MongoDB University

MongoDB Atlas and Stitch Workshop

mongoDB

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#/) or 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)

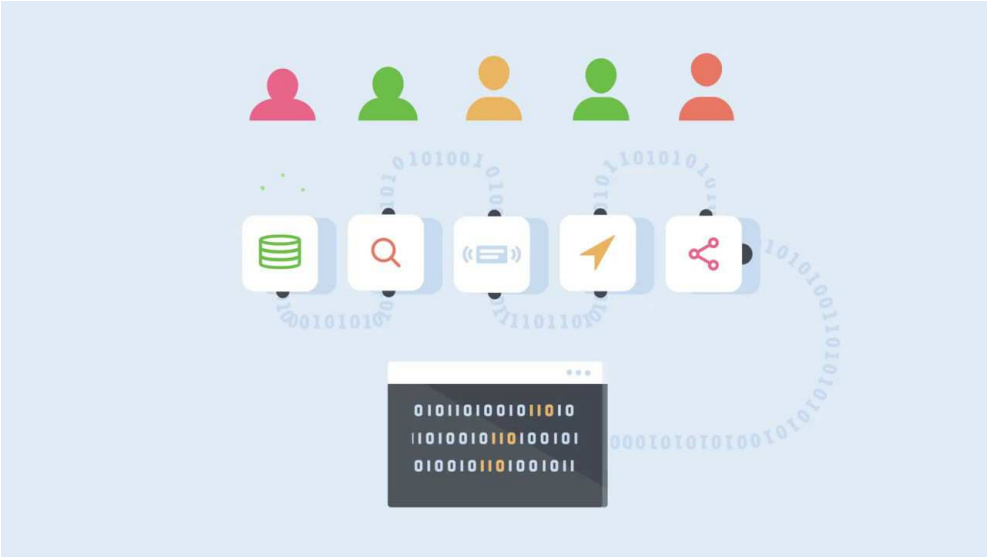
Learning objectives

- 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.

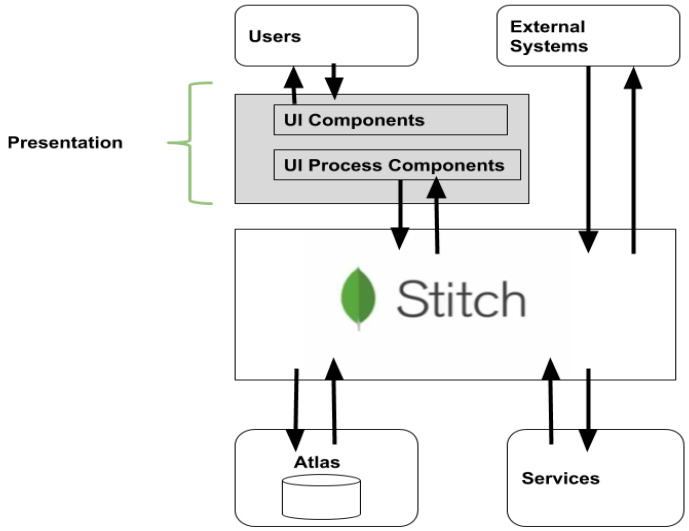
What is MongoDB Stitch?

- 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.

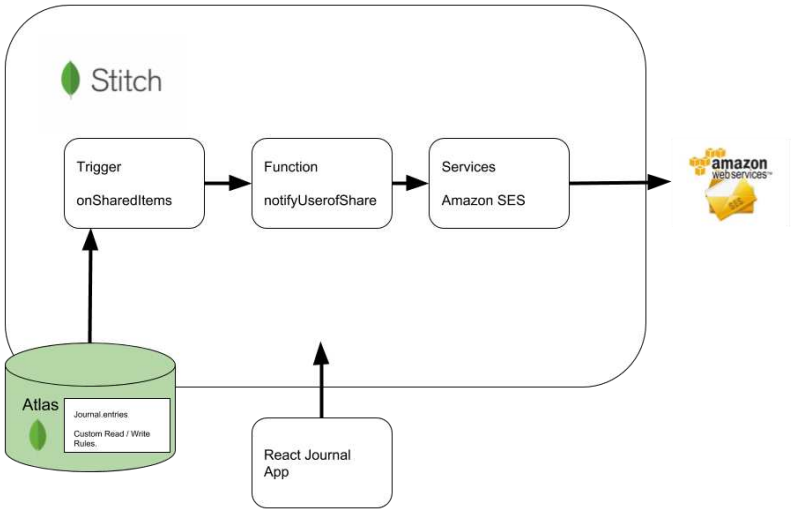
Triggers

Use database change events to trigger functions in real time.

