Noonian Documentation

Release 0.9.1

Eugene Newall-Lockett

Aug 22, 2022

Contents:

This documentation is divided into two parts:

Development Guide

A guide meant to be read in sequence to learn how to develop the system.

Reference

The nitty-gritty technical details you'll probably need to refer back to while developing on the system.

Todo: User Guide: general DBUI usage for non-developer end-users of applications built from the DBUI. Showing how to use features such as the query editor, Perspective/column editor, CSV export, ...

CHAPTER 1

Development Guide

This section is will walk you through the key concepts and structures needed for developing an application with Noonian.

1.1 Introduction

1.1.1 What is it?

- It is a MEAN-based integrated platform for rapid development of full-featured browser-based applications.
- It is a framework for defining data objects for persistence, and for creating business logic around those objects.
- It is its own browser-based development environment for full-stack javascript applications.

1.1.2 What can I do with it?

- Define your data objects using meaningful field types.
- Edit those objects within a rich, customizable UI that allows you to build a complete CRUD interface with a few lines of JSON.
- Build your application's UI entirely based on that built-in UI by adding buttons, triggers, web services, user roles and permissions, and custom pages.
- AND/OR build your application's UI to be completely distinct, leveraging Noonian to organize the Angular components, and for its data APIs and web services.

1.2 Getting Started

This section will guide you through the process of installing Noonian and its dependencies, and setting up a fresh instance.

These instructions should generally apply to Linux, Windows, and MacOS hosts.

1.2.1 Installation

The dependencies to required by Noonian are:

1. Node.js
2. MongoDB
3. bower

Mongo DB

Follow the instructions in the MongoDB documentation to get MongoDB installed on your system.

Ideally, you will be able to perform a system-wide installation and run it as a service. However, if you do not have root access to the machine on which you are running, it is possible to run it from a directory under user home directory.

Node.js

Package managers provide the easiest way to get the latest version installed on your system. The nodejs website provides a comprehensive list of available packages for most operating system hosts.

The Node.js installation includes the Node Package Manger *npm <https://npmjs.com>*, which will be used for the rest of the installation process.

Bower

At the commandline, install bower globally using npm:

```
npm install -g bower
```

If you performed a system-wide node install, you'll need to perform the above command as the root or administrative user.

Noonian

At the commandline, install noonian globally with npm:

```
sudo npm install -g noonian
```

NOTE: if you are installing to a system-wide node install (as opposed to a node installation that is owned by a non-root user), there may be issues installing dependencies compiled via node-gyp. To get around the issue, use the "unsafe-perm" parameter:

```
npm install --unsafe-perm -g noonian
```

This will be resovled in a future version that will have refactored out any dependencies that are not pure javascript.

1.2.2 Instance Setup

First, create a directory for your instance, and make it your current working directory.

```
mkdir my-noonian-instance
cd my-noonian-instance
```

Then use the Noonian CLI to initialize the directory as a noonian instance:

```
noonian init my-instance
```

This will create a new file **instance-config.js**, which may be edited to configure the instance.

Edit the configuration file, and then use the CLI to bootstrap the database for the instance:

```
noonian bootstrap -p adminPassword
```

This will initialize the configured database with the core system, setting the initial password for admin to "adminPassword".

Use the CLI to start the instance:

```
noonian start
```

This will start the instance as a background service (forked process), and direct stdout and stderr to stdout.log and stderr.log.

You may stop it with the CLI as well:

```
noonian stop my-instance
```

(if your current working directory is the instance directory, the "my-instance" can be omitted)

1.3 DBUI Basics

Noonian's graphical user interface is referred to as **DBUI**: an initialism for "DataBase User Interface".

The DBUI is what you see when you log in, and provides all of the screens for viewing and editing Business Objects in the system. It provides a rich interface for managing your data, and is designed to serve as a starting point for your database application.

This section will guide you through the key concepts around the DBUI, and how to customize and extend it to build your own application.

