



MongoDB Atlas and Stitch Workshop

Stitch and Atlas Workshop

What is MongoDB Stitch?	5
Stitch Features	5
The components we will be using in this workshop	7
Atlas Free Tier Setup	10
React App	12
Create a Stitch App	14
Explore the Journal React App	15
User Authentication	17
Stitch QueryAnywhere	21
Configuring Database and Collection	23
CRUD Operations	23
MongoDB Rules	26
Stitch Services	29
Stitch Functions	30
MongoDB Triggers	33
Stitch Logs	35

Agenda

- Introduction to Stitch features
- Configuring an Atlas Cluster.
- Initializing a Stitch App
- Creating React App and using Stitch as backend

Logistics

You will need:

- Node.js installed.
- Your favorite text editor.
- A web browser
- MongoDB Atlas account
- Follow via [slides](http://52.15.141.92:8000/build/stitch-workshop-student/stitch-workshop-student.slides.html#/) (<http://52.15.141.92:8000/build/stitch-workshop-student/stitch-workshop-student.slides.html#/>) or [pdf](http://52.15.141.92:8000/build/stitch-workshop-student/stitch-workshop-student.pdf) (<http://52.15.141.92:8000/build/stitch-workshop-student/stitch-workshop-student.pdf>).

Short Links

Slides

<https://tinyurl.com/y82egwul> (<https://tinyurl.com/y82egwul>)

PDF

<https://tinyurl.com/y89kd45w> (<https://tinyurl.com/y89kd45w>)

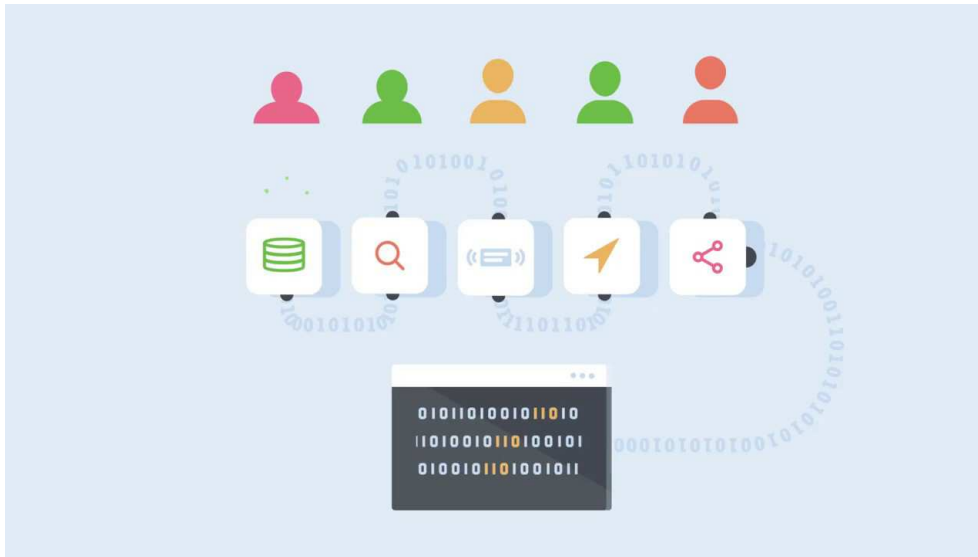
Learning objectives

- Develop a functional Stitch App.
- Demonstrate how to use Stitch functions from a local App.
- Create a Stitch app from the Atlas UI and link it to your cluster.
- Integrate Stitch into a React web app.

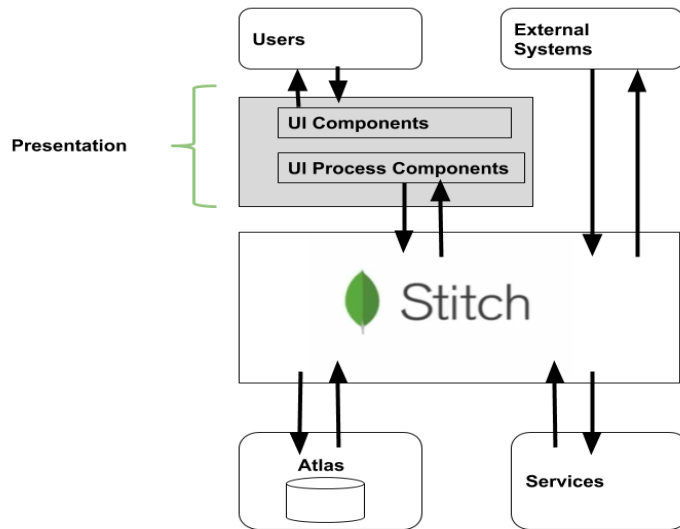
What is MongoDB Stitch?

- MongoDB Backend as a Service offering (BaaS).
- At the core of Stitch's back end is MongoDB managed and configured as per best practices.
- Access to your data through a REST-like API.
- Lets you focus on your app's functionality with declarative statments.
- Integrate third party services easily.
- Integrated security, authentication and access roles.

Stitch Features



Overview



QueryAnywhere

Bring MongoDB's rich query language safely to your application.

Build full apps for iOS, Android, Web, and IoT

Functions

Integrate microservices, server-side logic, and cloud services.

Power apps or enable Data as a Service with custom APIs.

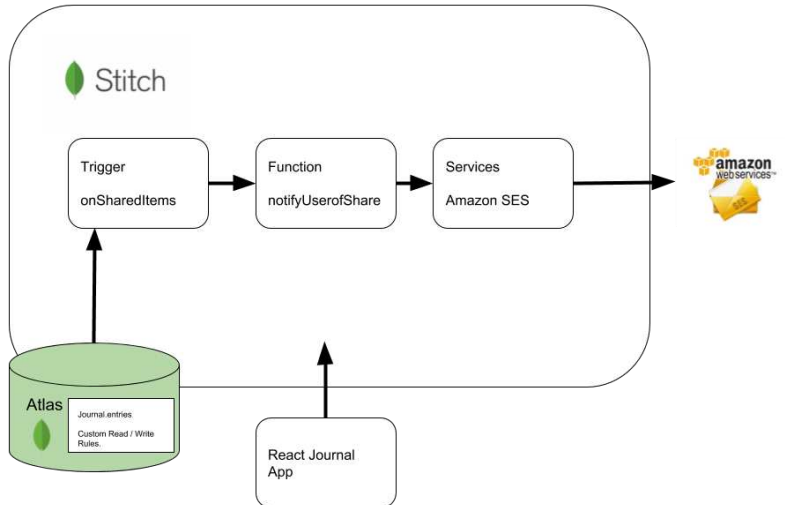
Triggers

Use database change events to trigger functions in real time.