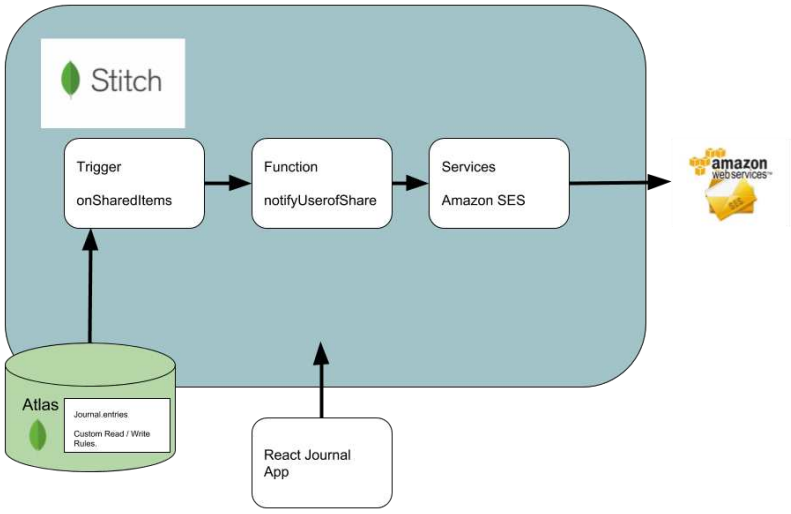
Respond immediately to changing data.

The components we will be using in this workshop

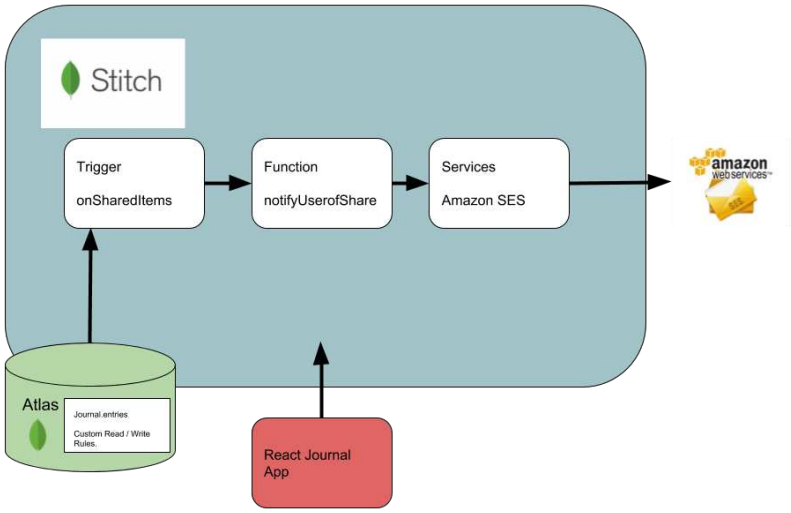
1. Atlas Free Tier cluster:



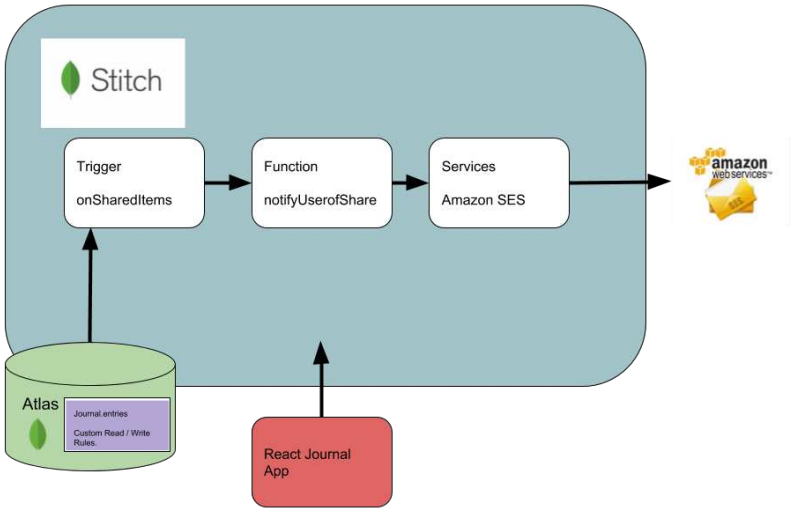
2. Stitch App



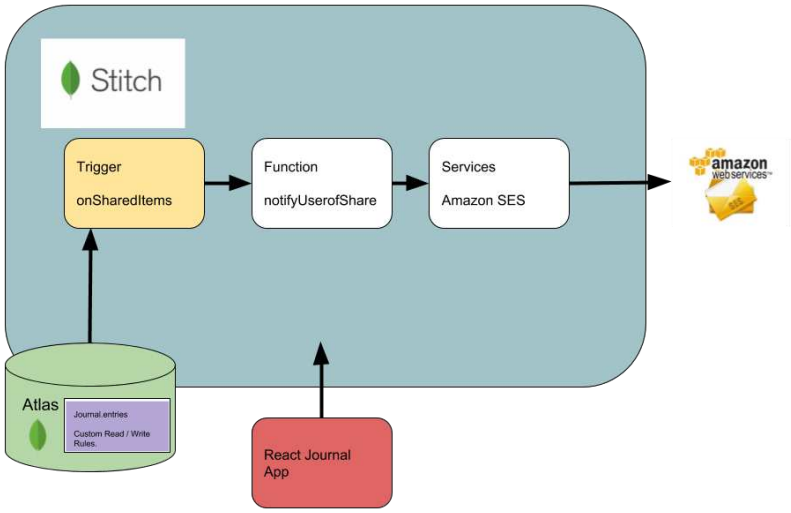
3. React App linked to Stitch using the JavaScript SDK.



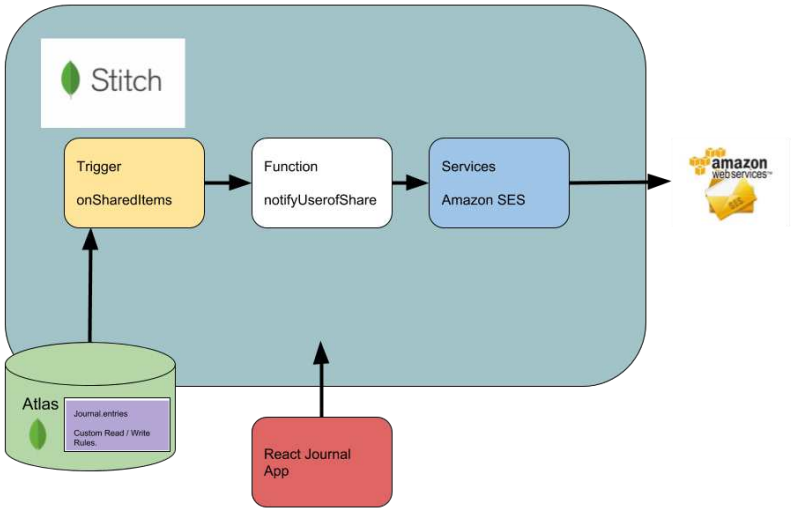
4. Configuring Namespace ACL via Stitch Roles



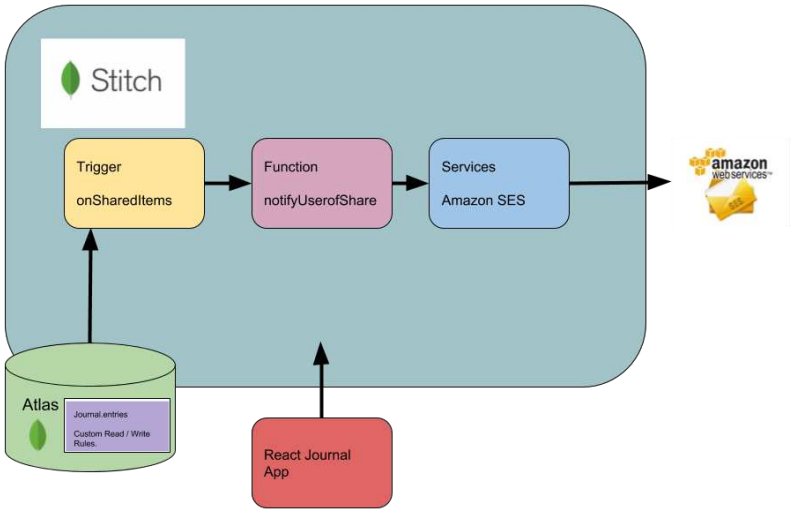
5. Stitch trigger to listen for change events