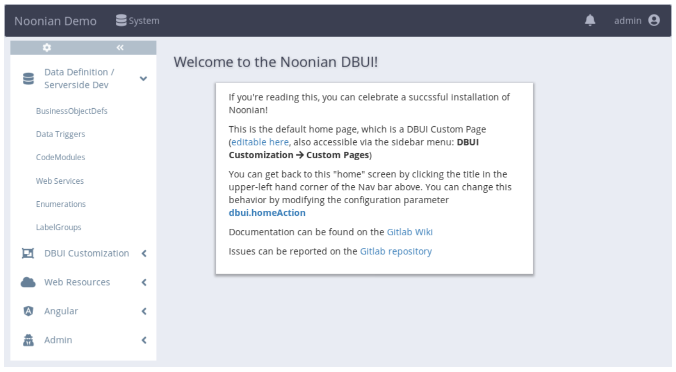
1.3.1 A Tour of the DBUI

When you first log in to a fresh instance, you are greeted with the default home screen:

At the top we've got the **navbar**, on the left we've got the **sidebar**, and the majority of the screen is our main content area.

The Navbar

The top bar we'll refer to as the **navbar** (this is by convention; in actuality most navigation in a Noonian system occurs via the sidebar)



Welcome to the Noonian DBUI!

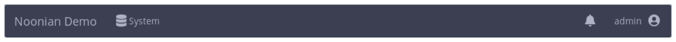
If you're reading this, you can celebrate a succssful installation of Noonian!

This is the default home page, which is a DBUI Custom Page (editable here, also accessible via the sidebar menu: **DBUI Customization → Custom Pages**)

You can get back to this "home" screen by clicking the title in the upper-left hand corner of the Nav bar above. You can change this behavior by modifying the configuration parameter **dbui.homeAction**

Documentation can be found on the Gitlab Wiki

Issues can be reported on the Gitlab repository

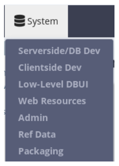


Starting from the left, the navbar components are:

Page Title

Displays the configured instance name. When clicked, takes user to the configurable "home" screen, which is the same screen displayed when the user first logs in.

System Menu



To the right of the title, the configured dropdown menus are displayed. Menus can be customized based on user role.

The **System** menu is the default navbar menu for the system admin role, and shows logical groupings of all *Business Object* classes in the system.

Notifications

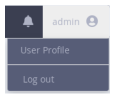
The bell icon on the right-hand side of the navbar contains a list of alert messages that the system has displayed to the user during their session.

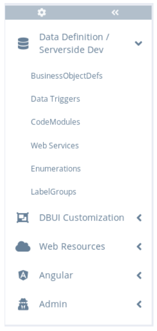
User Menu

The rightmost icon on the navbar is for user profile management (password reset, etc.) and log-out.

The Sidebar

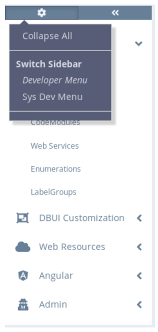
The sidebar is the primary mechanism by which the user navigates a Noonian system. The content of the sidebar is customizable based on user role.





The default sidebar for the system admin role is the *Developer Menu*, providing access to the components used for developing a Noonian application.

Gear Menu



The gear icon on the upper left-hand side of the sidebar allows the user to select which sidebar menu is displayed.

Additionally, it provides an option to collapse all submenus on the sidebar.

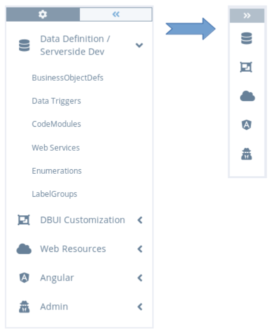
Minimize and Expand

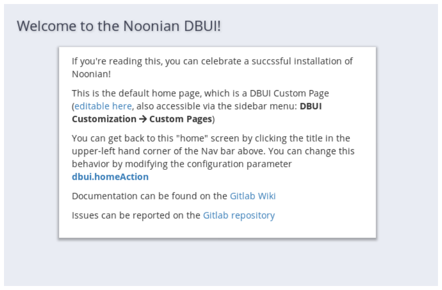
The double chevron icon on the top right of the sidebar provides a way to reduce the size of the sidebar. Clicking it reduces the display to show only icons for each submenu title. In minimized mode, a submenu is exposed when the mouse hovers over its submenu icon.

Home Screen

When the user initially logs in, the main content area is populated with a configurable home screen. This is the same screen that is shown when the user clicks the title on top left of the navbar.

The default home screen on a fresh instance provides information on how to customize it.




