Welcome to Stack9 Core developer documentation!

Overview

Stack9 is a battle-tested NodeJs / ReactJs Docker application used as the base component for rapid-development of enterprise web applications.

At it's core, it allows software developers to specify entity definitions which serve as building blocks for dynamic generated data management screens, generation of RESTFul Data API endpoints and dynamic provisioning of relational database schemas to store application data.

Further to this, Stack9 is designed to work with Azure / AWS services natively to avoid monolithic software architectures and leverage cloud managed services that scale such as containerization (serverless), object storage (AWS S3, Azure Blog), Azure AD IdP and the orchestration of distributed systems, making use of message queue architecture (AWS SQS, Azure storage queue) and long running processing services (AWS Batch, Azure Durable Functions).

Since the Stack9 platform is a data management tool designed to fit various use cases, configuration and customization of a Stack9-powered application has no boundaries. Any software developer with NodeJs / ReactJs skills is able to create applications easily with the full suite of available components/libraries from the JavaScript community.

Stack9 Team Support

Stack9 Core team gives to enterprises the support necessary to implement and develop in the best way their system.

Message Queue Services (Stack9-core v3.0)

In Stack9 version 2.0.0 it was introduced the ability to interact with message queue services (AWS SQS & Azure Queue Storage)

AWS SQS

Azure Queue Storage

Environment variables

These enviroment variables should be configured in order to start using message queue service in stack9

MESSAGE_QUEUE_POLLING_WAIT_TIME_MS // defaults 10000ms (10s)

MESSAGE_QUEUE_CREATE_IF_NOT_EXISTS // true or false

MESSAGE_QUEUE_SERVICE // 'aws-sqs' or 'azure-queue-storage'

MESSAGE_QUEUE_QUEUE_URL // http://localstack:4566/000000000000/stack9-core-sqs or http://azurite:10001devstoreaccount1/stack9-core-sqs

WORKFLOW_NOTIFICATION_SEND_GRID_DEFAULT_TEMPLATE_ID // d-123528a80a1022aab9a5d91afaf08b9a

Requires Authorization

AWS - AWS_REGION - AWS_ACCESS_KEY_ID - AWS_SECRET_ACCESS_KEY

Azure

- AZURE_QUEUE_STORAGE_CONNECTION_STRING // DefaultEndpointsProtocol=<http|https>;AccountName=<account-name>;AccountKey=<account-key>;BlobEndpoint=<blob-endpoint>;

Stack9-config

In stack9.config.json the message queue service should be specified in order to be enabled.

"MessageQueueService": "aws-sqs", "MessageQueueServiceOptions": ["aws-sqs", "azure-queue-storage"]

When running locally, a docker container with localstack will be started to mock SQS service when chosen aws-sqs and azurite for azure-queue-storage

File Adapters

Security Model

You use the security model in Stack9 to protect the data integrity and privacy in a organization. The security model also promotes efficient data access and collaboration. The goals of the model are as follows:

Grant users access that allows only the levels of information required to do their jobs.

Categorize users and teams by security role and restrict access based on those roles.

Prevent access to objects a user does not own or share.

You combine business units, role-based security, and record-based security to define the overall access to information that users have in your organization.

Security Roles

Security roles in Stack9 are a matrix of privileges and access levels for the various entities.

Privileges

Privileges are the basic security units that delineate what action a user can perform in the Stack9 system. These cannot be added or deleted but only modified. The common privileges in Stack9 for each entity are as follows:

Create — Allows the user to add a new record

Read — Allows the user to view a record

Write — Allows the user to edit a record

Delete — Allows the user to delete a record

Comment - Allow the user to write a comment to the record

Attach - Allows the user to attach files to a record

Export - Allows the user to export list of records from a grid

Workflow Step — Allows the user to edit a record when on a specific workflow step (not implemented)

Entity Definitions

Overview

The fields of the entity definition are: key, name, pluralisedName, isActive, allowComments and allowTasks.

Parameters

See below an example:

```
{
 "head": {
  "key": "test_class",
  "name": "Test Class",
  "pluralisedName": "Test Class",
  "isActive": true,
  "allowComments": true,
  "allowTasks": true
 }
}
```

Field Types

Overview

The field types are types of fields which we can define on json definition:

SingleDropDown

Checkbox

DateField

FileField

Grid

MapLocation

MultiDropDown

NumericField

MonacoEditorField

HtmlField

OptionSet

CustomUIComponent

PrivilegiesMatrix

ColorPickerField

In three of them should be specified RelationshipFieldTypes, read more about here.