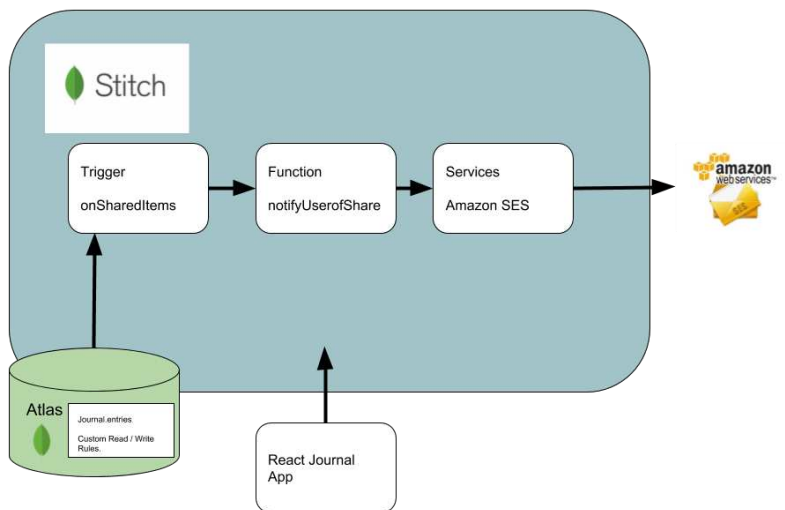
Respond immediately to changing data.

The components we will be using in this workshop

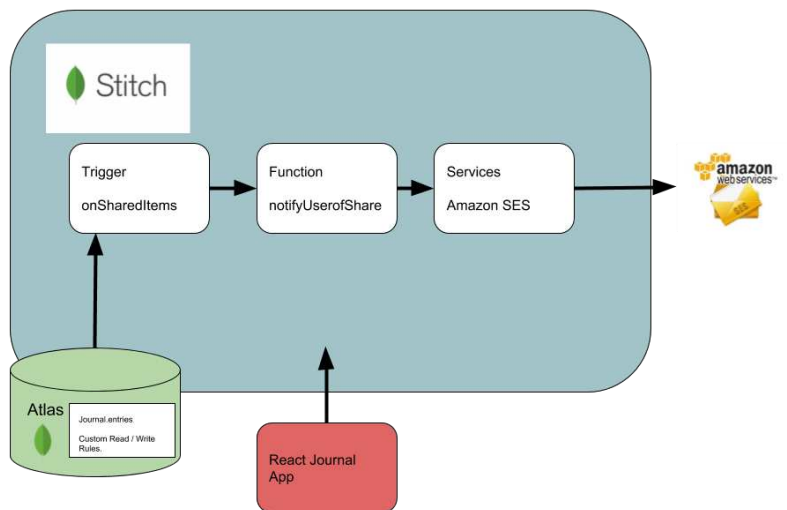
1. Atlas Free Tier cluster:



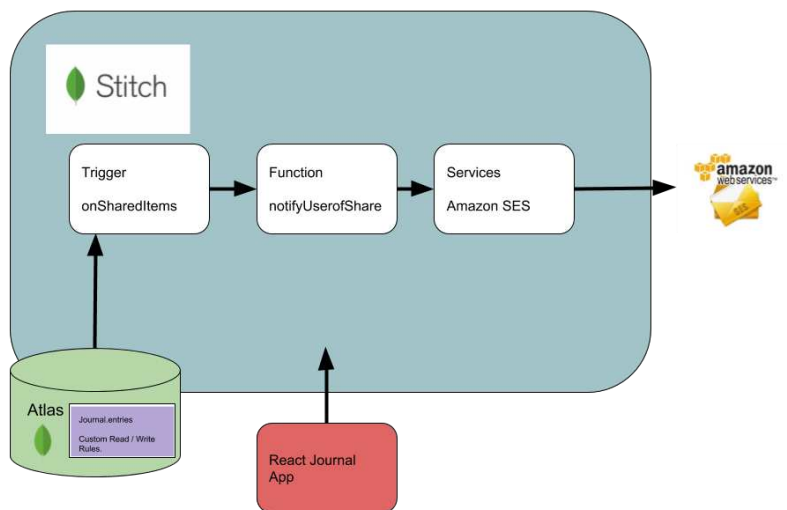
2. Stitch App



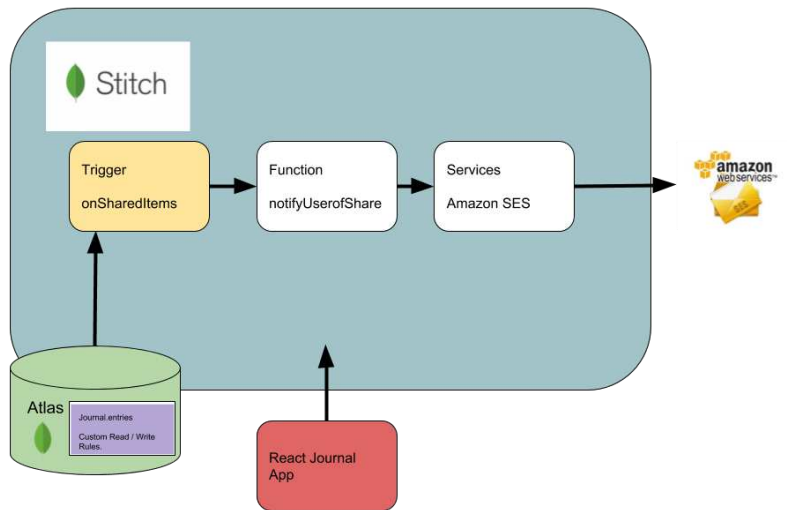
3. React App linked to Stitch using the JavaScript SDK.



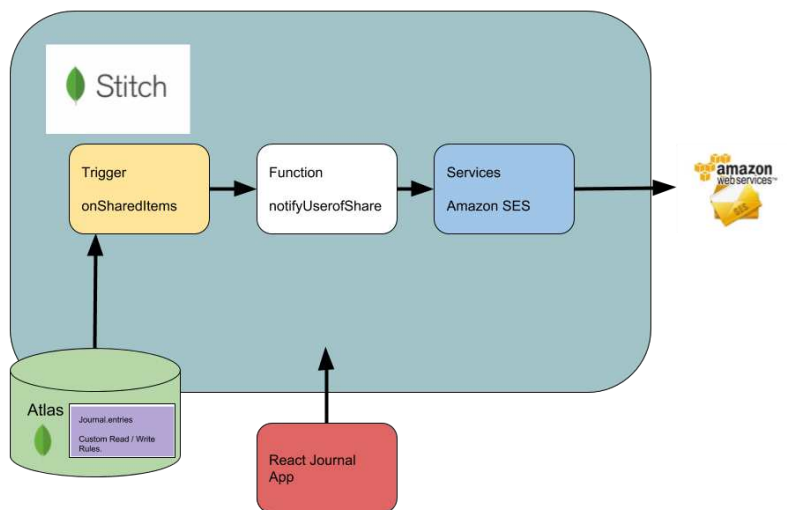
4. Configuring Namespace ACL via Stitch Roles



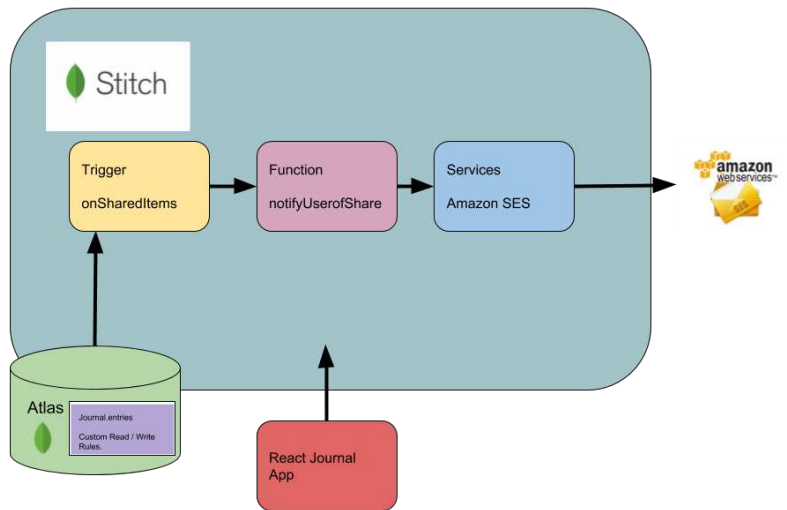
5. Stitch trigger to listen for change events



