

# Build your first Web App

## Build a Responsive Web App

## 1 Introduction

This guide will help you gain an understanding of the key features and components of Stack9 using a hands-on demonstration of the system.

### 1.1 Getting to know Stack9

Stack9 exists of 2 components or websites: a front-end application that your end-users will see and interact with, and a back-office console application where you will configure and build the application. Both are browser-based.

In this guide you will learn how to add an entity to your **Domain Model** (a visual model that describes your app's information or data in an abstract way). You will also get an understanding of how the **Query Library** allows you to build a toolbox of queries using **Connectors** between your data and the screens on it which that data is presented.

You will also set up **Modules** that will allow you to browse the data in your application via **Screens**, as well as validate information you capture using **Automations**. Lastly you will create a **Document Template** to allow you to export data in a user-friendly manner.

Lastly you will use the companion **front-end application** to browse the changes you made and create, update and print information in your application.

## 2 Prerequisites

You will need to have the following in place before continuing:

1. Browser access to a running Stack9 console app
2. Admin permissions to the console to allow you to configure and update components relevant to this guide.

## 3 Choosing an App template

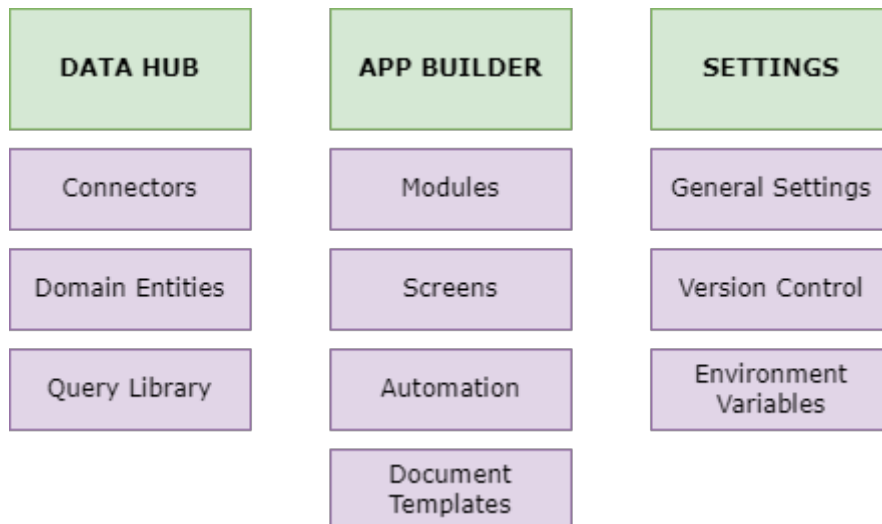
More details to follow...

# 4 Exploring the Stack9 Console

Before you start working in your app's console, take a look a look around to familiarise yourself with the various components; some of which you will be using in this guide.

Launch the Stack9 Console in your browser (from the following link): **xxxx-console.app.stack9.dev** and login with your supplied user credentials. If using single sign-on (SSO), you may need to accept your organisation policies the first time you do so.

Once signed in you will see the Console home page with the following navigation options:



## 4.1 Data Hub

Possibly too detailed? Happy to compact these down into a single para if too detailed for this section...

### 4.1.1 Connectors

Connectors allow your application to communicate with a variety of internal and external data sources. Stack9 comes with a set of preconfigured connectors already installed and configured ready to use.

STACK9

Space home

Entity models

Connectors

Query library

Automations

Apps

Screens

Environment Variables

Templates













...

R.S

Connectors

🔍 Search...

Create new

Name	Type	Description	Created at	Updated at	Action
 <a href="#">Stack9 API</a> 	<code>stack9_api</code>	Stack9 API Connector			
 <a href="#">Stack9 Database</a> 	<code>stack9_db</code>	Stack9 Database Connector			
 <a href="#">Stack9 Document Database</a> 	<code>stack9_mongo</code>	Stack9 Document Database Connector			
 <a href="#">Stack9 Cached Resources</a> 	<code>stack9_redis</code>	Connect to Stack9 cached resources			
 <a href="#">Stack9 Files S3</a> 	<code>stack9_s3</code>	Stack9 Connector for AWS S3			
 <a href="#">Stack9 Files Azure</a> 	<code>stack9_azure</code>	Stack9 Connector for Azure Blob			

See [Connectors Reference](#) page for more detailed information

For this simple CRM demonstration we will focus on using the **Stack9 API** connector.

You can also create and configure your own connectors or have April9 develop custom connectors for almost any use case.

### 4.1.2 Domain Entities

This is where all the tables for your application will be kept. Think of these tables as storage folders for data created and managed within your application. Entities can be linked to each other using relational database concepts.

Stack9 comes with a default set of tables (native entities) that cannot be changed or removed but you can extend them with your own custom fields. [Click here](#) for an explanation of each native entity.