Usage

See below examples of how to use the fields and properties:

SingleDropDown

```
{
   "key": "test_id",
   "label": "Test",
   "placeholder": "Select value...",
   "description": "SingleDropdown required",
   "type": "SingleDropDown",
   "typeOptions": {
     "label": "name"
   },
   "validateRules": {
     "required": false
   },
   "relationshipOptions": {
     "ref": "test_location",
     "many": false
   }
```

Checkbox

```
{
  "key": "is_tested",
  "label": "Tested",
  "type": "Checkbox",
  "validateRules": {
    "required": true
  }
}
```

DateField

The typeOptions allows to include the time. In the typeOptions can be configure to includes the time.

```
{
  "key": "date_field",
  "label": "Date field",
  "placeholder": "Select Date...",
  "description": "Date field required without time",
  "type": "DateField",
  "validateRules": {
    "required": true
  },
  "typeOptions": {
    "time": false
  }
}
```

FileField

In the typeOptions can be configure which file format are accepted. isPublic property specify if the file is in the private or public mode.

```
{
  "key": "file_field",
  "label": "File field",
  "type": "FileField",
  "description": "File field with all formats required",
  "typeOptions": {
    "accept": "image/jpeg, image/png, image/gif, application/pdf, application/msword, application/vnd.openxmlformats-officedocument.wordprocessingml.document, application/vnd.ms-excel, application/vnd.openxmlformats-officedocument.spreadsheetml.sheet",
    "isPublic": true
  },
  "validateRules": {
    "required": true
  }
}
```

Grid

The allowCreate specify if it will be allow to create a record directly from the grid. The gridSettings is used to define which columns should be displayed in the grid and the relationshipField should be the key of the related entity that was specified in the relationshipOptions.

```
{
  "key": "assessments",
  "label": "Assessments",
  "description": "Student assessments",
  "type": "Grid",
  "typeOptions": {
    "relationshipField": "test_student_id",
    "allowCreate": true,
    "gridSettings": {
      "agGrid": {
        "columnDefs": [
          {
            "headerName": "Assessment",
            "field": "name",
            "sortable": true
        }
      ]
    }
  }
 },
 "validateRules": {
  "required": false
 },
 "relationshipOptions": {
  "ref": "test_student_assessment",
  "many": false
 }
}
```

MapLocation

```
{
 "key": "gps_location",
 "label": "GPS Location",
 "type": "MapLocation",
 "validateRules": {
  "required": false,
  "pattern": "^(-?\\d+(\\.\\d+)?),\\w*(-?\\d+(\\.\\d+)?)$"
 }
}
```

MultiDropDown

The value is required in the property typeOptions.

```
{
 "key": "multidropdown_field_id",
 "label": "Multidropdown",
 "placeholder": "Select value...",
 "description": "Multidropdwon required",
 "type": "MultiDropDown",
 "typeOptions": {
  "value": "id",
  "label": "name"
 },
 "validateRules": {
  "required": false
 },
 "relationshipOptions": {
  "ref": "test_student",
  "many": true
 }
}
```

NumericField

```
{
  "key": "numeric_field",
  "label": "Numeric Field",
  "type": "NumericField",
  "description": "Numeric TextField required",
  "validateRules": {
   "required": false,
   "min": 0,
   "max": 10
  },
  "typeOptions": {
   "precision": 2
  }
}
```

MonacoEditorField

The typeOptions language is required and accepts: json, handlebars, text and css.

```
{
  "key": "json_field",
  "label": "Json field",
  "type": "MonacoEditorField",
  "description": "Json editor field required",
  "typeOptions": {
    "language": "json"
  },
  "validateRules": {
    "required": true
  }
}
```

HtmlField

```
{
  "key": "html_field",
  "label": "Html field",
  "type": "HTMLField",
  "description": "Html field required",
  "validateRules": {
    "required": true
  }
}
```

OptionSet

The property values is required and must be configured with the possible values.

```
{
  "key": "option_set",
  "label": "Option Set",
  "type": "OptionSet",
  "typeOptions": {
    "values": ["value1", "value2", "value3", "value4", "value5"]
  },
  "validateRules": {
    "required": true
  }
}
```

CustomUIComponent

The component property is required and must be a valid name of the file inside the folder client/UIComponent .

```
{
  "key": "test",
  "label": "Counter",
  "type": "CustomUIComponent",
  "typeOptions": {
    "component": "Counter"
  },
  "validateRules": {
    "required": false
  }
}
```

PrivilegiesMatrix

The possible scope values are app and entity.

```
{
  "key": "app_privilegies",
  "label": "App Privilegies",
  "type": "PrivilegiesMatrix",
  "validateRules": {
    "required": false
  },
  "typeOptions": {
    "scope": "app"
  }
}
```

