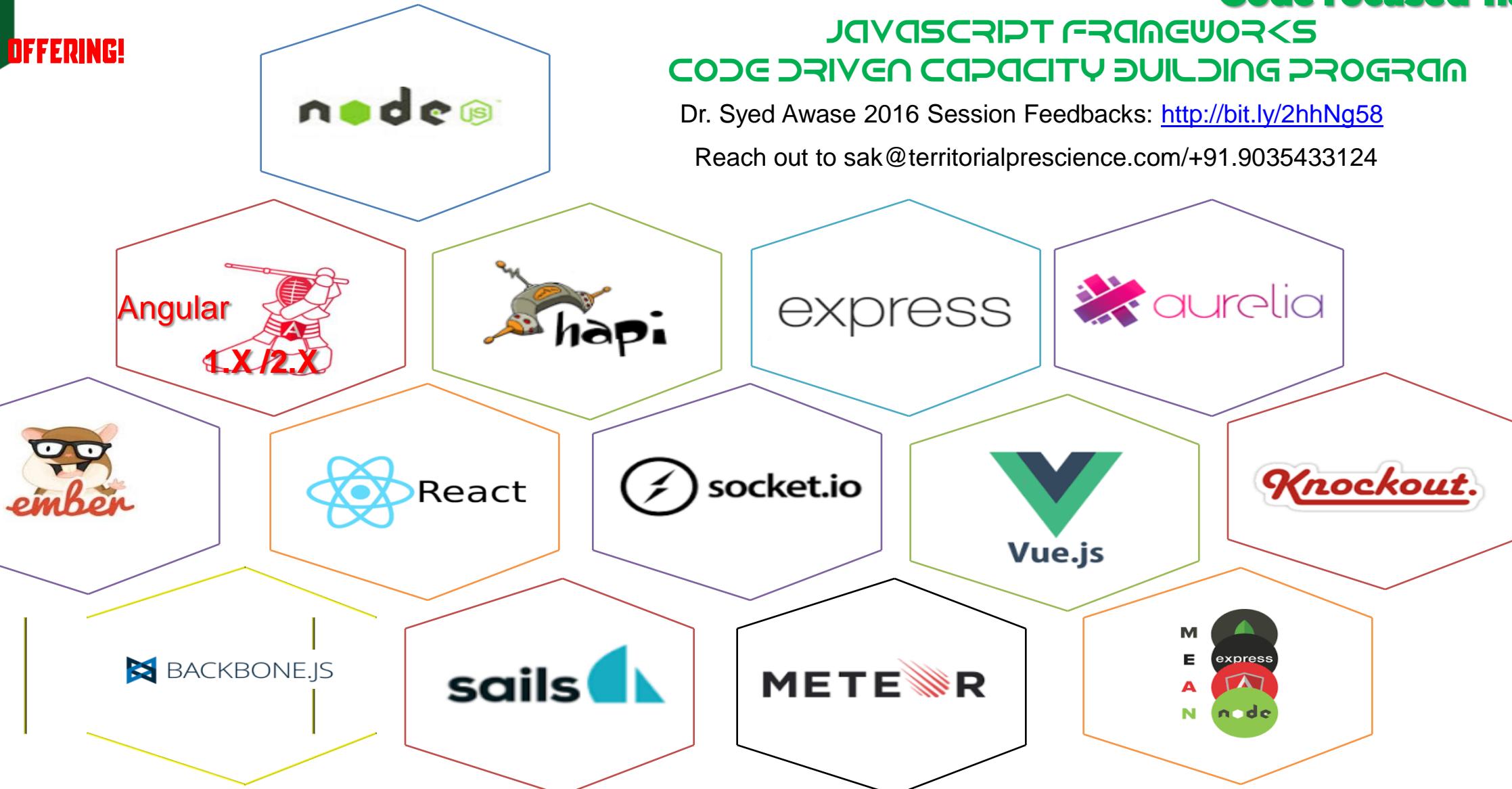
JAVASCRIPT FRAMEWORKS CODE DRIVEN CAPACITY BUILDING PROGRAM

Dr. Syed Awase 2016 Session Feedbacks: <http://bit.ly/2hhNg58>

Reach out to sak@territorialprescience.com/+91.9035433124

Please read terms of use for authorized access

Original Series



Dr. Syed Awase also offers Machine learning Stack, R Statistical Stack, .NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of performance from here <http://bit.ly/2hhNg58> © COPYRIGHT TPRI-SYCLIQ SYED AWASE KHIRNI-2014-17 LOOPBACK GROUND UP SERIES!