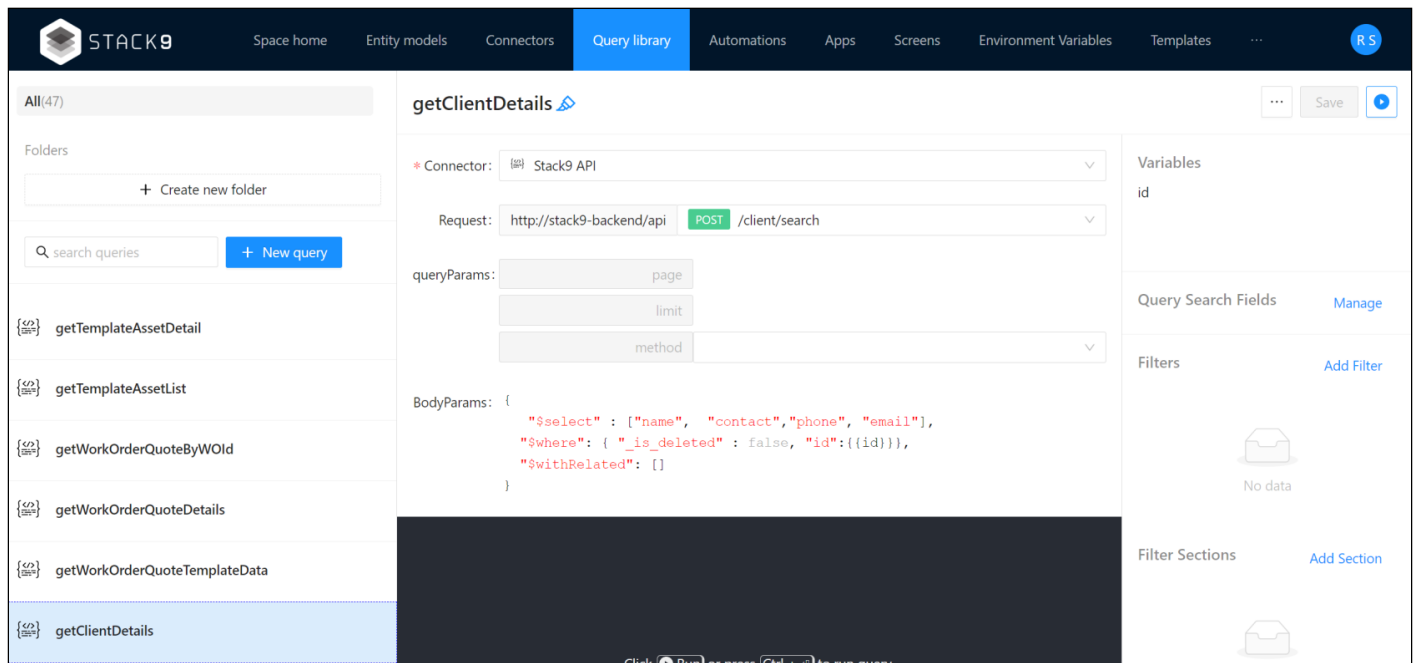
<div> <div>STACK9</div> <div> <a href="#">Space home</a> <a href="#">Entity models</a> <a href="#">Connectors</a> <a href="#">Query library</a> <a href="#">Automations</a> <a href="#">Apps</a> <a href="#">Screens</a> <a href="#">Environment Variables</a> <a href="#">Templates</a> <a href="#">...</a> </div> <div>RS</div> </div>																															
<div> <div>Entity models</div> <div> <input type="text" value="Search..."/> <div>Add Entity model</div> </div> </div>																															
<div> <div> <div>All(36)</div> <div>Custom(22)</div> <div>Native(14)</div> <div> <div>Folders</div> <div> <div>ADMIN 4</div> <div>CRM 3</div> <div>JOB 11</div> <div>+ Create new folder</div> </div> </div> </div> </div>																															
<table> <tr> <th>Entity</th><th>Type</th><th>Last modified at</th><th></th></tr> <tr> <td><a href="#">api_key</a></td><td>native</td><td>5/5/2023 3:50:48 PM</td><td>...</td></tr> <tr> <td><a href="#">app_registration</a></td><td>native</td><td>5/5/2023 3:50:48 PM</td><td>...</td></tr> <tr> <td><a href="#">client</a></td><td>custom</td><td>5/5/2023 3:50:48 PM</td><td>...</td></tr> <tr> <td><a href="#">client_markup</a></td><td>custom</td><td>6/7/2023 5:51:46 PM</td><td>...</td></tr> <tr> <td><a href="#">client_site</a></td><td>custom</td><td>5/5/2023 3:50:48 PM</td><td>...</td></tr> <tr> <td><a href="#">contractor</a></td><td>custom</td><td>5/5/2023 3:50:48 PM</td><td>...</td></tr> </table>				Entity	Type	Last modified at		<a href="#">api_key</a>	native	5/5/2023 3:50:48 PM	...	<a href="#">app_registration</a>	native	5/5/2023 3:50:48 PM	...	<a href="#">client</a>	custom	5/5/2023 3:50:48 PM	...	<a href="#">client_markup</a>	custom	6/7/2023 5:51:46 PM	...	<a href="#">client_site</a>	custom	5/5/2023 3:50:48 PM	...	<a href="#">contractor</a>	custom	5/5/2023 3:50:48 PM	...
Entity	Type	Last modified at																													
<a href="#">api_key</a>	native	5/5/2023 3:50:48 PM	...																												
<a href="#">app_registration</a>	native	5/5/2023 3:50:48 PM	...																												
<a href="#">client</a>	custom	5/5/2023 3:50:48 PM	...																												
<a href="#">client_markup</a>	custom	6/7/2023 5:51:46 PM	...																												
<a href="#">client_site</a>	custom	5/5/2023 3:50:48 PM	...																												
<a href="#">contractor</a>	custom	5/5/2023 3:50:48 PM	...																												