ColorPickerField

```
{
  "key": "color_picker_field",
  "label": "Color Picker Field",
  "type": "ColorPickerField",
  "validateRules": {
    "required": true
  }
}
```

Configuration

validateRules

required: indicates if the field is required

maxLength: set the max length allowed for the field

minLength: set the min length allowed for the field

max: set the max value allowed for the field

min: set the min value allowed for the field

pattern: can be configured a Regex expression to be validated

behaviourOptions

readOnly: read only behaviour option

hidden: hidden behavior option

displayAsDefaultFilter: default display filter behaviour option

Entity Relationships

Overview

The entity relationship are applied to these three Fields: SingleDropDown, MultiDropDown and Grid.

Parameters

See below the relationships types:

SingleDropDown - Many to One

MultiDropDown - Many to Many

Grid - One to Many

Entity Views

Stack9 lets you show your entity entries in different ways.

View Types

listView: Show entries as rows in a table. It is used as default if any view is specified.

mapView: Show entries as pins in a map. A MapLocation field type is required.

kanbanBoardView: Show entries as cards in a Kanban board.

timelineView: Show entries as events in a calendar.

Example

```
{
  "views": [
    {
      "name": "List View",
      "viewType": "listView",
      "isDefault": true
    },
    {
      "name": "Map View",
      "viewType": "mapView",
      "viewOptions": {
        "locationField": "point_field",
        "title": ["text_field"],
        "description": [
          { "label": "field", "value": "text_field" },
          { "label": "field", "value": "text_field" }
        ]
      }
    },
    {
      "name": "Board View",
      "viewType": "kanbanBoardView",
      "viewOptions": {
        "title": ["text_field"],
        "description": ["text_field", "text_field"],
        "groupBy": { "sourceType": "workflow" }
      }
    },
    {
      "name": "Calendar",
      "viewType": "timelineView",
      "viewOptions": {
        "title": ["text_field"],
        "groupBy": {
          "sourceType": "field",
          "sourceField": "text_field"
        },
        "dateFields": {
          "start": "date_field",
          "end": "date_field"
        }
      }
    }
  ]
}
```

App Structures

JSON properties available:

name

themeColor

dateFormat

timeFormat

timezone

loginBackgroundImage

companyLogo

favicon

instanceIcon

apps

cronJobs

mqHandlers

allowedAccountsForNonProdEmailSending

Example

```
{
 "name": "Project Name",
 "themeColor": "#243746",
 "dateFormat": "dd/MM/yyyy",
 "timeFormat": "h:mm a",
 "timezone": "Australia/Sydney",
 "loginBackgroundImage": "https://...backgroundImage.png",
 "companyLogo": "https://...companyLogo.png",
 "favicon": "https://...favicon.ico",
 "instanceIcon": "https://...instanceIcon.png",
 "apps": [
  {
   "key": "shipping",
   "icon": "fas fa-ship",
   "name": "Shipping",
   "dashboard": {
    "name": "Shipping"
```

```
},
"menu": [
 {
   "key": "shipments",
   "label": "Shipments",
   "entityKey": "shipment"
 },
 {
   "key": "shipping_setup",
   "label": "Shipping Setup",
   "item_groups": [
    {
      "key": "shipping_setup",
      "items": [
       {
         "key": "address",
         "entityKey": "address",
         "label": "Addresses"
       },
       {
         "key": "address_types",
         "entityKey": "address_type",
         "label": "Address Types"
       }
      ]
    }
   ]
 },
 {
   "key": "shipping_organisations",
   "label": "Shipping Organisations",
   "item_groups": [
    {
      "key": "shipping_organisation",
              "items": [
                {
                  "key": "organisations",
                  "entityKey": "organisation",
                  "label": "Organisations"
                }
              ]
            }
          ]
        }
      ]
    }
  ],
  "cronJobs": [
    { "command": "example-cron-job", "cronTime": "*/20 * * * * *" },
    { "command": "example-cron-job", "cronTime": "*/30 * * * * *" }
  ],
  "mqHandlers": [
    { "command": "example-custom-handler1" },
    { "command": "example-custom-handler2" },
  ],
  "allowedAccountsForNonProdEmailSending": [
    "april9.com.au",
    "user.test@april9.com.au",
  ]
}
```

Printable Documents

The solution is designed to store “printable document templates” which are associated with record types (e.g.: “sales transaction”). Once these templates are used with a given record (e.g.: “Sales Transaction #12345”), they are compiled/merged into a document that can be downloaded as a PDF file, be printed from the browser window or used as attachments in transactional emails.