6. Stitch service to send notifications via email using AWS services



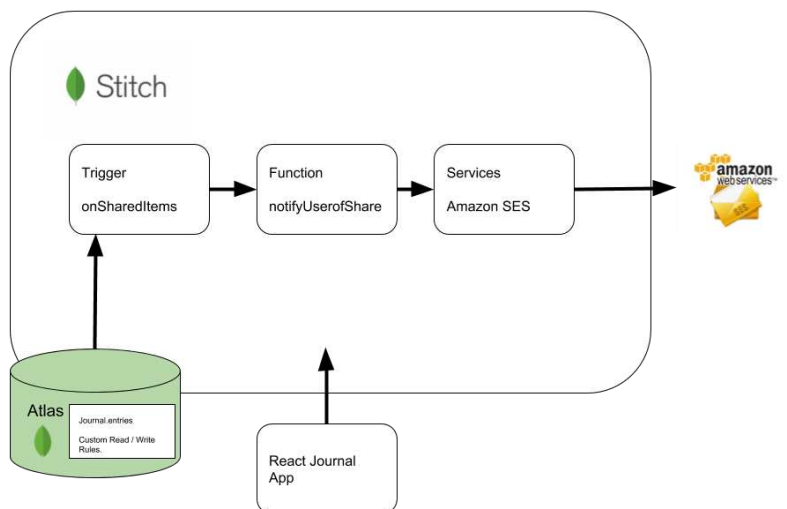
7. Stitch function which will be fired by the trigger.



Atlas Free Tier Setup

Create a free tier Atlas cluster.

Steps to do this can be found [here](https://docs.mongodb.com/manual/tutorial/atlas-free-tier-setup/) (<https://docs.mongodb.com/manual/tutorial/atlas-free-tier-setup/>).



Create an Atlas Organization

Account Settings

Profile Personalization **Organizations** Public API Access

[← Organizations](#)

Create Organization

Name and Service

Add Members

Next

Name Your Organization

Select Cloud Service

Features

Automated database configuration

☒ MongoDB Atlas

☐ Cloud Manager



Create an Atlas Project in your Organisation

Atlas **OK** All Clusters Usage This Month:\$0.00 [details](#) Admin Barry ▾

[STITCH_WORKSHOP > PROJECTS](#)

stitchworkshop ▾

Create a Project

Name Your Project

Add Members

Next

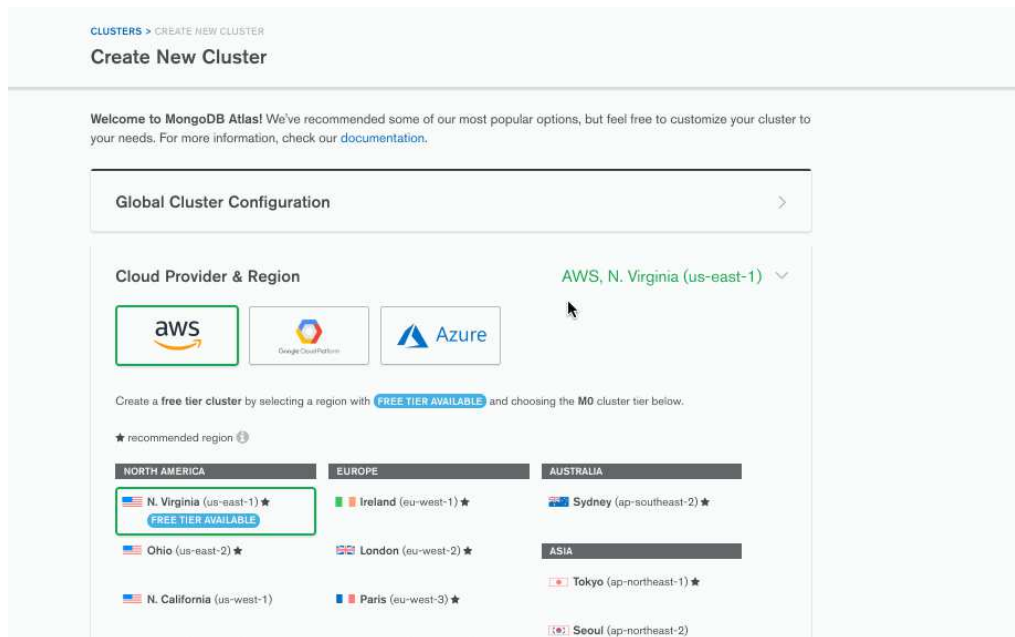
Name Your Project

Project names have to be unique within the organization (and other restrictions).

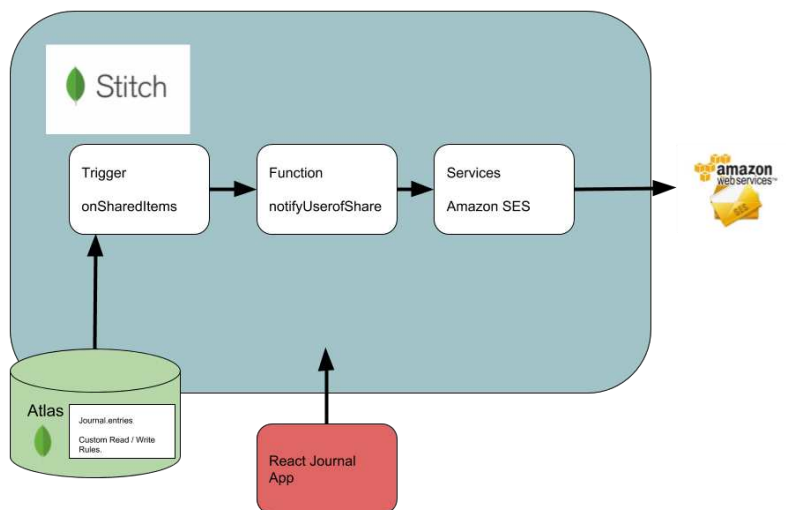
Cancel

Next

Select a Free Tier cluster



React App



Download: <https://s3.amazonaws.com/mongodb-training/stitch-workshop/stitch-workshop-exercise.zip> (<https://s3.amazonaws.com/mongodb-training/stitch-workshop/stitch-workshop-exercise.zip>).

Starting your React App

At this point we need to install and start our App.

Go ahead and give it a try...

Starting the React App from the shell.