Custom entities are the tables that will make up your own domain model. This guide will take you through creating some examples.

See the [Domain Model](#) section for more detail on how to set up your domain's entities.

### 4.1.3 Query Library

The Query Library is a central store for all the instructions your application will need in order to Create, Read, Update and Delete data in your domain model - often referred to as CRUD operations.



It is recommended that you follow a set naming convention when creating queries. For example: **get<entityname>List** for queries that return a summary list of records in a table and **get<entityname>Details** for the full details of a specific record.

Queries can also be arranged in folders for easier management, particularly as your application grows in size.

## 4.2 App Builder

### 4.2.1 Modules

This where you will set up and manage the overall navigation and menu of your application. You can separate the areas of your application into business units, departments or by role to group similar information together.

Key	Name	Description	Updated at	Created at	Action
admin	ADMIN	Administration	6/15/2023 8:51:33 PM	6/12/2023 3:51:32 PM	...
crm	CRM	Client relationship management	6/12/2023 3:51:32 PM	6/12/2023 3:51:32 PM	...
jobs	JOBS	Job management	6/12/2023 3:51:32 PM	6/12/2023 3:51:32 PM	...

## 4.2.2 Screens

Screens are the primary interface through which users of your application will interact with the underlying domain model and data contained within it.

App	Route	Title	Screen type	Updated at	Action
admin	admin/template-asset/create admin/template-asset/id admin/template-asset/list admin/template-asset	Template Asset	simpleCrud	6/15/2023 8:50:52 PM	...
jobs	jobs/work-order/create jobs/work-order/id jobs/work-order/list jobs/work-order	Work orders	simpleCrud	6/13/2023 2:30:12 PM	...
admin	admin/user/create admin/user/id admin/user/list admin/user	user	simpleCrud	6/12/2023 3:51:32 PM	...

## 4.2.3 Automation

The Automation section is for more advanced concepts where you can configure particular events to happen when data changes. For example, when a record is created or updated, some specified validation rules can be run to ensure required fields are included or data is captured with a valid format; such as an email address.

[Space home](#)
[Entity models](#)
[Connectors](#)
[Query library](#)
[Automations](#)
[Apps](#)
[Screens](#)
[Environment Variables](#)
[Templates](#)

All (9)

afterCreate (4)

afterUpdate (2)

afterDelete (1)

afterWorkflowMove (0)

afterWorkflowOwnershipTaken (0)

cronJob (0)

mqHandler (0)

webhook (2)

+

Create new folder

Automations

Create new

Key	Name	Entity Key	trigger type	Updated at	Action
addressfinderaddressinfo	AddressFinderAddressInfo	client_site	webhook	6/12/2023 3:51:32 PM	...
addressfinderautocomplete	AddressFinderAutoComplete	client_site	webhook	6/12/2023 3:51:32 PM	...
after_work_order_is_created	After work order is created	work_order	afterCreate	6/13/2023 2:30:12 PM	...

## 4.2.4 Document Templates

Data in your application can be printed or exported in a printable format. This section allows you to set up and configure templates and master templates that will be accessible from record and list menus in areas of your application that you specify.

[Space home](#)
[Entity models](#)
[Connectors](#)
[Query library](#)
[Automations](#)
[Apps](#)
[Screens](#)
[Environment Variables](#)
[Templates](#)

All (3)

getworkorderquotetemplatedata(3)

template

master template

Templates

Create new template

Create new master-template

Key	Name	Query Key	Use in grid	Test Data	Template	Updated at	Action
job_sheet	Job sheet	getworkorderquotetemplatedata	Yes	{}	<h1>{{client_site.cl...	6/16/2023 5:57:45 PM	...
client_quote	Client quote	getworkorderquotetemplatedata	No	{ "_updated_at": "2...	<table style="width...	6/16/2023 5:51:32 PM	...

Master templates allow you to add re-usable content that can be used in other documents templates, such as a logo and header and footer information, removing the need to maintain them in each document template.

## 4.3 Settings

General settings and configuration of your application... more details to follow

### 4.3.1 General Settings

This is where you can control the site-wide settings of your application, such as its title, logo and branding, icons and themes.

### 4.3.2 Version Control

Your application will grow and change over time. This section allows you to set up and control versions of your application that can be released in a staged and controlled manner. You will also be able to downgrade your application to an older version in some situations.