In Stack9 core release 2.1.0, printable documents can be sent in the email attachments as part of the notification system.

Task Management (Stack9-core v3.0)

Overview

The Task Management feature was introduced on v2.5 and following CRUD best practices Stack9 was capable of deliver the ability to create, assignee, update, mark and remove tasks.

Parameters

Searching for a better understanding, below are the endpoints for each request:

Get task by entityID:

GET /:entityKey/:entityId/task

Returns:

```
[
 {
   "id": 2,
   "entity_id": 1,
   "key": "test_notification",
   "description": "gfcyfytg",
   "is_completed": false,
   "due_date": null,
   "assigne_id": null,
   "type": "native",
   "payload": null,
   "_is_deleted": false,
   "_created_by": 1,
   "_updated_by": 1,
   "_created_at": "2021-11-30T00:50:54.260Z",
   "_updated_at": "2021-11-30T00:50:54.260Z",
    "updated_by": {
      "id": 1,
      "_is_deleted": false,
      "_created_by": null,
      "_updated_by": null,
      "_created_at": "2021-11-29T05:29:19.162Z",
      "_updated_at": "2021-11-29T05:29:19.162Z",
      "identity_provider_user_id": "8181878171ada",
      "first_name": "test",
      "last_name": "test",
      "full_name": "test test",
      "username": "test@test.com.au",
      "email": "test@test.com.au",
      "phone": null,
      "job_title": null,
      "is_contact": false,
      "security_role_id": 1,
      "is_active": false,
      "is_blocked": false,
      "blocked_at": null
    }
  }
]
```

Create new task to the entity:

POST /:entityKey/:entityId/task

Payload example:

```
{
  "description": "string",
  "is_completed": false,
  "due_date": "2021-10-09 00:57:06",
  "assigne_id": 1,
  "type": "string",
  "payload": "string"
}
```

Returns:

```
{
  "id": 8
}
```

Task Related

Exploring the task management engine on Stack9 v3.0 it was added a task related entity, which means that the task can be linked to one or more entities.

Payload with related_entities:

```
{
  "description": "string",
  "is_completed": false,
  "due_date": "2021-10-09 00:57:06",
  "assigne_id": 1,
  "type": "string",
  "payload": "string",
  "related_entities": [
    {
      "key": "test",
      "entity_id": 1
    }
  ]
}
```

Note: Task related is an optional array of entities.

Update/Mark a task:

```
PUT /:entityKey/:entityId/:taskId
```

Payload example:

```
{
  "description": "updated description",
  "is_completed": true,
}
```

Remove a task from the entity:

```
DELETE /:entityKey/:entityId/:taskId/
```

Note: The PUT and DELETE requests returns a status code 204.

Filters

Filters available

By default, Stack9 makes available all entity fields as filters.

Pinned filters

In the fields definition, set displayAsDefaultFilter as true to pin the field as filter when the page is loaded.

```
{
  "fields": [
    {
      "key": "name",
      "label": "Name",
      "type": "TextField",
      "behaviourOptions": {
        "displayAsDefaultFilter": true
      }
    }
  ]
}
```

Additional filters (Stack9-core v3.0)

When required to add fields from related entities, use addionalFilters configuration to make them available.

```
{
  "additionalFilters": [
    "user.email",
    "user.security_role.name",
    "user.is_blocked",
    "single_self_relationship_field.option_set",
  ]
}
```

note: the additional filter will inherit the properties from the related entity, if the field is set with displayAsDefaultFilter in the parent entity it will be displayed as default in the entity as well.

Notifications (Stack9-core v3.0)

Overview

In Stack9 Core v2.1, it was introduced the ability to set up notifications with attachments for operations (create, takeOwnership) and/or workflow actions. Each notification can be configured with user fields and/or user groups to be notified when the specific operation/action is executed.

In order to start using notifications, a hook type message must be added to the entity definition using a native Stack9 queue called _sendNotificationHandle. See more native handlers

```
{
  "hooks": [
    {
      "type": "message",
      "hookType": "AfterChange",
      "functionName": "_sendNotificationHandler"
    }
  ]
}
```

Definition

operation: Required (create or takeOwnership)

action: Required only if operation is proceedToStep. It accepts (proceedToStep || approveWorkflow || rejectWorkflow)

emailTemplateId: Optional Sendgrid template Id. If not specified, default value will be the value configured for the environment variable WORKFLOW_NOTIFICATION_SEND_GRID_DEFAULT_TEMPLATE_ID

recipient: Required object with the user fields and/or user group codes to be notified.