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



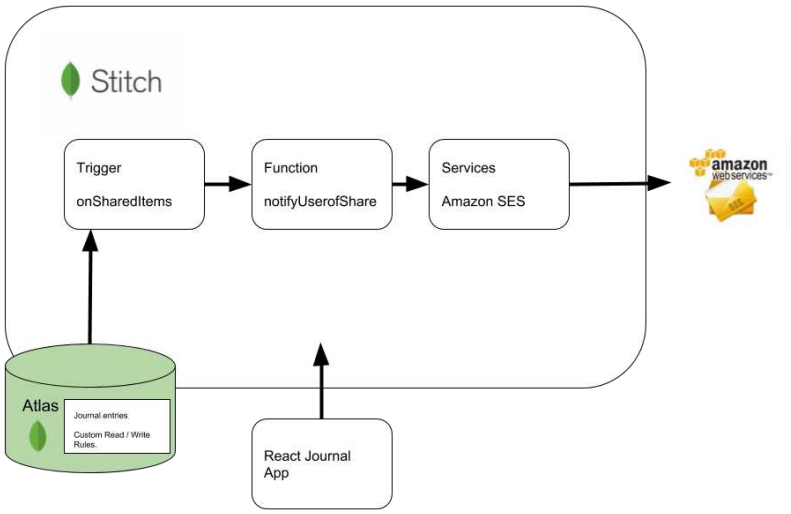
7. Stitch function which will be fired by the trigger.



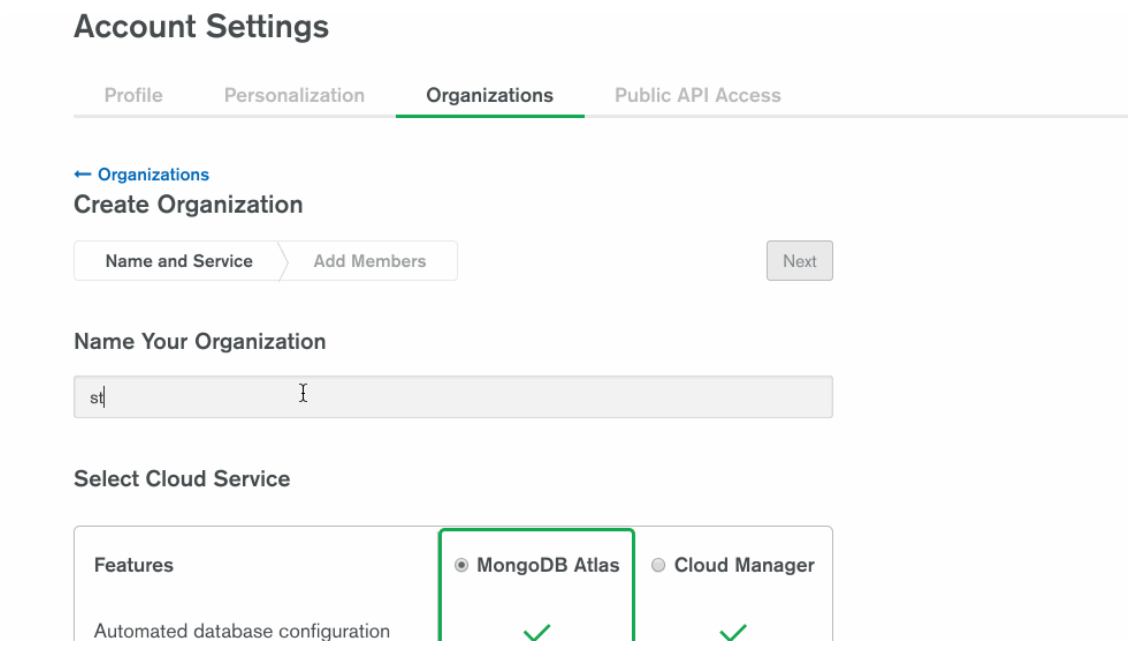
Atlas Free Tier Setup

Create a free tier Atlas cluster.