Welcome to the Noonian DBUI!

If you're reading this, you can celebrate a succssful installation of Noonian!

This is the default home page, which is a DBUI Custom Page (editable here, also accessible via the sidebar menu: **DBUI Customization → Custom Pages**)

You can get back to this "home" screen by clicking the title in the upper-left hand corner of the Nav bar above. You can change this behavior by modifying the configuration parameter **dbui.homeAction**

Documentation can be found on the Gitlab Wiki

Issues can be reported on the Gitlab repository

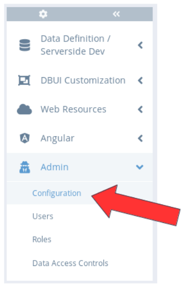
List, Edit, and View Business Objects

The core functionality of the Noonan DBUI is in editing, viewing and querying *Business Objects* in the database. There are 3 primary screens for doing so: **list**, **edit**, and **view**.

List

The **list** screen displays a list of Business Objects in the system. It includes search and query tools and configurable action buttons.

It happens that all of the links on the Developer Menu actually link to a **list** screen. As an example, let's take a look at the configuration list, since there are a number of objects to be seen on a fresh instance:

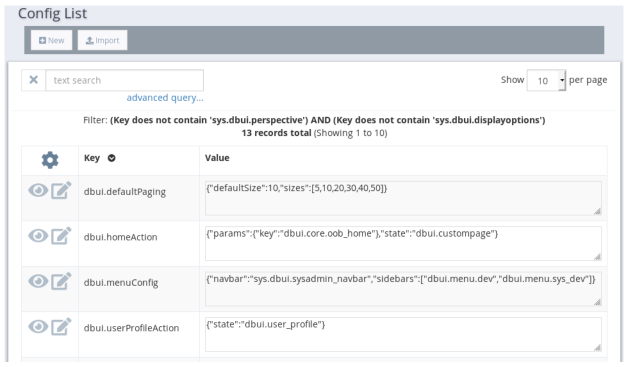


You see a filtered list of **Config** Business Objects in the system, and several buttons and tools.

Action Bar Shows buttons for configured actions for the current *perspective*.

Search/Query Tool Allows for a quick text search, or opening the Query Builder to perform more sophisticated query on the data.

Gear Menu Shows a dropdown to perform table-related actions, such as editing the displayed columns, or refreshing or exporting the data.



Config List

New | Import

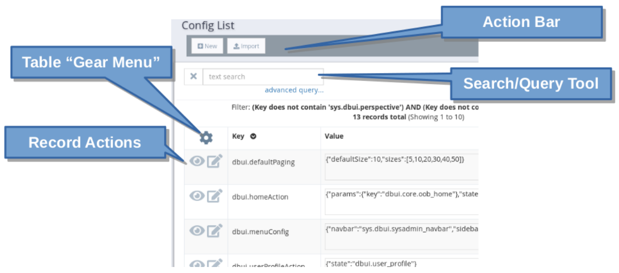
text search

advanced query...

Show 10 per page

Filter: **(Key does not contain 'sys.dbui.perspective') AND (Key does not contain 'sys.dbui.displayoptions')**
13 records total (Showing 1 to 10)

	Key	Value
	dbui.defaultPaging	{"defaultSize":10,"sizes":[5,10,20,30,40,50]}
	dbui.homeAction	{"params":{"key":"dbui.core.oob_home"},"state":"dbui.custompage"}
	dbui.menuConfig	{"navbar":"sys.dbui.sysadmin_navbar","sidebars":["dbui.menu.dev","dbui.menu.sys_dev"]}
	dbui.userProfileAction	{"state":"dbui.user_profile"}

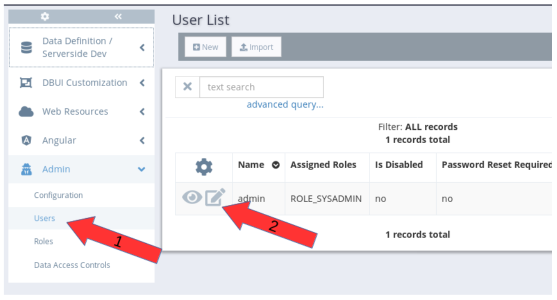


Record Actions Shows record-specific actions that can be invoked for a specific row. The **View** and **Edit** actions are generally present for the default list *perspectives*.