attachments: Optional object with printable document codes and/or custom document functions to generate documents to be sent as attachments.

```
"notifications": [
  {
    "operation": "create",
    "emailTemplateId": "d-1234567389fe4a84be8e41cd8ead333c",
    "recipient": {
      "userGroups": [],
      "userFields": ["user_id"]
    },
    "attachments": {
```

```
    "printableDocuments": ["user_report"],
    "documentFunctions": ["generate-excel-file"]
  }
},
{
  "operation": "proceedToStep",
  "action": "newtoreview",
  "emailTemplateId": "d-1234567389fe4a84be8e41cd8ead333c",
  "recipient": {
    "userGroups": [],
    "userFields": ["reviewer_id"]
  },
  "attachments": {
    "printableDocuments": ["user_report", "order_list"],
    "documentFunctions": []
  },
},
{
  "operation": "takeOwnership",
  "emailTemplateId": "d-1234567389fe4a84be8e41cd8ead333c",
  "recipient": {
    "userGroups": [],
    "userFields": ["user_id"]
  },
  "attachments": {
    "printableDocuments": [],
    "documentFunctions": []
  }
},
{
  "operation": "proceedToStep",
  "action": "approve",
  "emailTemplateId": "d-1234567389fe4a84be8e41cd8ead333c",
  "recipient": {
   "userGroups": [],
   "userFields": ["user_id"]
  },
  "attachments": {
   "printableDocuments": ["test_1"],
   "documentFunctions": ["native/example-document-function"]
  }
 },
 {
  "operation": "proceedToStep",
  "action": "reject",
  "emailTemplateId": "d-1234567389fe4a84be8e41cd8ead333c",
  "recipient": {
   "userGroups": [],
   "userFields": ["user_id"]
  },
  "attachments": {
   "printableDocuments": [
    "test_1",
    "test_2",
    "test_3"
   ],
   "documentFunctions": ["native/example-document-function"]
  }
 }
]
```

Recipients

Recipients are the users who will receive the notification. Can be specified any field from the form that has relationship with user entity or user group codes.

userFields: Array of user fields, must be a field that has relationship with user entity.

userGroups: Array of user group codes, if specified, all users from the group will be notified.

Attachments

Printable Documents

In Stack9 core v2.1 it was added a new column code to the printable_document entity. To generate a PDF file from a printable document and send as an attachment in the email notification, one or more codes can be specified in the attachments.printableDocuments property.

```
{
 "attachments": {
  "printableDocuments": ["user_report", "activities"]
 }
}
```

Customise attachments using document functions

Custom functions can be added to the attachments.documentFunctions to generate a custom document with a different extension and sent as attachment in the email notification.

Document functions must be placed in the stack9/document-functions directory and should return an array of documents to be attached to the email notification.

```
class ExampleDocumentFunction {
 constructor(context) {
  this.services = context.services;
 }

 async exec() {
  const file = await this.services.entity.export('test_notification', {
   $where: {
    id: this.entity.id,
   },
  });

  return {
   documents: [
    {
     filename: `Test`,
     mimetype: `application/vnd.openxmlformats-officedocument.spreadsheetml.sheet`,
     data: file, // buffer[]
    },
   ],
  };
 }
}


module.exports = {
  exec: args => new ExampleDocumentFunction(args).exec(),
  timeout: 5000,
};
```

User notification storage

Now in Stack9 Core v3.0, it was added a new feature that allows users notifications records to be stored by the notification engine through API endpoint allowing records management.

The new feature structure looks like this on MongoDB:

notificationnotification1

Note: The content field is an object which has it owns parameters.

Parameters

The notifications can be filter by status, sorted by field and/or order, paginated by limit and/or skip.

```
  GET /api/notifications?status=seen&sortField=_id&sortOrder=desc&limit=25&skip=5
```

Returns:

```
[
  {
    "_id": "61a5a71bbbc2b722ccc5b9e2",
    "user_id": 1,
    "entity_key": "test_notification",
    "entity_id": 2,
    "content": {
      "icon": "http://",
      "title": "Ella Ferreira has created the record123",
      "message": "Testing notification handlebars Ella Ferreira",
      "due_date": "6 minutes ago"
    },
    "timestamp": "2021-11-30T04:22:51.495Z",
    "seen": true
  }
]
```

Note: Default value for status is all and sort by order is desc.

Stack9 also has the count feature which shows how many notifications the user has.

GET /api/notifications/count?status=all

Returns:

```
{
  "count": 2
}
```

Note: Default value for count is all.

Users are able to manage their notifications easily.