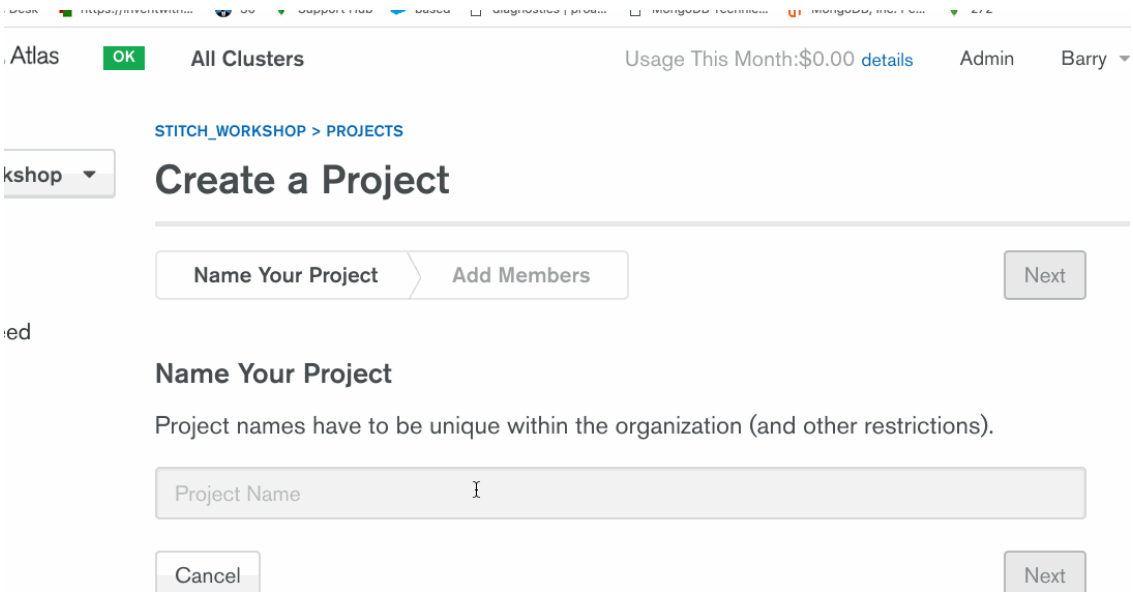
Steps to do this can be found here (https://docs.mongodb.com/manual/tutorial/atlas-free-tier-setup/)