Edit

The **edit** screen displays a form for editing a single Business Object. It includes layout editing tools and configurable action buttons.

As an example, let’s take a look at an **edit** screen for a User Business Object:



User List

New | Import

text search

advanced query...

Filter: **ALL records**
1 records total

	Name	Assigned Roles	Is Disabled	Password Reset Required
	admin	ROLE_SYSADMIN	no	no

1 records total

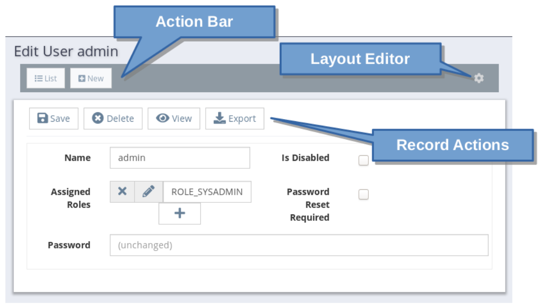
Action Bar Shows buttons for configured actions for the current *perspective*.

Layout Editor Shows a dialog that allows the user to change the fields displayed and the layout of the form.

Record Actions Shows record-specific actions that can be invoked for the object being edited. The **Save**, **Delete**, **View**, and **Export** actions are generally present for the default edit *perspectives*.

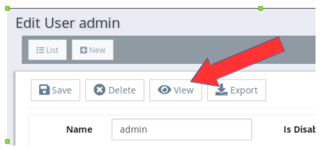
View

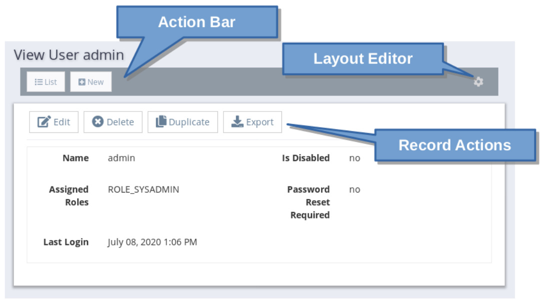
The **view** screen is very similar to an edit screen, the obvious difference being the editability of the displayed Business Object.



Name	admin	Is Disabled	☐
Assigned Roles	ROLE_SYSADMIN	Password Reset Required	☐
Password	(unchanged)		

As an example, let's look at the corresponding **view** for the User object we examined above. Use the **View** button from the edit screen:





Name	admin	Is Disabled	no
Assigned Roles	ROLE_SYSADMIN	Password Reset Required	no
Last Login	July 08, 2020 1:06 PM		

Action Bar Shows buttons for configured actions for the current *perspective*.

Layout Editor Shows a dialog that allows the user to change the fields displayed and the layout of the form.

Record Actions Shows record-specific actions that can be invoked for the object being edited. The **Edit**, **Delete**, **Duplicate**, and **Export** actions are generally present for the default view *perspectives*.

1.3.2 Configuration

1.3.3 Querying Data

TODO

1.3.4 Perspectives and Layouts

TODO

1.3.5 Menus

TODO

1.3.6 Custom Pages

1.3.7 UI Actions

1.4 Data Definition

1.4.1 The Business Object

Business Objects are the basic building blocks of any Noonian application. Defining a Business Object is akin to defining a database table in the SQL world, and resembles class definition in the Object Oriented Programming world.

In Noonian, you define the the data schema for your application by creating a set of **Business Object Definitions**.

When you create a Business Object Definition, you're defining structure for a particular class of Business Object that you want to persist.

That class of Business Object you defined translates to several components in a Noonian system:

1. A corresponding collection in MongoDB that contains the persisted objects
2. An API within the server-side Node.js environment for querying, reading, and updating the objects
3. An API provided by an Angular.js service that allows you to query, read and update from the client-side.
4. A set of screens for viewing, querying, and updating the objects within the Angular.js front-end, called the DBUI.