PUT /api/notifications/mark;

See below examples of how the notifications are set as seen or unseen.

```
{
  "ids": ["61777e720e81d5980f417ca6", "6171f884aa767a5347dd6b34"],
  "status": "seen",
  "after": "2021-10-09 00:57:06"
}
{
  "ids": ["61777e720e81d5980f417ca6", "6171f884aa767a5347dd6b34"],
  "status": "unseen",
  "after": "2021-10-09 00:57:06"
}
```

Note: The after field is optional.

Workflow Actions

Actions are connections between steps and/or workflow outcomes.

At least two actions are required. One that finalised the

from: array of steps where the action can initiate from. use ALL if actions can be initiated from all steps.

to: use { "step": "KEY" } to connect to another step. Use { "outcome": 1 } to finalise the workflow. Possible values are 1 for success and 2 for failure.

preventUserInteraction: set to true if action should NOT be available in the UI for user selection. Used for when action should be trigger automatically through business logic (from workflow functions or cron jobs).

Example

```
{
 "actions": [
  {
   "key": "approve",
   "name": "Approve Record",
   "from": ["ALL"],
   "to": { "outcome": 1 },
   "preventUserInteraction": true
  },
  {
   "key": "reject",
   "name": "Reject record",
   "from": ["ALL"],
   "to": { "outcome": 2 }
  }
 ]
}
```

Workflow Steps

Entity workflows can have multiple steps.

key: string used to define the step key

name: string used to define the step label in the UI

position

x: x-axis

y: y-axis

ownershipDetails: specify who is the owner of the step. It will be used to send notifications

userGroups: array of codes from the native user_group entity

userFields: array of related fields with the native user and/or user_group entity

Example

```
{
 "steps": [
  {
   "key": "initial",
   "name": "Initial",
   "order": 1,
   "position": {
    "x": 50,
    "y": 0
   },
   "ownershipDetails": {
    "userGroups": [],
    "userFields": ["user_id", "user_group_id"]
   }
  },
  {
   "key": "final",
   "name": "final",
   "order": 2,
   "position": {
    "x": 100,
    "y": 0
   },
   "ownershipDetails": {
    "userGroups": [],
    "userFields": ["user_id", "user_group_id"]
   }
  }
 ]
}
```

Workflow Steps

Entity workflows can have multiple steps.

key: string used to define the step key

name: string used to define the step label in the UI

position

x: x-axis

y: y-axis

ownershipDetails: specify who is the owner of the step. It will be used to send notifications

userGroups: array of codes from the native user_group entity

userFields: array of related fields with the native user and/or user_group entity

Example

```
{
 "steps": [
  {
   "key": "initial",
   "name": "Initial",
   "order": 1,
   "position": {
    "x": 50,
    "y": 0
   },
   "ownershipDetails": {
    "userGroups": [],
    "userFields": ["user_id", "user_group_id"]
   }
  },
  {
   "key": "final",
   "name": "final",
   "order": 2,
   "position": {
    "x": 100,
    "y": 0
    },
    "ownershipDetails": {
      "userGroups": [],
      "userFields": ["user_id", "user_group_id"]
    }
  }
 ]
}
```

Workflow Outcomes

Entity workflows have two outcomes: success or failure

label: string used to define the outcome label in the UI

position

x: x-axis

y: y-axis

Example

```
{
 "outcome": {
  "success": {
    "label": "Completed",
    "position": {
      "x": 750,
      "y": -30
    }
  },
  "failure": {
    "label": "Cancelled",
    "position": {
      "x": 750,
      "y": 235
    }
  }
 }
```

}

DATA-API Overview

The DATA-API is organized based on REST service. Our API has predictable resource-oriented URLs, returns JSON-encoded responses, and uses standard HTTP response codes, authentication, and verbs.

The DATA-API allows the user to execute the following operations

Verb	Operation	Endpoint
POST	Create Record	/api/:entity_name
PUT	Edit Record	/api/:entity_name/:id
DELETE	Delete Record	/api/:entity_name/:id
POST	Search Record	/api/:entity_name/search
POST	Export Record	/api/:entity_name/export

Authentication

The Stack9 Data-API uses API keys to authenticate requests. You can view and manage your API keys in the YOUR_DOMAIN/apps/administration/api_key Stack9 Back office Dashboard.

Your API keys have many privileges, so be sure to keep them safe. Do not share your secret API keys in publicly accessible areas such as GitHub repository, Docker-Hub, client-side code, and so on.