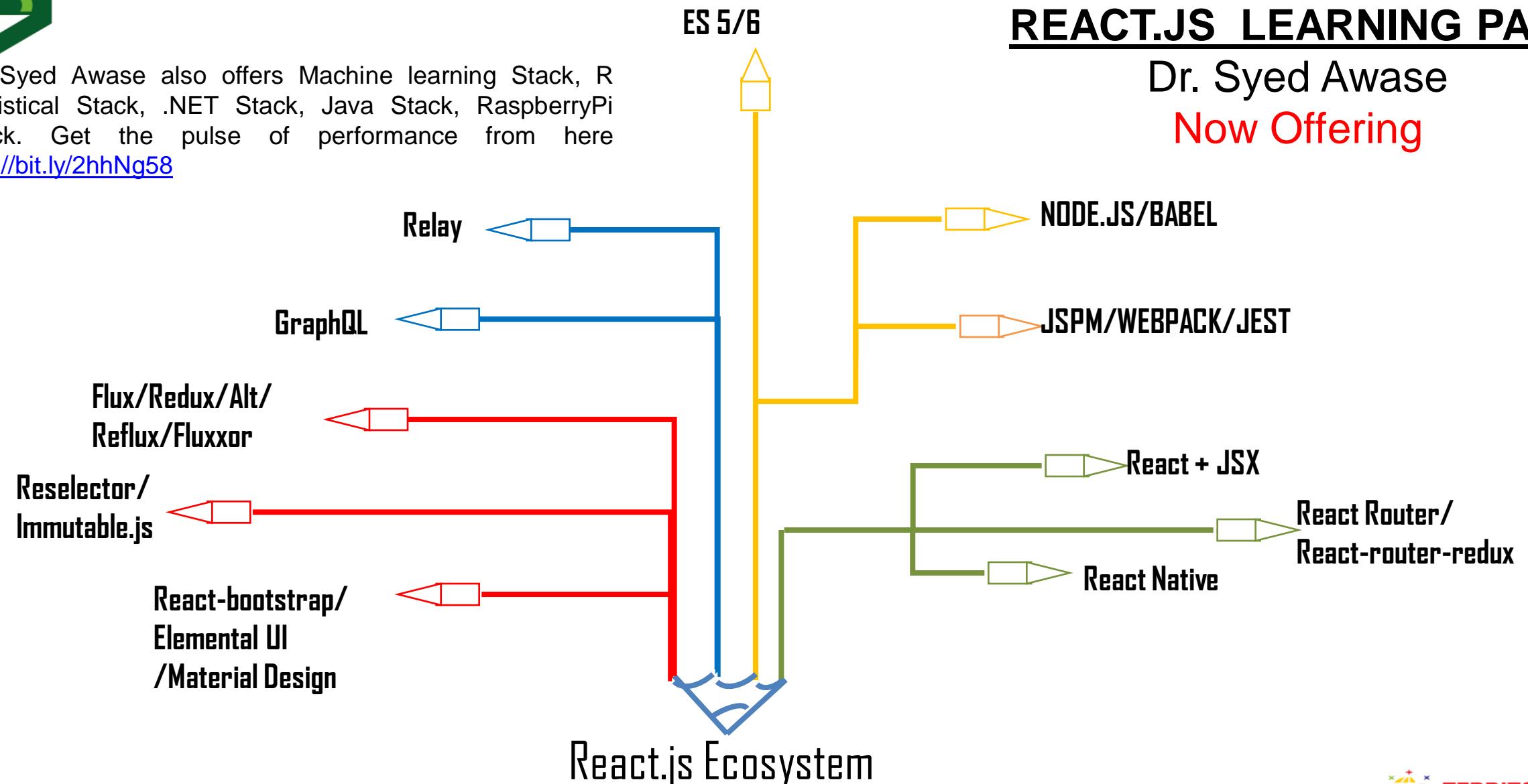
REACT.JS LEARNING PATH

Dr. Syed Awase
Now Offering

Dr. Syed Awase also offers Machine learning Stack, R Statistical Stack, .NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of performance from here
<http://bit.ly/2hhNg58>