1.4.2 Creating a Business Object Definition

Expand the sidebar menu **Data Definition / Serverside Dev**; and select **BusinessObjectDefs**

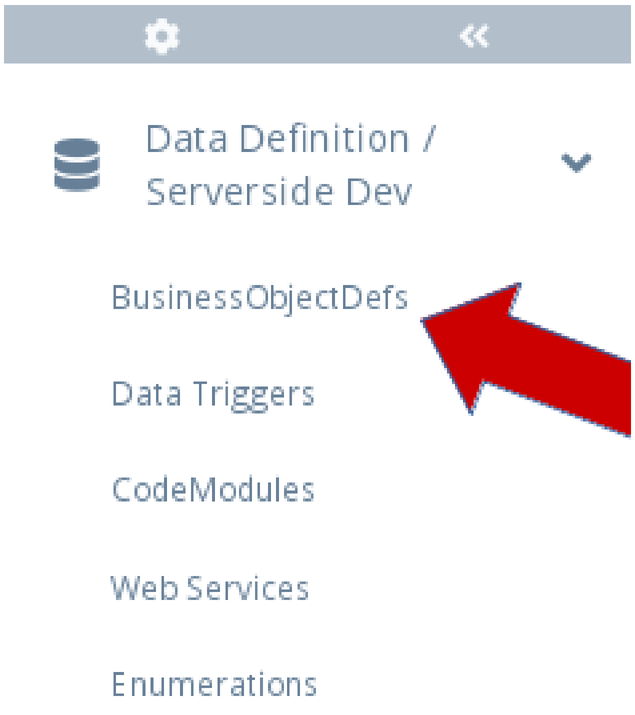


Fig. 1: BusinessObjectDefs Menu

You are shown the list of BusinessObjectDef's in the system. Select the **New** button at the top to create a new one.

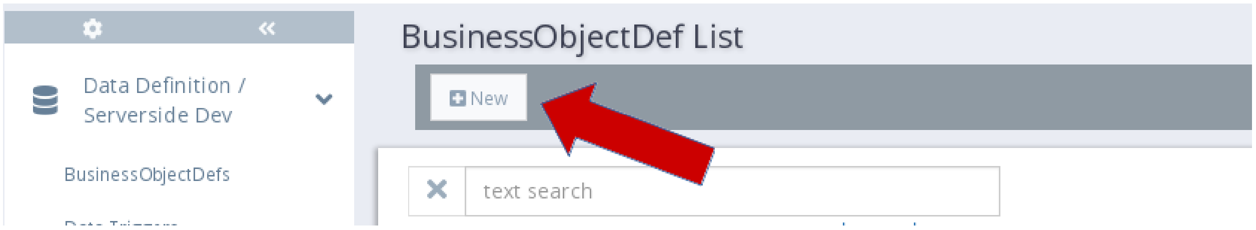
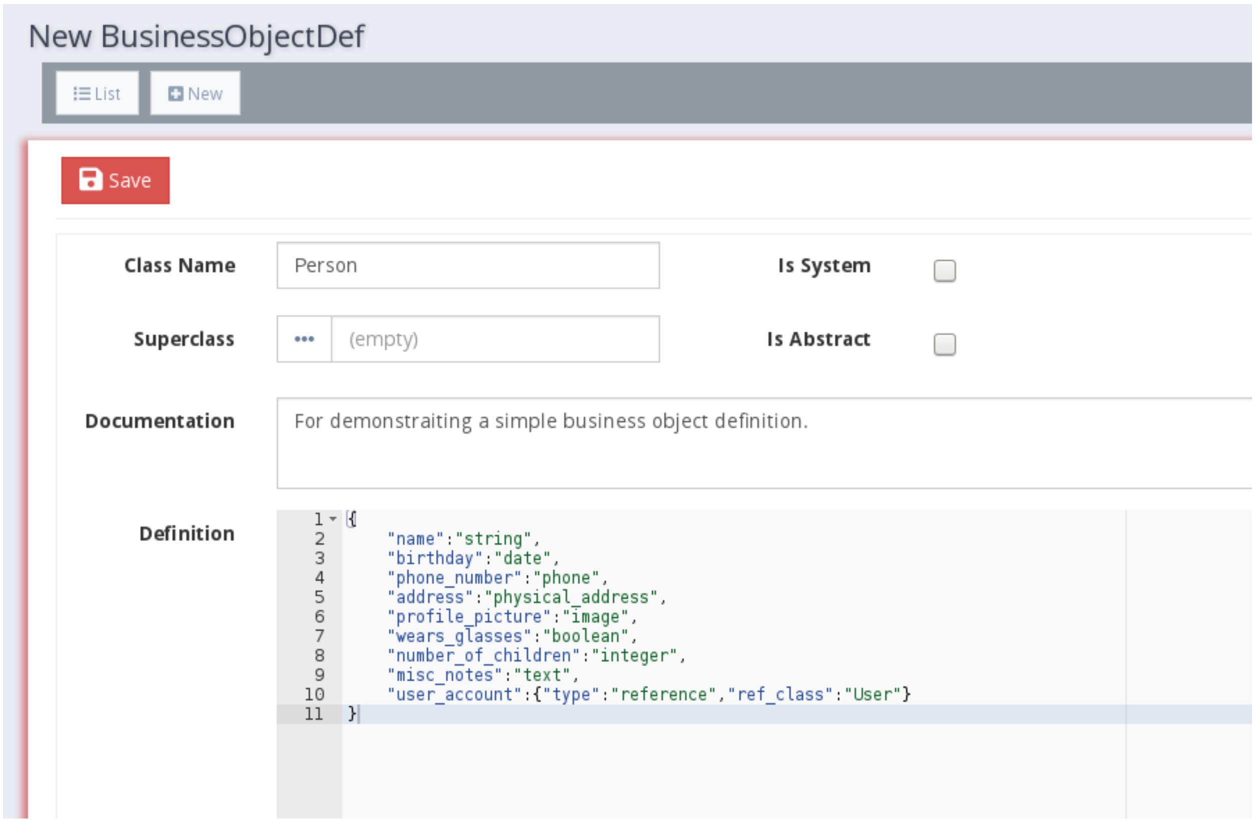