Use your API key by adding it on the Api-Key header in each request.

```
// NodeJS example
fetch('https://april9.stack9.co/api/customer/search', {
 method: 'POST',
 headers: {
  'Api-Key': '<my_secret_api_key>',
  'Content-Type': 'application/json',
 },
})
 .then(response => response.json())
 .then(data => console.log(data));

// OR
```

```
const response = await fetch('https://april9.stack9.co/api/user', {
  headers: {
    'Api-Key': '<my_secret_api_key>',
    'Content-Type': 'application/json',
  },
});
const data = await response.json();
console.log(data);

// OUTPUT
// [
//   {
//     id: 1,
//     email: 'my_email@domain.com.au'
//   },
//   {
//     id: 2,
//     email: 'other_email@domain.com.au'
//   },
// ];
```

All API requests must be made over HTTPS. Calls made over plain HTTP will fail. API requests without authentication will also fail.

Status Code	Operation	Response
200	Queried successfully	
400	Bad Request Error	{ error: {}}

```
},
```

Create Record

One of the operation of DATA-API is to create an entity.

In order to create a specific entity, a POST request to /api/:entity_name with json format payload enclosed in the body is sent.

The payload format will vary according to your Entity Definition which will define which format should be sent to each property and which fields are required.

```
// e.g.: customer.json [Entity Definition]
{
 // ...,
 "fields": [
  {
   "key": "email",
   "label": "email",
   "type": "TextField",
   "validateRules": {
    "required": false,
    "maxLength": 5
   }
  },
  {
   "key": "full_name",
   "label": "Full Name",
   "type": "TextField",
   "validateRules": {
    "required": false
   }
  },
  {
   "key": "dob",
   "label": "Date of Birth",
   "type": "DateField",
      "validateRules": {
        "required": false
      },
      "typeOptions": {
        "time": false
      }
    },
    {
      "key": "country_id",
      "label": "Country",
      "type": "SingleDropDown",
      "relationshipOptions": {
        "many": false,
        "ref": "country"
      },
      "typeOptions": {
        "label": "name"
      },
      "validateRules": {
        "required": false
      }
    },
    {
      "key": "verified",
      "label": "Vefified",
      "type": "Checkbox",
      "validateRules": {
        "required": false
      }
    }
  ]
  // ...
}
```

Using the entity above as an example, It is possible to create an customer entity as following

```
// e.g: NodeJS example
fetch('https://april9.stack9.co/api/customer', {
  method: 'POST',
  headers: {
    'Api-Key': '<my_secret_api_key>',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    email: 'new_email@domain.com.au',
    full_name: 'John Doe',
    dob: '1993-04-02',
    country_id: 1,
    verified: true,
  }),
})
  .then(response => response.json())
  .then(data => console.log(data));

/** OUTPUT:
 * {
 *    "id": 4,
 *    "entity": {
 *       email: 'new_email@domain.com.au'
 *       full_name: 'John Doe'
 *       dob: '1993-04-02'
 *       country_id: 1
 *       verified: true
 *    }
**/ }
```

Status Code	Operation	Response
201	Resource Created successfully	{ ...entity_props}
422	Validation Error	{ error: { _original: {}, details: [] } }
404	Resource not found	

Edit Record

One of the operation of DATA-API is to edit an entity.

In order to edit a specific entity, a PUT request to /api/:entity_name/:id with json format payload enclosed in the body is sent.

The payload format will vary according to your Entity Definition which will define which format should be sent to each property and which fields are required.

```
// e.g.: customer.json [Entity Definition]
{
 // ...,
 "fields": [
  {
   "key": "email",
   "label": "email",
   "type": "TextField",
   "validateRules": {
    "required": false,
    "maxLength": 5
   }
  },
  {
   "key": "full_name",
   "label": "Full Name",
   "type": "TextField",
   "validateRules": {
    "required": false
   }
  },
  {
   "key": "dob",
   "label": "Date of Birth",
   "type": "DateField",
   "validateRules": {
      "required": false
    },
    "typeOptions": {
      "time": false
    }
  },
  {
    "key": "country_id",
    "label": "Country",
    "type": "SingleDropDown",
    "relationshipOptions": {
      "many": false,
      "ref": "country"
    },
    "typeOptions": {
      "label": "name"
    },
    "validateRules": {
      "required": false
    }
  },
  {
    "key": "verified",
    "label": "Vefified",
    "type": "Checkbox",
    "validateRules": {
      "required": false
    }
  }
 ]
 // ...
}
```

Using the entity above as an example, It is possible to edit an customer entity as following

```
// e.g: NodeJS example
fetch('https://april9.stack9.co/api/customer/1', {
 method: 'PUT',
 headers: {
  'Api-Key': '<my_secret_api_key>',
  'Content-Type': 'application/json',
 },
 body: JSON.stringify({
  email: 'new_email@domain.com.au',
  full_name: 'John Doe',
  dob: '1993-04-02',
  country_id: 1,
  verified: true,
 }),
})
 .then(response => response.json())
 .then(data => console.log(data));

/** OUTPUT:
 * {
 *    "id": 4,
 *    "entity": {
 *      email: 'new_email@domain.com.au'
 *      full_name: 'John Doe'
 *      dob: '1993-04-02'
 *      country_id: 1
 *      verified: true
 *    }
**/ }
```

Status Code	Operation	Response
204	Resource edited successfully	
422	Validation Error	{ error: { _original: {}, details: [] } }

Delete Record

One of the operation of DATA-API is to delete an entity.