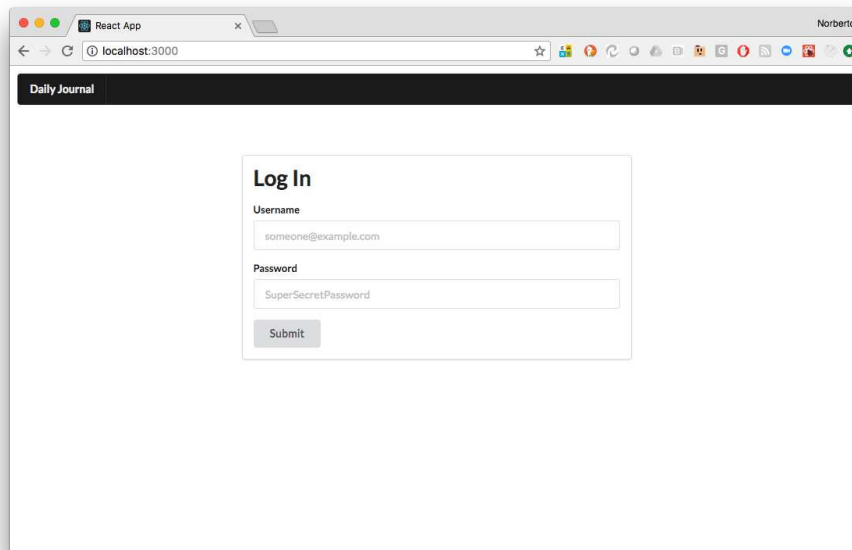
```
cd stitch-workshop-exercise
```

```
npm install
```

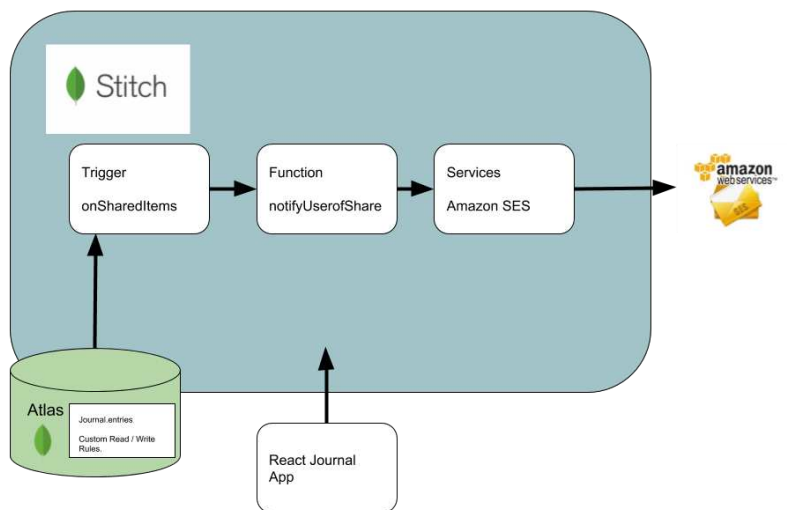
```
npm start
```

Once the service is loaded this should open a browser window. If this is not automatic, try to access this URL from your browser:

<http://localhost:3000> (<http://localhost:3000>).



Create a Stitch App

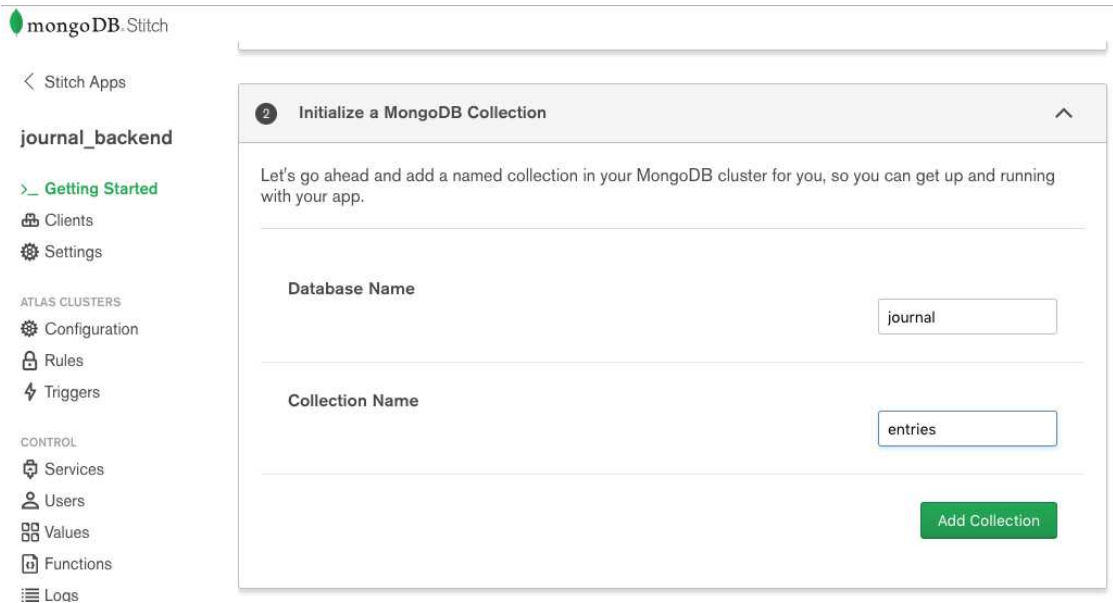


Stitch App UI



In your Atlas dashboard go to the *Stitch Apps* page and press the *Create New Application* button. Set an *Application Name* (`journal_backend`) and select the **stitch_atlas** cluster in the *Link to Cluster* dropdown.

Initialize a MongoDB collection.



mongoDB Stitch

< Stitch Apps

journal_backend

> Getting Started

Clients

Settings

ATLAS CLUSTERS

Configuration

Rules

Triggers

CONTROL

Services

Users

Values

Functions

Logs

2 Initialize a MongoDB Collection

Let's go ahead and add a named collection in your MongoDB cluster for you, so you can get up and running with your app.

Database Name

Collection Name

Add Collection

Database and Collection names

Use the following as Database and Collection names:

- Database: **journal**
- Collection: **entries**

Mid-Flight Check 1

So far we have:

- Created an Atlas cluster
- Downloaded and installed the React Journal App
- Created the Stitch App

Explore the Journal React App

MongoDB Stitch Browser SDK

MongoDB provides a browser SDK (JavaScript) that allows developers to perform client side commands calls from the application front-end. To show you that, we've included in the `source/index.js` file the SDK package.