Fig. 2: BusinessObjectDef List -> New

You are shown a form that allows you to specify the fields for your new Business Object Definition. Let's demonstrate with a basic Person class:



Class Name	Person	Is System	☐
Superclass	(empty)	Is Abstract	☐
Documentation	For demonstraiting a simple business object definition.		

Definition

```
{
    "name":"string",
    "birthday":"date",
    "phone_number":"phone",
    "address":"physical_address",
    "profile_picture":"image",
    "wears_glasses":"boolean",
    "number_of_children":"integer",
    "misc_notes":"text",
    "user_account":{"type":"reference","ref_class":"User"}
}
```

Fig. 3: New BusinessObjectDef

The definition is simply a JSON object mapping each field name to its respective **Type Descriptor** :

```
{
    "name":"string",
    "birthday":"date",
    "phone_number":"phone",
    "address":"physical_address",
    "profile_picture":"image",
    "wears_glasses":"boolean",
    "number_of_children":"integer",
    "misc_notes":"text",
```

(continues on next page)

(continued from previous page)

```
    "user_account":{"type":"reference","ref_class":"User"}
}
```

Notice in the example, most of the types are described simple strings: “string”, “date”, “boolean”, etc. This is a shorthand for those types that require no more than a single string. The user_account field is the exception: its type is “reference”, so we need to specify what class of objects it should reference.

Also perhaps you’ve noticed how the fields “name” and “misc_notes”, have the types “string” and “text”, and you’re wondering the purpose of the distinction - isn’t a “text” field is just a long string? Indeed it is! However, by calling it a “text” field, we are 1) creating a more precise description, 2) telling the display logic to use the larger text block for editing it, and 3) telling the system to index it differently.

Let’s go ahead and save the BusinessObjectDef. Notice how those types get expanded after you save:

```
{
  "name": {
    "type": "string"
  },
  "birthday": {
    "type": "date"
  },
  "phone_number": {
    "type": "phone"
  },
  "address": {
    "type": "physical_address"
  },
  "profile_picture": {
    "type": "image"
  },
  "wears_glasses": {
    "type": "boolean"
  },
  "number_of_children": {
    "type": "integer"
  },
  "misc_notes": {
    "type": "text"
  },
  "user_account": {
    "type": "reference",
    "ref_class": "User"
  }
}
```

When you save the a BusinessObjectDef, any single-string values get replaced by a full **Type Descriptor**. The **Type Descriptor** fully describes a field’s type, and at its simplest consists of a single “type” property. Other properties on a Type Descriptor add neccessary detail in describing the fields type. Some are mandatory (e.g. “ref_class” for a reference field) , and others are optional (e.g. you may specify a minimum or maximum value of an integer field with a min or max property.)