### 4.3.3 Environment Variables

Your application can be deployed to different environments for different purposes or audiences. For example, a Development (Dev) environment with limited access that you and your development team can use to start setting up or used when developing new entities, features or screens prior to release into Production. Or a UAT environment where stakeholders can test the application or any new features for acceptance prior to release.

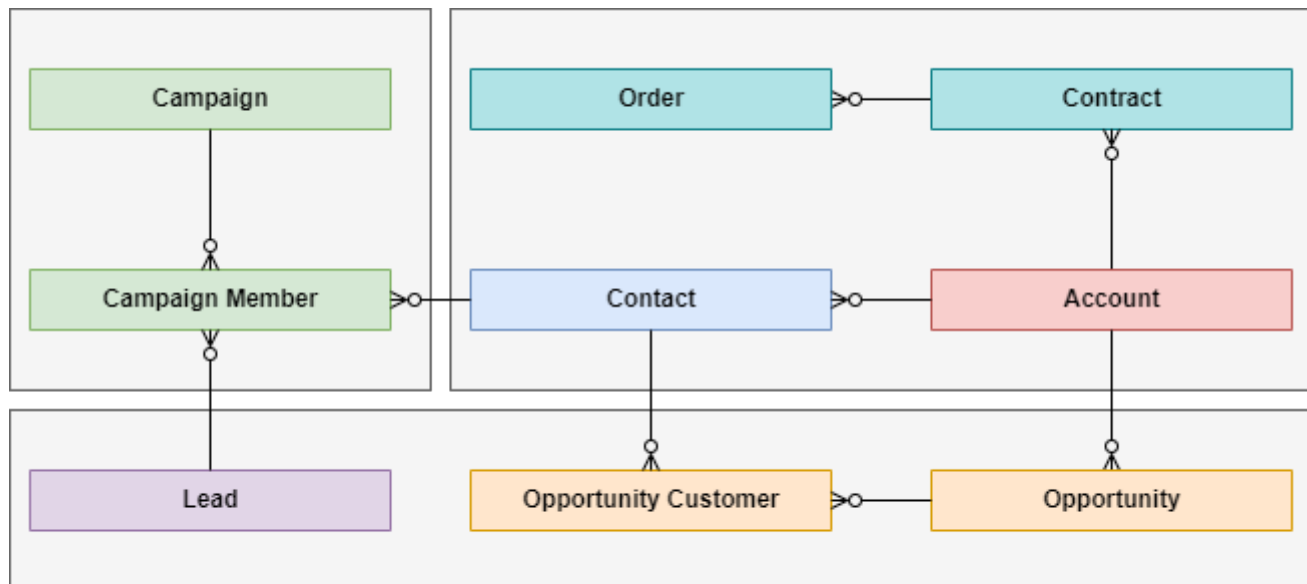
A place to store information and keys that need to be different on each of your environments.

## 5 Creating your app's Domain Model

A Domain Model is central to your application's data structure. It holds a blueprint for the tables that will hold all the information and content in your app.

### 5.1 Understanding the Domain Model

The domain model is a visual representation of your app's database. The Simple CRM domain model we're using for this guide looks as follows (too detailed - will review and simplify):



Simple CRM Domain Model

### 5.2 Creating an Entity to Store Information

We'll start by creating an **Account** (or Customer?) entity to hold information about organisations whose information and related interactions and entities the system will hold.

1. In the Console, choose Data Hub > **Entity Models** and choose **Add Entity Model**.
2. You will be prompted to enter the following information:
  - Name: **Account**
  - Entity Key: **account** - lowercase is the preferred format
  - Pluralised Name: **Accounts** - this is how the entity will be referred to in titles and menus where a plural version of the entity is required
  - Is active: **Checked**
  - Leave the following fields at their default values for now:
    - Allow Comments: checked
    - Allow Tasks: checked
    - Attachment Max Size: empty
    - Attachment File Types: empty
    - Folder: empty
3. Click **Create**.

Your entity is now created and will be listed in the Custom folder. Next you need to add fields to the entity:

1. Open the entity (or click Edit account entity after the last step above) and click the **Add field** button to bring up a list of field types to choose from.
2. Choose Text and you will then be prompted to enter the following information:
  - Name: **Company Name**
  - Field ID: **company\_name** - lower case with no spaces is recommended
  - Placeholder: this is optional but can be used to prompt users when they capture information, e.g. **Enter the business's Trading Name**
  - This Field is Required: **Checked**
  - The remaining fields are optional and will be covered in other guides so you can leave them at their default values.
3. Click **Create**
4. Repeat the Add field process to create the following fields in the Account entity:
  - **Account No**: use the *Number* field type (set Precision to 0 as this will be normally be a whole number)
  - **Status**: use the *Choice* field type and under Values available enter the following values:
    - Active
    - Inactive
    - Archived
  - **Notes**: use the *Rich Text* field type
5. You can use the Settings option to change any of the fields you have already created. You can also drag the fields to re-order them.
6. Once you are done adding fields, click **Save changes**.



Refer to [Field Types page](#) for details of the different types of field you can add to your entity.