In this example, we are importing *Stitch*, *UserPasswordCredential* and *RemoteMongoClient* objects:

```
// MongoDB Stitch sdk here
import {
  Stitch,
  UserPasswordCredential,
  RemoteMongoClient
} from "mongodb-stitch-browser-sdk";
```

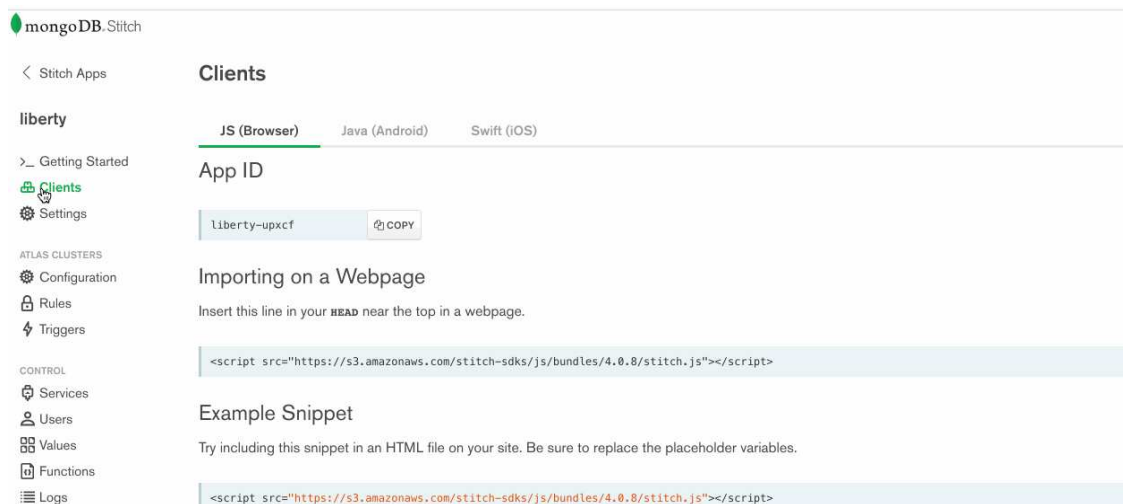
Configuring Stitch appId

Replace with the App ID of your Stitch App.

```
ReactDOM.render(
  <StitchApp appId="<your_app_id>" />,
  document.getElementById("root")
);
```

We are going to pass the unique id of the Stitch App that you created earlier.

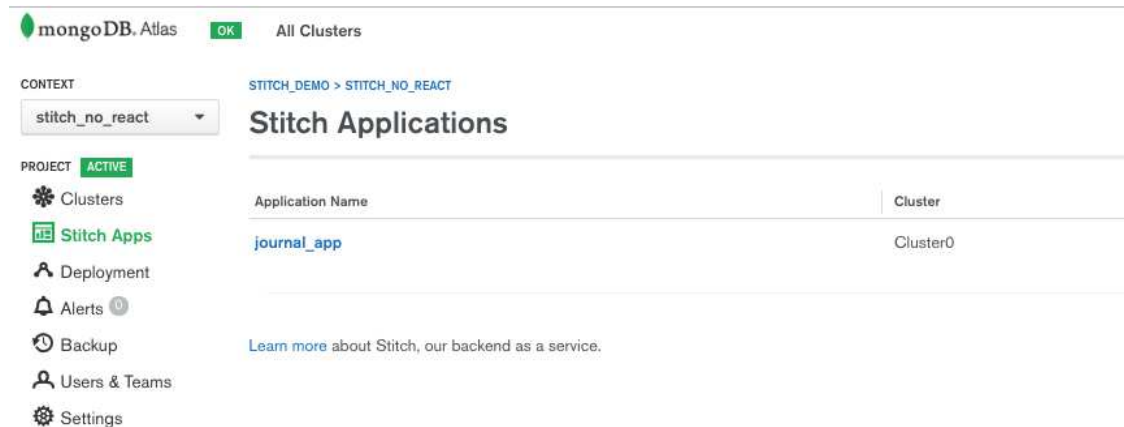
At this stage our React App is able to connect to Stitch. However, we are not able to do anything because we do not have any application users created yet!



The screenshot shows the MongoDB Stitch dashboard interface. On the left is a sidebar with navigation links: **liberty**, `>_ Getting Started`, **Clients** (highlighted), **Settings**, **ATLAS CLUSTERS**, **Configuration**, **Rules**, **Triggers**, **CONTROL**, **Services**, **Users**, **Values**, **Functions**, and **Logs**. The main content area is titled **Clients** and has tabs for **JS (Browser)**, **Java (Android)**, and **Swift (iOS)**. Under the **JS (Browser)** tab, there is a section for **App ID** showing the ID `liberty-upxcf` with a **COPY** button. Below this is a section for **Importing on a Webpage** with the instruction "Insert this line in your **HEAD** near the top in a webpage." and a code snippet: `<script src="https://s3.amazonaws.com/stitch-sdks/js/bundles/4.0.8/stitch.js"></script>`. At the bottom is an **Example Snippet** section with the instruction "Try including this snippet in an HTML file on your site. Be sure to replace the placeholder variables." and the same code snippet.

In order to copy the **App ID** from the Stitch dashboard got to the *Clients* page and copy the code under *App ID*

Check that your App is linked to your cluster



The screenshot displays the MongoDB Atlas interface for managing Stitch Applications. The main heading is 'Stitch Applications'. Below it, a table lists the applications:

Application Name	Cluster
journal_app	Cluster0

Below the table, there is a link: [Learn more about Stitch, our backend as a service.](#)

User Authentication

Secure Data Access: Simple, Declarative Rules



- Complete Authentication & Authorization out of the box: Authenticate anything and anyone; protect anything
- Precise, Flexible, Extensible Rules: Lock access down tighter
- Flexible Auth Options: Choose which auth service(s) to use

Authentication Mandatory!

A particular aspect of MongoDB Stitch is that Authentication is mandatory for anything we want to do on the Backend side!

Authentication Providers

- Simple Login API
- Pluggable Authentication Providers
 - Email/Password
 - OAuth (Facebook/Google)
- Custom Authentication using Json Web Token(JWT)
- Multiple Providers per App
 - Link user accounts
 - Log in with any provider

What is an Authentication Provider in Stitch?

- A pluggable way for you to allow users to login using different methods (Facebook, Google, email).
- You can use multiple providers to login to a single account.
- Users are stored at Stitch level and not in your Atlas cluster.
- Data access permissions can be defined at collection and document level to allow fine grained data access control.

Configuring a Authentication Provider

For this app, we will use *Email/Password* authentication

In your Stitch App, click "*Users*" on the left hand sider of the UI:

CONTROL



Services



Users



Values



Functions



Logs



Push Notifications

Enable Authentication Provider

Next, click the "Providers" tab in the UI

Users

Users

Providers

Provider	Status	Actions
Allow users to log in anonymously	Disabled	<button>EDIT</button>
Email/Password	Disabled	<button>EDIT</button>
Facebook	Disabled	<button>EDIT</button>
Google	Disabled	<button>EDIT</button>
API Keys	Disabled	<button>EDIT</button>
Custom Authentication	Disabled	<button>EDIT</button>

To enable this Authentication Provider, select *Users* page and *Providers* tab. Select the *Email/Password* by clicking on the *EDIT* button and enable the *Provider Status*

Configure Authentication Provider settings

...and complete the following fields

Provider Status <small>Allow your users to sign in with this authentication method.</small>	<input checked="" type="checkbox"/> Enabled
Email Confirmation URL	<input type="text" value="http://localhost:8000/confirm"/>
Password Reset URL	<input type="text" value="http://localhost:8000/reset"/>
Reset Password Email Subject	<input type="text" value="reset"/>
E-Mail Confirmation Subject	<input type="text" value="confirm"/>

[Discard Changes](#)
[Save](#)

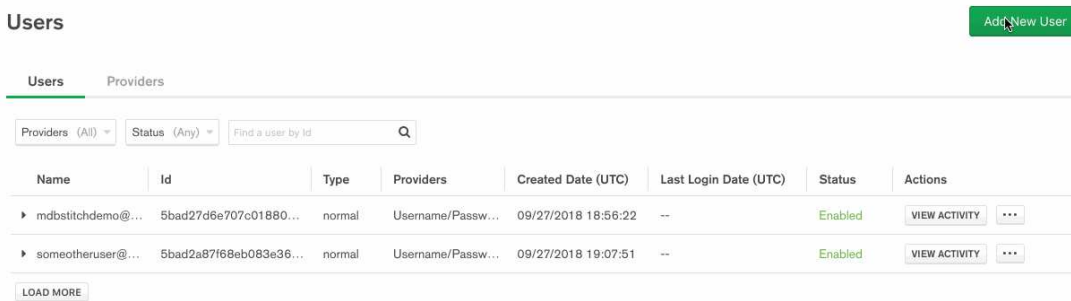
Configure Authentication Provider settings (cont)

Please copy and paste the values below into the Stitch UI.