Please see the reference page for Field Types for a complete list of available field types and their respective type descriptor properties.

1.4.3 Indexing

coming soon

Todo: pending implementation of BusinessObjectDef index specification

1.5 Adding Business Logic

This section is will walk you through the structures available to you when developing business logic in your Noonian application.

1.5.1 Data Triggers

TODO

1.5.2 Web Services

TODO

1.5.3 Code Modules

TODO

1.5.4 Member Functions

TODO

1.5.5 Schedule Triggers

TODO

1.6 Adding Users and Roles

1.6.1 Menus and Perspectives

TODO

1.6.2 Data Access Controls

TODO

1.7 Installing and Building Packages

TODO

1.8 Documentation

Most of the Business Objects you create in the process of building an application have a **Documentation** field you can use to include relevant documentation for the respective components. These fields support JSDoc and reStructuredText, and can be used to generate API and development documenation for your Noonian applications.

TODO

Todo: pending implementation of Noonian Doc Generation

..todo:: Application Diagrams add-on

1.9 Application and Unit Testing

Noonian integrates Jasmine, Selenium, and Protractor for creating unit tests for your applications.

TODO

Todo: Pending implementation of Jasmine/Selenium/Protractor integration

1.10 Logging

TODO

1.11 Extended DBUI Features

This section takes you through several useful DBUI features not yet mentioned in this guide.

1.11.1 DBUI Pivot Page

TODO

1.11.2 Reporting

TODO

1.11.3 Internationalization

TODO

1.12 Building a Non-DBUI Angular Application

The guide thus far has show how it is possible to build a full-featured application by extending and customizing the DBUI. This section describes how you can use Noonian to build an Angular.js application that is not based on DBUI. In this scenario, it is still possible to leverage the DBUI and Noonian's data layer and APIs for your application.

A practical example of a use case might be a lightweight Angular-based front-end that leverages Noonian for the data layer and business logic, and the DBUI as an administrative back-end.

Note: A basic understanding of Angular.js compoeenents is assumed for this section

1.12.1 Angular Components

1.13 Advanced Customization

This section is describes how to extend the fundamental objects of Noonian and the DBUI.

1.13.1 Custom Field Types

TODO

1.13.2 Custom Query Operations

TODO

CHAPTER 2

Indices and tables

- *Glossary*
- genindex
- modindex
- search

2.1 Glossary

2.1.1 Noonian Terminology

Business Object A persisted data object.

DBUI The graphical user interface of Noonian: “DataBase User Interface”

Perspective A configuration object that describes how to render a DBUI screen: layout, table columns, action buttons, etc.

RecordAction An action within the DBUI that is tied to a Business Object when invoked.

2.1.2 Related Jargon

MEAN A stack of technologies for web development that consists of Javascript and JSON for both front and back-end: M = MongoDB E = Express A = AngularJS N = Node.js

CHAPTER 3

Noonian Reference

Detailed technical documentation for Noonian development and administration.

3.1 Admin, Configuration, and Deployment

3.1.1 Instance Configuration

Configuration for a Noonian instance is stored in a javascript file in the base of the instance directory: **instance-config.js**. A default is generated from a template when an instance is initialized using the CLI.

Configuration File

Required Parameters

At instance initialization, all required parameters are populated/generated with reasonable defaults.

instanceName

A name used for referring to the instance via the CLI. Should be unique across instances on a server for the CLI to work properly.

instanceId

An identifier used in versioning the persisted business objects. It should be short; by default it matches the instance name.

serverListen

TCP Port and address on which to bind. On initialization, the Noonian CLI examines the config files for all configured instances on the machine and chooses a port number by adding 1 to the highest one found.

```
serverListen: {
  port: 11100,
  host: '127.0.0.1'
}
```

mongo

MongoDB connection string and options. Initialization defaults to localhost

```
mongo: {
  uri: 'mongodb://localhost/noonian-myinstance'
}
```

secrets

Secret string used for signing the auth tokens for authentication. Initialization generates a random UUID.

```
secrets: {
  session: '1f6885e7-36a8-4dd9-909e-d585e5bd0879'
}
```

Logging

By default, log level is set to debug, writing logs to instance.log (all log messages) and error.log (only error messages).