Next we will create a second entity to hold customer (or account) contact information:

1. Choose Add Entity Model and name it **Contact** following the steps above.
2. Then add the following *Text* fields to it:
  - **Full Name** (read-only)
  - **First Name** (required)
  - **Last Name** (required)
  - **Phone**
  - **Mobile No.**

Next we will add a field that will link the contact to the an account. As an account could have many contacts, we don't want to make users enter the same information for each, so instead we will allow users to choose the account from a dropdown menu.

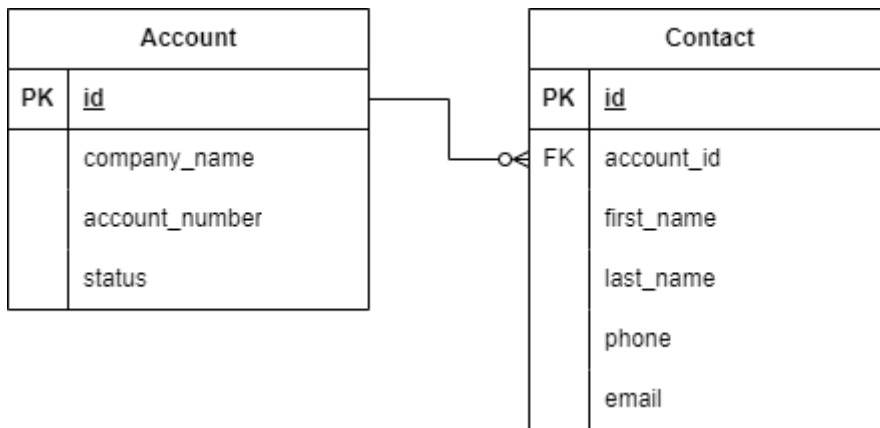
1. Add a field using the N-1 Reference (single dropdown) field type with the following details:
  - Name: **Account**
  - Field ID: **account\_id** (the `_id` suffix is important)
  - Placeholder: **Choose account...**
  - Select your entity: choose **Account** from the list
  - Label Property: **Company Name**
2. Click **Create**
3. Drag the Account field to be first in the list so it will always appear at the tope of the form by default.
4. Then make sure you click **Save changes** to update the entity.

Now that we have a contact entity linked to the account entity, we can go one step further and allow the account entity to list all contacts for the account.

1. Go back to the Entity Models list and click the **account** entity to edit it.
2. Click Add Field and choose 1-N Reference (grid).
3. Enter the following information:
  - Name: **Contacts**
  - Field ID: **contacts** - important to use pluralised name of entity you are using here
  - Select your Entity: **Contact**
  - Foreign Key Field: **account\_id**
4. Click **Create** and then save the entity using the **Save changes** button.

You have now created 2 simple entities that we will use in the next section: **Query Library**

A simple graphical representation of the entities you created will look like this:



## 6 Creating queries using the Query Library

Queries allow you to create instructions that send data to and from your app's underlying domain entities. To do this you will use connectors and some simple, human-readable JSON code.

### 6.1 Understanding connectors

Connectors allow your application to communicate with a variety of internal and external data sources. For example, the Stack9 API connector allows you to read and update information in Stack9's own SQL database. Stack9 comes with a set of preconfigured connectors already installed and configured ready to use.

We will use the Stack9 API connector as a means to connect the data you capture in your domain entities to a screen that your end-users will interact with.

### 6.2 Creating queries for simple CRUD operations

In this section you will create one query that will return a list of accounts and a separate one that will get the details of a specific account together with any contacts linked to it.

#### getAccountList

1. In the Console, choose Data Hub > **Query Library** and click **New Query**.
2. Change the title to **getAccountList**.
3. Choose **Stack9 API** from the Connector dropdown.
4. From the Request dropdown, choose **POST /account/search**.
5. For QueryParams, make sure the values are as follows:
  - page: **{{page}}**
  - limit: **{{limit}}**
  - method: leave empty
6. In the BodyParams field you will create a simple JSON object that specifies which fields from the Account entity to return, together with a condition to exclude any deleted records. Copy and paste the following code:

```
{
  "$select": [
    "company_name",
    "account_no",
    "status",
    "notes"
  ],
  "$where": { "_is_deleted" : false },
  "$withRelated": ["contacts"]
}
```

7. Click **Save**.
8. Test your query by clicking the **Run** button (or use Ctrl-Enter).

## getAccountDetails

Next, create another query named **getAccountDetails** that will return all the details of the account together with a list of contacts linked to it:

1. Use the **Stack9 API** connector again.
2. From the Request dropdown, choose **POST /account/search**.
3. QueryParams can be left empty.
4. Use the following for the BodyParams:

```
{
  "$select" : [
    "company_name",
    "account_no",
    "status",
    "notes"
  ],
  "$where": {
    "_is_deleted" : false,
    "id": {{id}}
  },
  "$withRelated": [
    "contact"
  ]
}
```

The *id: {{id}}* condition tells the query to return only a single account based on a variable that will be passed in.