- Email confirmation URL: `http://localhost:8080/confirm`
- Password Reset URL: `http://localhost:8080/reset`
- Reset Password Subject: `reset`
- E-Mail Confirmation Subject: `confirm`

Create Application Users

Create two users



Users Add New User

Users Providers

Providers (All) Status (Any) Find a user by Id

Name	Id	Type	Providers	Created Date (UTC)	Last Login Date (UTC)	Status	Actions
▶ mdbstitchdemo@...	5bad27d6e707c01880...	normal	Username/Passw...	09/27/2018 18:56:22	--	Enabled	VIEW ACTIVITY ...
▶ someotheruser@...	5bad2a87f68eb083e36...	normal	Username/Passw...	09/27/2018 19:07:51	--	Enabled	VIEW ACTIVITY ...

LOAD MORE

To create the users go to *Users* page and click the button *Add New User* and fill in the user details with the following *Username* and *Password*.

User details

Create the following two users. You can access the Gmail accounts using the credentials below:

- Username: `libertystitch1@gmail.com`
 - Password: `LibertyStitch1`
-

- Username: `libertystitch2@gmail.com`
- Password: `LibertyStitch2`

We will use these to send email notifications from Stitch

Mid-Flight Check 2

So far we have:

- Created an Atlas cluster and linked it to our Stitch App.
- Configured our app to connect to Stitch using the Stitch SDK.
- We have created our authentication provider in Stitch and added the methods to the App.
- Our React App now has the functionality to allow people to login.

Next step: implement **CRUD** operations in our React App

Stitch QueryAnywhere

- To allow users to write `Entries`, we will be using the MongoDB Service.
- What is the *MongoDB Service*?
 - Familiar query language.
 - `find`, `insert`, `update`, `aggregate`
 - Database access from client code (front-end)
 - User level database access control.

Import the RemoteMongoClient from the Stitch SDK

As you can see from the `src/index.js` we have provided (line 14), we have imported the `RemoteMongoClient` from the Stitch SDK:

```
import {
  Stitch,
  UserPasswordCredential,
  RemoteMongoClient
} from "mongodb-stitch-browser-sdk";
```

Component Initialization

In line 59 of the `src/index.js` file, we can see that `this.mongodb` is passed to our `Journal` component:

```
return (
  <Page
    currentUser={currentUser}
    logoutCurrentUser={this.logout}
  >{
    isAuthenticated ? (
      <Journal
        mongodb={this.mongodb}
        currentUser={currentUser}
      />
    ) : (
      <Login loginUser={this.login} />
    )
  }
  </Page>
...

```

This is where we will be working next...

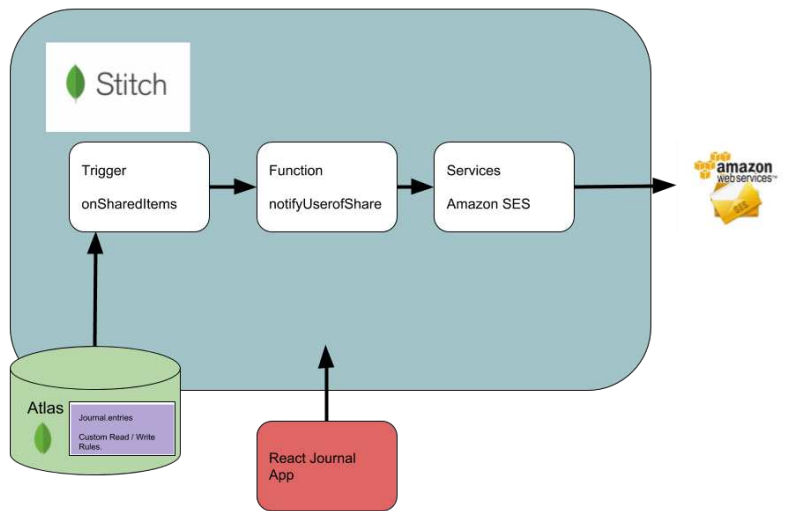
The Journal component

Edit file `src/components/Journal/index.js`

We can see that the code at line 22 is taking the current user that is logged in, as well as our `mongodb` object:

```
Class Journal extends Component {
  static propTypes = {
    currentUser: PropTypes.any.isRequired,
    mongodb: PropTypes.object.isRequired,
  };
}
```

Configuring Database and Collection



Referencing a database and collection

We need to reference a database and a collection in the journal code.

Add your database and collection name at line 25 in the `src/components/Journal/index.js` file we are working on for the journal component:

```

constructor(props) {
  super(props);
  const { mongodb } = this.props;
  this.entries = mongodb.db("<your_database>").collection("<your_collection>");
  this.state = {
    entries: []
  };
}
  
```

CRUD Operations

CRUD stands for create, read, update and delete operations.

Loading Entries at startup time

- We need to have the React App load all existing entries from the database when the page loads.
- In React, we do this using the `componentDidMount()` method.
- As we are using Stitch and the MongoDB Service, we can use MongoDB query language directly here to load the entries.

Fetching Entries from MongoDB

We need to add the following line of code after the comment below:

```
// TODO: Fetch existing journal entries  
const entries = await this.entries.find({}).toArray();
```

How do our Entries look?

- Great that we can list `entries` but how is the schema being defined?

Entries model

```
//define entry here  
addEntry = async (title = "Untitled", body) => {  
  const { currentUser } = this.props;  
  const newEntry = {  
    title,  
    body,  
    owner_id: currentUser.id,  
    author: currentUser.profile.data.email,  
    date: new Date(),  
    sharedWith: []  
  };  
};
```

Inserting an Entry into MongoDB

We need to add the following lines of code after the comment below:

```
// TODO: Add newEntry to MongoDB here  
const result = await this.entries.insertOne(newEntry);  
newEntry._id = result.insertedId;
```


Removing an Entry

We need to add this code below the previous block we added paste the following at the comment below.

```
// TODO: Delete the entry from MongoDB
await this.entries.deleteOne({ _id: entryId });
```

Updating an Entry

Next, we are going to add a method to update an entry, paste the following code in below the comment below:

```
// TODO: Update the Entry body in MongoDB
await this.entries.updateOne(
  { _id: entryId },
  { $set: { body: newBody } }
);
```

Sharing an Entry

We need to add this code where the comment below is:

```
// TODO: Share Entry to the provided email by setting it in the sharedWith array
await this.entries.updateOne(
  { _id: entryId },
  { $push: { sharedWith: email } }
);
```

Unsharing an Entry

If we share, we should also be able to unshare:

```
// TODO: Remove the provided email from the Entry sharedWith array
await this.entries.updateOne(
  { _id: entryId },
  { $pull: { sharedWith: email } },
  { multi: true }
);
```

Next, do the following

- Login to your app using the first user you created.
- Add three Entries.
- Share one with the second user.
- Login with that second user.
- See if can see the shared entry.