Logging can be configured via either of the following parameters.

Alternatively, logging can be configured via the logging-config.js in the instance directory.

logLevel

For simple configuration if the default files/formats are sufficient. Set it to one of:

error, *warn*, *info*, *verbose*, *debug*, *silly*

Logs messages at or above the configured level are written to instance.log, and errors to error.log.

loggers

Provides more detailed control over log formats and transports.

Optional Parameters

urlBase (string)

A path under which the noonian instance will be served.

Use urlBase if noonian will be served to a path other than the root of the domain. For example, one domain can host multiple noonian instances as such: mydomain.com/instance1, mydomain.com/instance2

enableBackRefs

Enable "back reference" processing on reference fields.

enableHistory

Instance Directory Structure

The files and directories in the instance directory are described below.

- **instance-config.js**
- **client** - base directory for static content served from the filesystem by the webserver (this directory is checked BEFORE client directory of the core app listed above) * **bower_components** - holds bower dependencies of packages installed on this instance
- **node_modules** - holds any npm dependencies of packages installed on this instance
- *[pkg-key]* - if package building with filesystem sync is enabled, data updates are synced to this directory

3.1.2 Command Line Interface

```
Usage: noonian [options] [command]

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

Commands:
  start [options] [instanceName]       launch an instance
  stop [options] [instanceName]        stop a running instance
  restart [options] [instanceName]     restart a running instance
  env [options] [instanceName]         show environment variables for launching an␣
↪instance (returns source-able list of variable export lines)
  list                                 list configured instances
  status                               list the currently-running Noonian instances
  startall                             Start all of the configured Noonian instances
  init [options] <instanceName>        initialize directory for a instance
  bootstrap [options] [instanceName]   bootstrap database for an instance
  add [options]                        add an existing instance to the index
  remove <instanceName>                remove an instance from the instance index
  dbdump [options] [instanceName]      call mongodump to create a dump of the database
  dbshell [options] [instanceName]     start mongo shell for the instance's database
  open [options] [instanceName]        launch browser window
  watch [options] [instanceName]       watch stdout and stderr of an instance
  pm2-eco                              generate pm2 ecosystem JSON (print to stdout)
```

Instance Set-up

init, bootstrap, add, remove

Process Management

The CLI contains basic functionality for managing Noonian instances on the local machine. These features are meant for use in a development envrionment. For a production deployment, either a Docker container or a more sophisticated process management tool such as PM2 is recommended.

Start / Stop / Restart

Status

PM2 Ecosystem Generation

Utilities

Database Dump

Database Shell

Get Environment Variables

Installing Bash Autocompletion

The Noonian CLI has a bash auto-completion script so commands and instance names can be auto-completed at the command prompt.

If you have root access to the system, you can create a symbolic link in **/etc/bash_completion.d** to the NOONIAN_HOME/template/bash_completion.d

```
ln -s /path/to/installation/node_modules/noonian/template/bash_completion.sh /etc/
↪bash_completion.d/noonian
```

If you do not have root access, you can simply source the bash_completion.sh in your .bashrc. (Alternatively, refer to this discussion for other options)

3.1.3 Deployment

This document describes various options for production deployment of a Noonian instance.

PM2

Run Noonian as a service using PM2

Proxy through Apache

Docker

There are multiple scenarios for deploying Noonian using docker:

- All-in-one, single instance
- All-in-one, multi-instance
- External MongoDB, single-instance Noonian container
- External MongoDB, multi-instance Noonian container

All-in-one Container

Full Node + MongoDB + Noonian stack in one image/container.

Docker-Compose Stack

Two separate containers: one MongoDB, one Node.js/Noonian

3.2 Development Reference

This documentation provides detailed technical information for Noonian development to supplement the API documentation.

3.2.1 Data Model

3.2.2 Indexing

TODO

3.2.3 Query Ops

TODO

3.2.4 References

TODO

3.2.5 Config

TODO

3.2.6 Auth

TODO

3.2.7 Data Export

TODO

3.2.8 GridFS

TODO

3.2.9 Packaging

3.2.10 Web Sockets

TODO

3.3 Noonian Core API

TODO: generate from sourcecode/jsdoc

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index