5. Click **Save**.

## getContactList

Next, do the same for the Contact entity by creating a query called **getContactList**:

1. Again, use **Stack9 API** as the connector.
2. From the Request dropdown, choose **POST /contact/search**.
3. Use the same QueryParams as before.
4. In the BodyParams copy and paste the following JSON:

```
{
  "$select" : [
    "account_id",
    "account.company_name",
    "first_name",
    "last_name",
    "phone",
    "email"
  ],
  "$where": { "_is_deleted" : false},
  "$withRelated": []
}
```

The *account.company\_name* field will use the *account\_id* field in the Contact entity to read the Company Name from the associated account.

5. Click **Save**.

## getContactDetails

Repeat the process to create a **getContactDetails** query using the following BodyParams:

```
{
  "$select" : [
    "account_id",
    "account.company_name"
    "full_name",
    "phone",
    "email"
  ],
  "$where": {
    "_is_deleted" : false,
    "id":{{id}}
  }
}
```

```
},
"$withRelated": []
}
```

## getContactDetailsByAccount

There is one more query we need to create. This will allow us to link up the Contacts grid field we added to the Account entity that allows us to see all the contacts in an account.

Create another query called **getContactDetailsByAccount** using the **POST /contact/search** Stack9 API request with the following BodyParams:

```
{
  "$select": [
    "first_name",
    "last_name",
    "phone",
    "email"
  ],
  "$where": {
    "account_id": {{id}}
  }
}
```

Make sure to include the **{{page}}** and **{{limit}}** values in the queryParams as before and click **Save**.

You now have 5 queries ready to be used in the next section.

Advanced topic: add filters to your query - maybe this should be moved to a separate section/page?

## Searching and Filters

To help your users search for records, you can tag fields that will be included in the search, or add filters to your list queries. First we will add a few fields to be included when users enter a search term in the search bar on the list page.

1. Edit the **getAccountList** query.
2. In the right hand pane find the Query Search Fields section and click **Manage**.
3. Check the box next to *company\_name* and click **OK**.

Next we will add a filter that will allow you to list all accounts with one or more specific status values.

1. Edit the **getAccountList** query again.
2. In the right hand pane, look for the **Filters** section.
3. Click **Add Filter** and enter the following information:
  - Label: **Account Status**
  - Field ID: **status**
  - Query Filter Type: **Array**
  - Field: **status**
  - Use sub query: (leave unchecked)
  - Filter Render: **Select (multiple values)**
  - Data Source Type: **Query distinct**
  - Query Library: **Import from query libraries**
  - Query Name: **Status**
  - Query: **getAccountList**
  - Sequence: 1
  - Filter Section: (leave empty)
4. Make sure to **Save** the query.

You can add as many filters as you like. You can also group them into sections using the Filter Sections option.

## 7 Creating your app's User Interface

An application is of no use unless your users can interact with it through their web browser. This section covers how to create a series of simple **screens** that will allow users to:

- Add new accounts
- List those accounts in a table or grid
- Add a new contacts to an account

### 7.1 Create a Module from which to access your screens

First though, we will create a **module** into which these screens will be added.

1. In the Console, choose App Builder > **Modules** and click **Create New**.
2. Enter the following required info:
  - Name: **CRM**
  - App Key: **crm** (hint: use lowercase as a standard and separate words with an underscore)
  - Node Type: **appNode**
  - Leave the rest of the fields blank for now.
3. Click **Create**.

4. Next, click **Edit crm app** (or choose it from the module list if you have returned to the list) and you will see 3 tabs: Configuration, Navigation and Code View
5. Choose Navigation and hover over the CRM box and click the **+** button then choose **Link**.
6. Enter the following information:
  - Label: **Accounts**
  - Node Key: **accounts**
  - Link: **accounts/list**
7. Click **Update** to save your changes to the link.
8. Then choose **Save**.

## 7.2 Create an Accounts screen

Next we will create a **screen** to list, create and update the details of **Account** records.

1. In the Console, choose App Builder > **Screens** and click **Create New**.
2. Choose **Simple CRUD** as the screen type and enter the following required information.
  - Title: **Accounts**
  - Key: **accounts**
  - Route: **accounts**
  - App: **CRM**
  - Entity Key: choose **Account** from the dropdown
3. Next, in **Grid Configuration**, choose **getAccountList** from the list of queries you added in the previous section.
4. In the Columns section check the boxes next to the following fields:
  - *account.company\_name*
  - *account\_no*
  - *status*
  - Rich text fields cannot be listed in a Grid so we don't include notes here.
5. Click the gear icon next to *account.company\_name* and change the Label to **Company**. This is what will be displayed as the column header in your list view
6. Then, in the **Form Configuration** section, choose **getAccountDetails** as your query.
7. In the *When creating* section, check the box next to the following fields:
  - *company\_name*
  - *account\_number*
  - *status*
  - *notes*
8. Select the *Customize update fields* radio button. A set of fields will be displayed including *contacts*.
9. Check the box next to Contacts and then click the gear icon and enter the following information:
  - Grid Query: **getContactListByAccount**
  - Columns: check the box next to:
    - *id*
    - *full\_name*
    - *phone*
    - *email*

10. Click the gear icon next to *full\_name*.
11. Select the **Link** radio button and enter: `/crm/contacts/{id}`
12. Click **OK** to save the contacts field config.
13. Click **Save** to update the Contacts screen.