A user should not be able to see the entries shared by another user yet...

We need to grant additional privileges from the server side!

Mid-Flight Check 3

So far we have:

- Created an Atlas cluster and linked it to our Stitch App.
- Configured our app to connect to Stitch using the Stitch SDK.
- We have created our authentication provider in Stitch and added the methods to the App.
- Our React App now has the functionality to allow people to login.
- We have added methods to add, update and delete entries.
- We have added a method to share certain posts with specified users.

MongoDB Rules

ATLAS CLUSTERS



Configuration



Rules



Triggers

MongoDB Rules allow you to

- Specify exactly who sees what data
 - Customized for the Current User
 - Configured per Collection
- Enforce Document Schemas
- Fully Configurable
 - Pre-configured Templates
 - "Advanced Mode" JSON

Adding a Stitch role

Our rules should look like this once configured



Collections

+ ADD COLLECTION

▼ journal

entries

◀ journal.entries

Permissions*

Schema

Filters

	<div>1 owner</div> <div>◀ ▶ ✎ 🗑</div>	<div>2 shared</div> <div>◀ ▶ ✎ 🗑</div>
Fields	<div>Read</div> <div>✓</div> <div>Write</div> <div>✓</div>	<div>Read</div> <div>✓</div> <div>Write</div> <div>□</div>
+ ADD FIELD		
All Additional Fields	<div>✓</div> <div>✓</div>	<div>✓</div> <div>□</div>

Default owner role

By default, Stitch creates an "owner" role which specifies that only the owner of an entry can view it's own entries:

1 owner

Name
owner

Apply When
For each document returned by a request, define the conditions for when this role should be applied.
[Learn more about Apply When.](#)

1 {
2 "owner_id": "%user.id"
3 }

Document-Level Permissions
Define document-level permissions for this role.

☒ Insert Documents
☒ Delete Documents

CLOSE

Creating custom role

If Entry posts are to be shared with other application users, we need to create a role that enables the user to read is own posts, as well as the ones shared with him:

2 sharedWith

Name
sharedWith

Apply When
For each document returned by a request, define the conditions for when this role should be applied.
[Learn more about Apply When.](#)

1 {
2 "sharedWith": "%user.data.email"
3 }

Document-Level Permissions
Define document-level permissions for this role.

☐ Insert Documents
☐ Delete Documents

CLOSE

Create a `sharedWith` Role

Copy this json document into the *Apply When* box of your new role:

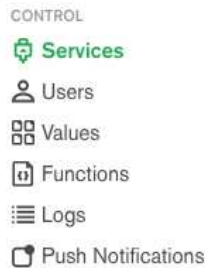
```
{
  "sharedWith": "%user.data.email"
}
```

Up Next: send email to users when a comment is shared with them

- To implement this feature we will use a Stitch function and trigger in parallel.
- Stitch functions are serverless Java functions that can connect to other services.
- For this function, we are going to use AWS Simple Email Service (SES) to send an email when an entry is shared with a user.

Stitch Services

We need to create a stitch service to enable our function to send mails to a recipient using AWS SES.



AWS Keys for SES service

To do this we will need to use the following AWS keys. Copy them into the Stitch UI as illustrated in the next slide:

Access key ID:

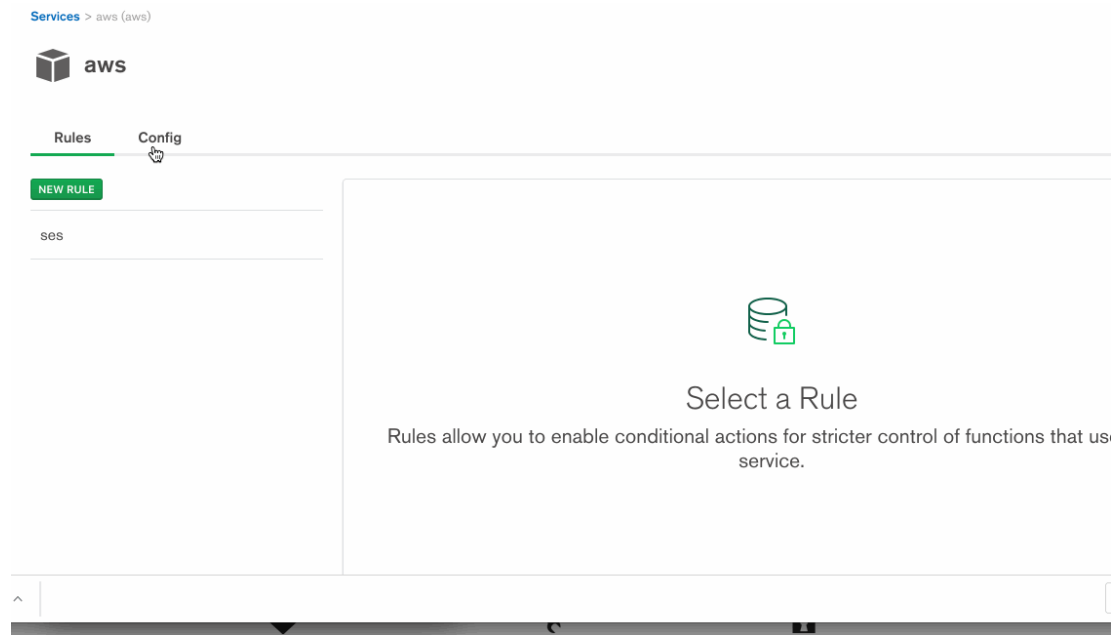
AKIAI6MY7Z4VLO232BNA

Secret access key:

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

Configure AWS Simple Mail Service (SES)

Create a service with the service name **aws** with a rule named **ses**:



To create the service go to *Services* page and select an **AWS** service. Once created go to the *Rules* tab and click on the *New Rule* button. Name the rule as **ses** and add a *Action* from the *API* dropdown named **ses** with *All Actions* selected. There is nothing to provide in the *When* section.

Stitch Functions

CONTROL

 Services

 Users

 Values

 **Functions**

 Logs

 Push Notifications

Stitch function editor

Functions > notifySharedUsers

notifySharedUsers

Function Editor Settings

```

1 // notifyUserOfShare
2 exports = function(changeEvent){
3   const { updateDescription, fullDocument } = changeEvent;
4   const updatedFields = Object.keys(updateDescription.updatedFields);
5
6   const sharedWithFieldWasUpdated = updatedFields.some(
7     field => /sharedWith/.test(field)
8   );
9
10
11 if(sharedWithFieldWasUpdated) {
12   const { sharedWith, author, title } = fullDocument;
13   const sharedWithEmail = sharedWith(sharedWith.length - 1);
14   if(!sharedWithEmail) { return }
15
16   const aws = context.services.get("aws");
17   var from = "libertystitch2@gmail.com";
18   var input = {
19     Destination: {
20       ToAddresses: [sharedWithEmail]
21     },
22     Message: {
23       Body: {
24         Html: {
25           Charset: "UTF-8",
26           Data: "This is a message from" + context.user.data.email
27         }
28       }
29     }
30   };
31   aws.sendEmail({
32     From: from,
33     Message: input,
34     Destination: {
35       ToAddresses: [sharedWithEmail]
36     }
37   });
38 }

```

Stich notifyUserOfShare function

```

// notifyUserOfShare
exports = function(changeEvent){
  const { updateDescription, fullDocument } = changeEvent;
  const updatedFields = Object.keys(updateDescription.updatedFields);

  const sharedWithFieldWasUpdated = updatedFields.some(
    field => /sharedWith/.test(field)
  );
  ...
}

```

<https://gist.github.com/nleite/8862266f693bb87f83570a80710306a8>
 (https://gist.github.com/nleite/8862266f693bb87f83570a80710306a8).