In order to delete a specific entity, a DELETE request to /api/:entity_name/:id is sent.

```
// e.g: NodeJS example
fetch('https://april9.stack9.co/api/customer/1', {
  method: 'DELETE',
  headers: {
    'Api-Key': '<my_secret_api_key>',
    'Content-Type': 'application/json',
  },
});
```

Status Code	Operation
204	Resource Deleted successfully

Search Records

One of the operation of DATA-API is to list the records of a specific entity according to a specific condition.

In order to get a list of a specific entity, a POST request to /api/:entity_name/export.

The search accepts all the Stack9 Query Object to be passed through the request body

Example

For instance, let's say that we need to query all the customers who fit as Baby Boomers (generation X) that call John.

```
// NodeJS example
fetch('https://april9.stack9.co/api/customer/search', {
  method: 'POST',
  headers: {
    'Api-Key': '<my_secret_api_key>',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    $select: ['id', 'dob', 'name', 'account_type.name'],
    $withRelated: ['account_type'],
    $where: {
      name: {
        $like: '%John%',
      },
      dob: {
        $lt: '1964-12-31',
        $gt: '1946-01-01',
      },
    },
    $sort: {
      dob: -1,
    },
  }),
})
  .then(response => response.json())
  .then(data => {
    const johnBoomers = data;
    console.log(johnBoomers);
  });

// OUTPUT
// [
//   {
//     id: 12,
//     dob: '1963-06-02',
//     name: 'John Doe',
//     account_type: {
//       name: 'Basic'
//     },
//   },
//   {
```

```
//    id: 13,
//    dob: '1946-03-10',
//    name: 'Silvester John',
//    account_type: {
//      name: 'Premium'
//    },
//  },
//  ...
// ];
```

Also you can use GET parameters to create pagination or add a limit to the query results

Params	Type	Description
limit	number	The limit size for the query
page	number	Page index that represents the offset of your pagination

```
// NodeJS example
fetch('https://april9.stack9.co/api/customer/search?limit=2&page=0', {
  method: 'POST',
  headers: {
    'Api-Key': '<my_secret_api_key>',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    $where: {
      gender: 'Male',
    },
  }),
})
  .then(response => response.json())
  .then(data => console.log(data));

// OUTPUT
// {
//   results: [
//    {
```

```
//      id: 1,
//      dob: '1990-06-01',
//      name: 'John Doe'
//    },
//    {
//      id: 2,
//      dob: '1986-03-10',
//      name: 'James Doe'
//    },
//  ],
//  total: 10
// }
```

Using the total property, you can calculate how many pages are needed to paginate the records

```
const limit = 2;
const pagesNeeded = Math.round(data.total / limit);
console.log(pagesNeeded); // 5
```

Responses

Status Code	Operation	Response
200	Resource listed successfully	Entity[] or {results: Entity[]}
400	Bad Request Error	{ error: string }
404	Resource not found	

Export data

Stack9 allows to export entity data to a CSV. If no configuration is provided, all entity fields will be exported.

Configuration

In the entity definition JSON file, customise what fields should be part of the export:

```
 "exportSettings": {
   "useFieldKeyAsHeader": false,
   "columnDefs": [
     { "headerName": "Text Field", "field": "text_field" },
     { "headerName": "Id", "field": "id" },
    { "headerName": "Updated At", "field": "_updated_at" }
  ]
 }
```

Definition

useFieldKeyAsHeader - bool (optional, default false) - Choose between columnDefs.headerName or the field.key defined in the entity as column header name.

columnDefs - array (optional) - List of fields to be exported.

columnDefs.headerName - string (optional) - Label for the column header.

columnDefs.field - string (required) - Field key or association path for many-to-one (*) fields

(*) one-to-many fields relationship are not supported.

Endpoint

POST /:entity_type/export

Querystring limit=[integer]&page=[integer]

Body [query criteria]

Success code 200

Response type byte[]

Sample Call:

```
POST /user/export?limit=100
{
  "$where": { "is_active": true }
}
```