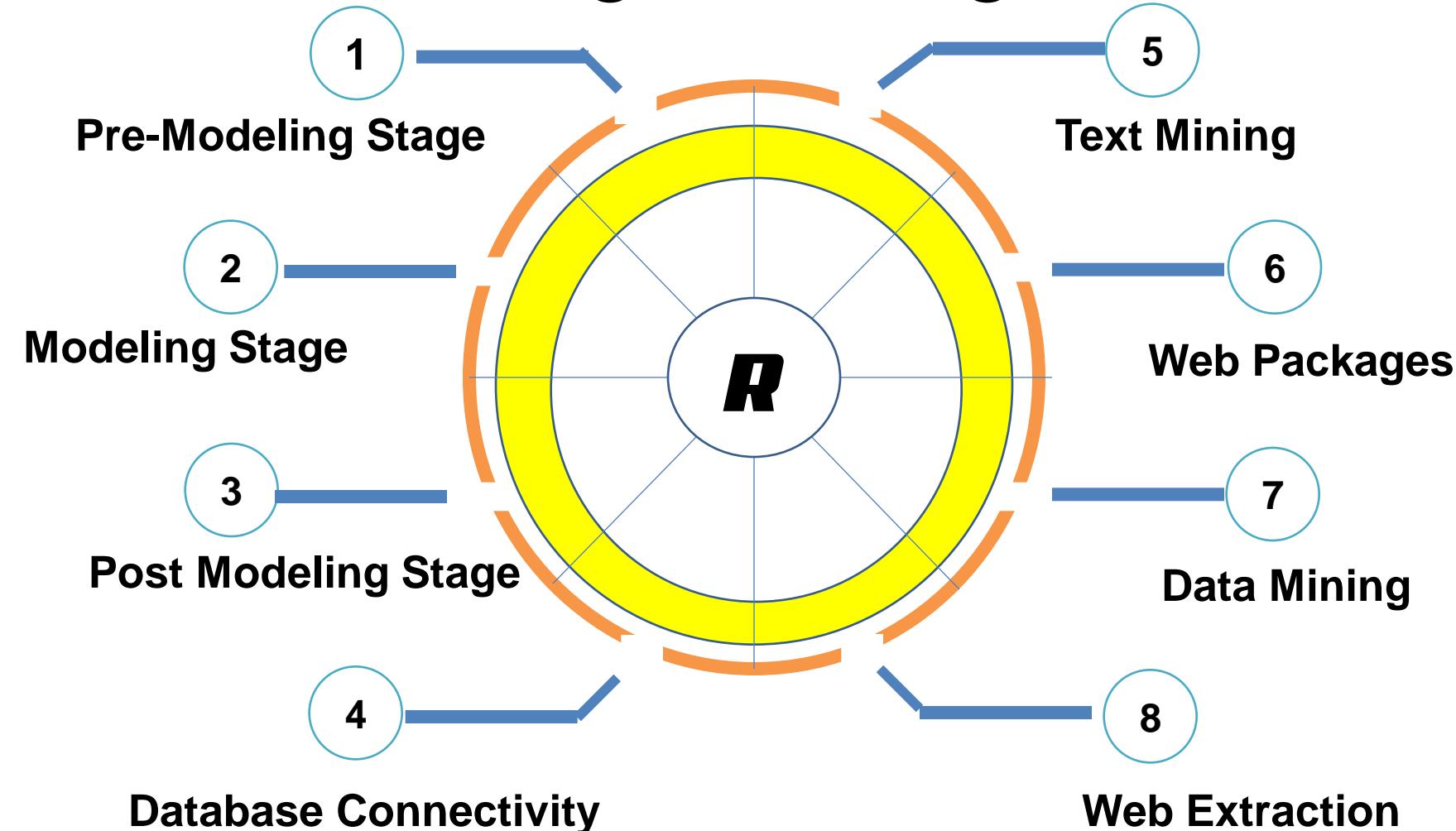
Please read terms of use for authorized access

Original Series





R-Statistical Programming



Please read terms of use for authorized access

Dr. Syed Awase 2016 Session
Feedbacks: <http://bit.ly/2hhNg58>

Reach out to
sak@territorialprescience.com/
+91.9035433124

Original Series

Dr. Syed Awase also offers Machine learning Stack, R
Statistical Stack,

.NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of
performance from here <http://bit.ly/2hhNg58>



For code driven trainings for Technology Firms reach out to us +91-9035433124
 We are hardcore Technologists/Architects/Programmers
 trainings are offered by Dr. SYED Awase

Code Focused Training



Thank You

We also provide Code Driven Open House Trainings : sak@territorialprescience.com or sak@sycliq.com

Please read terms of use for authorized access



Java Technologies

- Core Java
- Hibernate
- Spring Framework
- Play Framework
- Hadoop
- Groovy & Grails



Microsoft Technologies

- C# Core
- Entity Framework
- MVC 5/6
- Web Api
- OWIN/KATANA
- WCF
- WPF



Python

- Python
- Django
- Flask
- Numpy
- Scipy
- Machine Learning



Data Science

- R Statistical Programming
- Julia



SQL and NoSQL

- Oracle
- PostgreSQ L
- MSSQL
- MongoDB
- Neo4j
- Redis
- Firebase
- Apache Cassandra



Client-Side Frameworks

- Angular JS 1.5.x
- Angular 2.4.x
- React JS
- KnockOut JS
- VueJS
- Backbone JS
- EMBER JS
- Hapi JS
- METEORJS
- MEANJS
- Coffeescript
- Dart



Others

- LISP
- CLOJURE
- RUST
- GO
- RaspberryPI
- Coming Soon
- PHP
- Robotic OS