Once you have created your screen, you will need to link it to your module navigation

## 7.3 Create a Contacts screen

Repeat the process above to create another screen for Contacts using the **getContactList** and **getContactDetails** queries.

1. In the Console, choose App Builder > **Screens** and click **Create New**.
2. Choose **Simple CRUD** as the screen type and enter the following required information.
  - Title: **Contacts**
  - Key: **contact**
  - Route: **contact**
  - App: **CRM**
  - Entity Key: choose **Contact** from the dropdown
3. Next, in *Grid Configuration*, choose **getContactList** from the list of queries you added in the previous section.
4. In the Columns section use **Add new field** to add fields for:
  - Field: **Account** / Key: **account.company\_name**
  - Field: **First Name** / Key: **first\_name**
  - Field: **Last Name** / Key: **last\_name**
  - Field: **Phone** / Key: **phone**
  - Field: **Email** / Key: **email**
5. Then, in the *Form Configuration* section, choose **getContactDetails** from the list of queries you added in the previous section.
6. Select the *Use create...* radio button and check the box next to the following fields:
  - *full\_name*
  - *first\_name*
  - *last\_name*
  - *phone*
  - *email*
7. Click **Save**.

## 7.4 Test your Application

Need a demo site to confirm the steps below

You can now test all the previous steps by running the web front-end for your application. In the test you will do the following:



- Create a number of new Account records
- Search for an Account
- Filter the account list by Account Status
- Add a new Contact to one of the Accounts
- Delete an Account

1. Point your browser at the front-end web application at **xxxx-ui.app.stack9.dev**
2. Click the **CRM** app tile and click the **Accounts** menu item.
3. Click **Create** and a form will be displayed.
4. Enter the following details:
  - Company Name: **Test Company A**
  - Account No.: **1234**
  - Status: **Active**
  - Notes: **any text you like...**
5. Click **Save**.
6. Repeat the process to create as many account records you want.

Use the search bar in the left hand panel to enter all or part of the company name of one of your accounts to see the list filtered.

Use the Company Status filter to filter you accounts list by one or more status values, e.g. Active accounts only. You can also save this or more complex filters by using the Save Filter button at the top of the list. Saved filters can then be accessed in the future from the Saved Filters menu.

To change an existing record, click the Company Name link of an Account to bring up the edit form and make some changes before clicking Save to update and return to the list.

To create a contact there are 2 ways to do this: either by clicking Contacts in the CRM module in the same way as you created an Account, or, given contacts can belong to an an account, by editing an Account and adding a Contact to it via its own Contacts list. We'll do the latter here:

1. Open one of the Account records to edit it.
2. At the bottom of the form you will see an empty Contacts list. Click the **Create new** link.
3. Enter the following details:
  - First Name: **Joe**
  - Last Name: **Bloggs**
  - Phone: **0455 555 555**
  - Email: **test@example.com**
4. Note the Full Name field is empty and read-only. We'll deal with that in the next section: **Automation**.

You can also delete records by opening a record to edit it and then clicking the delete button. Note this only flags the record for deletion so the record can still be recovered if necessary. This will be dealt with in a **separate section**.

# 8 Defining Logic using Automation

Introduction to the different types of automation...

## 8.1 Update a field automatically

You may want a separate field in your contact records that holds a Full Name value made up from the contact's First Name and their Last Name. You could get the user to enter it manually when they create or update the record but that would be inefficient. Instead we can get the system to automatically update the Full Name field.

First we will create a query that will update the relevant record's full\_name field with a concatenated value made of first\_name and last\_name.

1. In the Console, navigate to Data Hub > **Query Library** and click **Create New**.
2. Call it **updateContactFullName**
3. Choose **Stack9 Database** as the connector
4. Enter the following query:

```
update contact
set full_name = first_name || ' ' || last_name
where id = {{contact_id}}
```

5. Click **Save**.

Next we will use the query in an automation that will be run when a new contact record is created.

1. In the Console, navigate to App Builder > **Automations** and click **Create New**.
2. Choose **When record updated** as the automation type.
3. Enter the following details:
  - Name of the automation: **Update Contact Full Name**
  - Field ID: **update\_contact\_full\_name**
  - App: **CRM**
  - Entity Key: **Contact**
4. Click the **Create** button.
5. Open the automation to configure it.
6. In the Actions section click **Add Action**.
7. Choose **Run Query**.
8. Enter the following details:
  - Name: **Update Full Name**
  - Key: **update\_full\_name**
  - Query Library Key: **updateContactFullName** (the query you created in the first step).
  - ID: **{{trigger.entity.id}}**
9. Click **Save Changes** to update the automation.

10. Test out the automation by creating a new contact and then opening it up again to see the Full Name field automatically updated.

## Next steps...

Extended topics including:

- Workflow
- 

## Workflow

Whilst the Status field in your account entity may be a good way to see the *current* state of the account, you may want a bit more control over how that status is set, including more detail and history of when it changed. For example, you could ensure account managers satisfy certain conditions before an account is marked as Active - like signing contracts; or when archiving an account.