Copy this code into the function editor in the Stitch UI and save it as notifySharedUsers.

notifySharedUsers code:

```
// notifyUserOfShare
exports = function(changeEvent){
  const { updateDescription, fullDocument } = changeEvent;
  const updatedFields = Object.keys(updateDescription.updatedFields);

  const sharedWithFieldWasUpdated = updatedFields.some(
    field => /sharedWith/.test(field)
  );

  if(sharedWithFieldWasUpdated) {
    const { sharedWith, author, title } = fullDocument;
    const sharedWithEmail = sharedWith[sharedWith.length - 1];
    if(!sharedWithEmail) { return }

    const aws = context.services.get("aws");
    var from = context.user.data.email;
    var input = {
      Destination: {
        ToAddresses: [sharedWithEmail]
      },
      Message: {
        Body: {
          Html: {
            Charset: "UTF-8",
            Data: "This is a message from" + context.user.data.email
          }
        },
        Subject: {
          Charset: "UTF-8",
          Data: "you got a new journal post: " + title
        }
      },
      Source: from
    };
    try{

      console.log(JSON.stringify(input));
      aws.ses().SendEmail(input).then(function (result){
        console.log(JSON.stringify(result));
      });
    } catch(error){
      console.log(JSON.stringify(error));
    }
  }
  console.log("done");
};
```


MongoDB Triggers

ATLAS CLUSTERS

 Configuration

 Rules

 **Triggers**

Triggers

- Fire in Response to Data Changes
 - Built on MongoDB Change Streams
 - Pass Change Events to Functions
 - Use Multiple Triggers per Collection

Creating Trigger in the Stitch UI

Edit Trigger

⌂ RESTART⋮

▼ TRIGGER DETAILS

Trigger Name

This is the name your trigger will use in the Stitch Admin UI and application configuration.

Enabled

☐

▼ TRIGGER SOURCE DETAILS

Database Name

Collection Name

33

To create a trigger we need to perform the following tasks:

- Go to the *Triggers* page and click on *Add Database Trigger* button.
- Set *Trigger Name* as `onSharedItem`
 - Set *Database Name* as `journal`
 - Set *Collection Name* `entries`
 - Check mark the *Operation Type* the *Insert*, *Update* and *Replace* options.
- In *Linked Function* select the *Function* `notifyUserOfShare` from the dropdown.

Full Document

Get the full document in your [change event](#).

☒

LINKED FUNCTION

Function

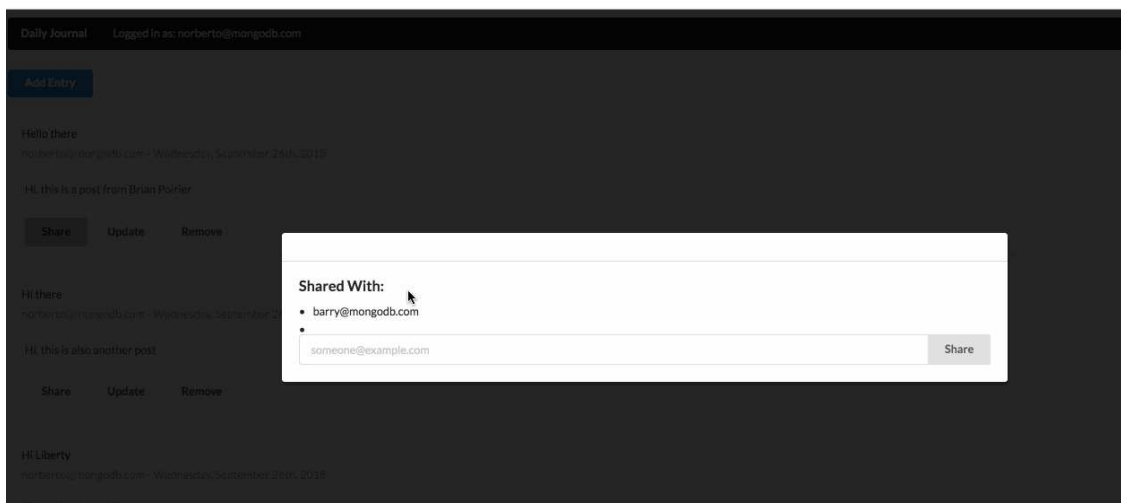
Select the function to be executed on a [change event](#). Selecting a new function will create a default function you can edit in the future.

Notify a user when an Entry is shared

At this point, our Stitch App would send a mail to a user when they have been added the `sharedWith` array.

Share an Entry with another user

Let's give it a try.



Stitch Logs

Apart from checking our email, we can check if this was successful by checking the Stitch logs:

CONTROL



Services



Users



Values



Functions



Logs



Push Notifications

Stitch function log

We can see that there was a `$push` operation against the `sharedWith` array at the time we shared the post:

▼ OK	2018-10-02T18:56:36-04:00	443ms	5bb3f7a4698a67f1a9...	5babff013dcb8b710c...	updateOne	Function	MONGODB-ATLAS
<pre>{ "name": "updateOne", "service": "mongodb-atlas", "functionArguments": { { "collection": "entries", "database": "journal", "query": { "_id": { "\$oid": "5bac029b1c9d440000b5aale" } }, "update": { "\$push": { "sharedWith": "" } } } } }</pre>							
<p>Arguments (JavaScript):</p> <pre>JSON.parse('{"collection":"entries","database":"journal","query":{"_id":{"\$oid":"5bac029b1c9d440000b5aale"}}, "update":{"\$push":{"sharedWith":""}}}')'</pre>							
<p>Compute Used: 2119943 bytes*ms</p>							

These logs are very useful for auditing any events in your application and can also be invaluable in root cause analysis for any issues.

Congratulations!



You now have a fully functioning React Journal App linked to your Atlas cluster through Stitch.

Recap

To achieve all this, we have...

- Created an Atlas cluster and linked it to our Stitch App
- Configured our app to connect to Stitch using the Stitch SDK
- Enabled a Stitch Authentication Provider
- Executed MongoDB commands from the front-end with QueryAnywhere
- Create content sharing (ACL) with a couple of rules
- Created a service linked to Amazon AWS SES
- Created triggers that use functions to send emails



Find out more
mongodb.com | mongodb.org
university.mongodb.com

Having trouble?
File a JIRA ticket:
jira.mongodb.org

Follow us on twitter
[@MongoDBInc](https://twitter.com/MongoDBInc)
[@MongoDB](https://twitter.com/MongoDB)