In this guide we will cover the following:

- Add a simple workflow to the Account entity you created earlier with the following steps:
  - **Setup** - information about the account is gathered
  - **Contract** - contracts are negotiated
  - **Live** - account is live and one or more contracts are in progress
  - **Archived / Lost**
- Add some conditions to the Setup step that will prevent it from being moved to the Contract Negotiation step if they are not met

1. In the Console, navigate to Data Hub > **Domain Entities** and open the Account entity to edit it.
2. Switch to the ~~Workflow tab~~ or Code View tab
3. Copy and paste the following code at the end of the line that reads **"hooks": []**

```
,
  "workflowDefinition": {
    "steps": [
      {
        "key": "setup",
        "name": "Setup",
        "description": "",
        "color": "orange",
        "order": 1,
        "position": {
```

```
    "x": 100,  
    "y": 100  
  },  
  "ownershipDetails": {  
    "userGroups": [],  
    "userFields": []  
  }  
},  
{  
  "key": "contract",  
  "name": "Contract",  
  "description": "",  
  "color": "blue",  
  "order": 2,  
  "position": {  
    "x": 300,  
    "y": 0  
  },  
  "ownershipDetails": {  
    "userGroups": [],  
    "userFields": []  
  }  
},  
{  
  "key": "live",  
  "name": "Live",  
  "description": "",  
  "color": "blue",  
  "order": 3,  
  "position": {  
    "x": 500,  
    "y": 80  
  },  
  "ownershipDetails": {  
    "userGroups": [],  
    "userFields": []  
  }  
}  
],
```

```
"outcome": {
  "success": {
    "label": "Archive",
    "position": {
      "x": 1100,
      "y": 40
    },
  },
  "ownershipDetails": {
    "userGroups": [],
    "userFields": []
  }
},
"failure": {
  "label": "Lost",
  "position": {
    "x": 300,
    "y": 200
  },
  "ownershipDetails": {
    "userGroups": [],
    "userFields": []
  }
},
"actions": [
  {
    "key": "setup_to_contract",
    "name": "Ready for contract",
    "default": true,
    "from": {
      "step": [
        "setup"
      ]
    },
    "to": {
      "step": "contract"
    }
  },
  {
```

```
"key": "contract_to_live",
"name": "Mark as Live",
"default": true,
"from": {
  "step": [
    "contract"
  ]
},
"to": {
  "step": "live"
}
},
{
  "key": "live_to_contract",
  "name": "Back to Contract",
  "default": false,
  "from": {
    "step": [
      "live"
    ]
  },
  "to": {
    "step": "contract"
  }
},
{
  "key": "expired",
  "name": "All audit reports completed",
  "default": true,
  "from": {
    "step": [
      "pending_audit"
    ]
  },
  "to": {
    "outcome": 1
  }
},
{
```

```
"key": "reject",
"name": "Account closed, archive",
"default": false,
"from": {
  "step": [
    "setup",
    "contract",
    "live"
  ]
},
"to": {
  "outcome": 1
}
},
{
  "key": "reactivate",
  "name": "Unarchive",
  "default": false,
  "from": {
    "outcome": [
      1
    ]
  },
  "to": {
    "step": "setup"
  }
},
{
  "key": "lost",
  "name": "Account lost, close",
  "default": false,
  "from": {
    "step": [
      "setup",
      "contract",
      "live"
    ]
  },
  "to": {
```

```
        "outcome": 2
      }
    }
  ],
  "conditions": [
    {
      "key": "contract_signed",
      "name": "Contract signed",
      "description": "",
      "action": "contract_to_live",
      "required": true
    }
  ]
}
```

4. Click **Save Changes**.
5. Switch to the **Workflow** tab and you will see a visual representation of your workflow.

Let's test the workflow out next.

1. Go back to the front-end application and open the Accounts screen.
2. Click Create new and enter the following details:
  - Company Name: **Test Company B**
  - Account No.: **3456**
  - Status: **(leave empty for now)**
3. Click **Save**.
4. Open the Account and note the new Setup option at the top of the record.
5. Click the arrow on this option and choose Ready for Contract.
6. The records is updated and the label.
7. Now try open the account and move the workflow to Live.
8. From the Action menu, choose Mark as Live.
9. An error is displayed: "" This is because we haven't marked one of the conditions of the move as true.
10. In the record edit view, click the tasks

## Adding Automation to Workflow

When we created the account we didn't set a value for Status. Here we will set the status to Active when the workflow changes to Live. In this section we'll do the following:

- Create a query that updates the account entity's status field to "Active"
- 

## Custom Screens



Say we wanted to add a new Contract entity to the model and link that to the Account in the same way as we did for Contacts. The simple approach would be to add the new entity as before, together with an `account_id` Single Drop Down field to link it to the account. Then we would add a new Contracts grid field to the Account details view and

---

Revision #15

Created 19 June 2023 00:53:14 by Thiago Passos

Updated 23 June 2023 03:04:30 by Richard Schedler