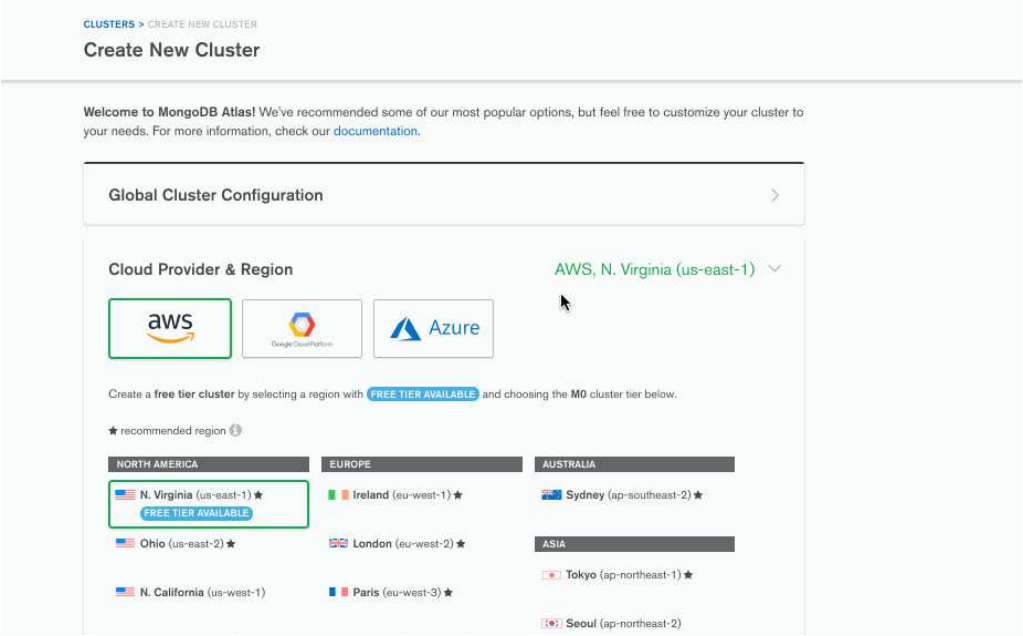
Create an Atlas Organization



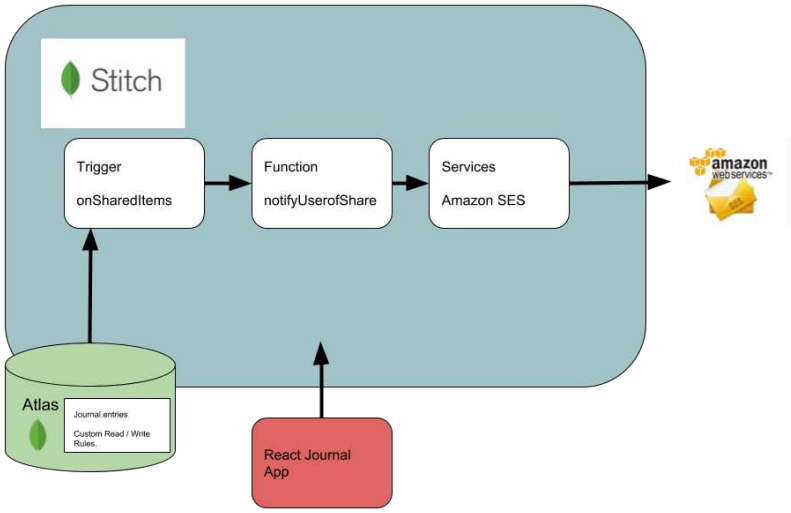
Create an Atlas Project in your Organisation



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.

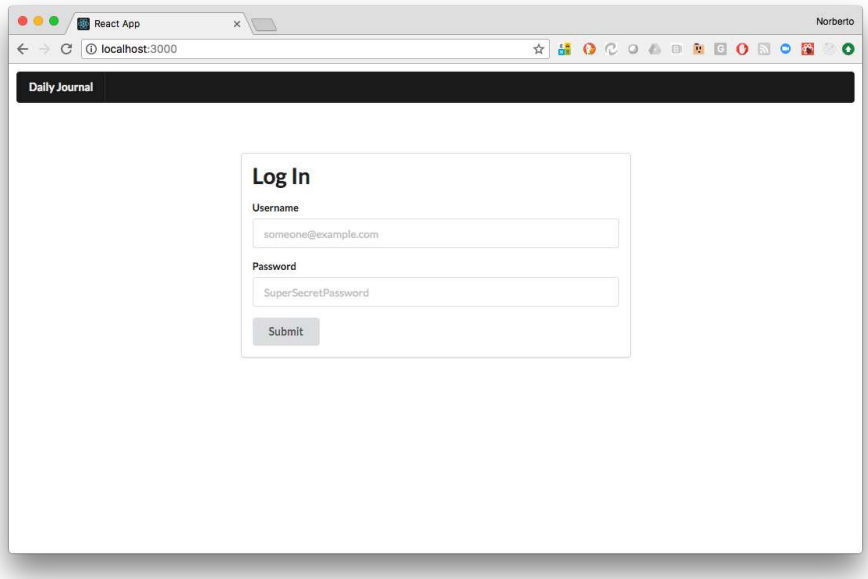
Go ahead and give it a try...

Starting the React App from the shell.

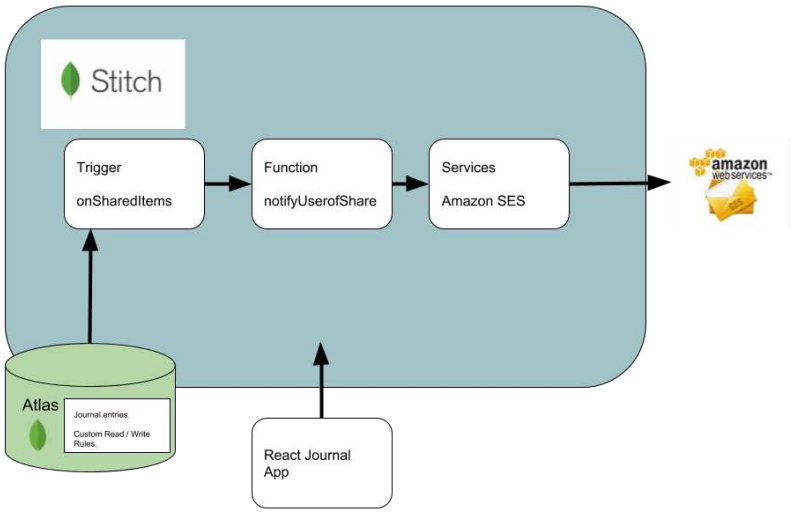
```
cd stitch-workshop-exercise

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:

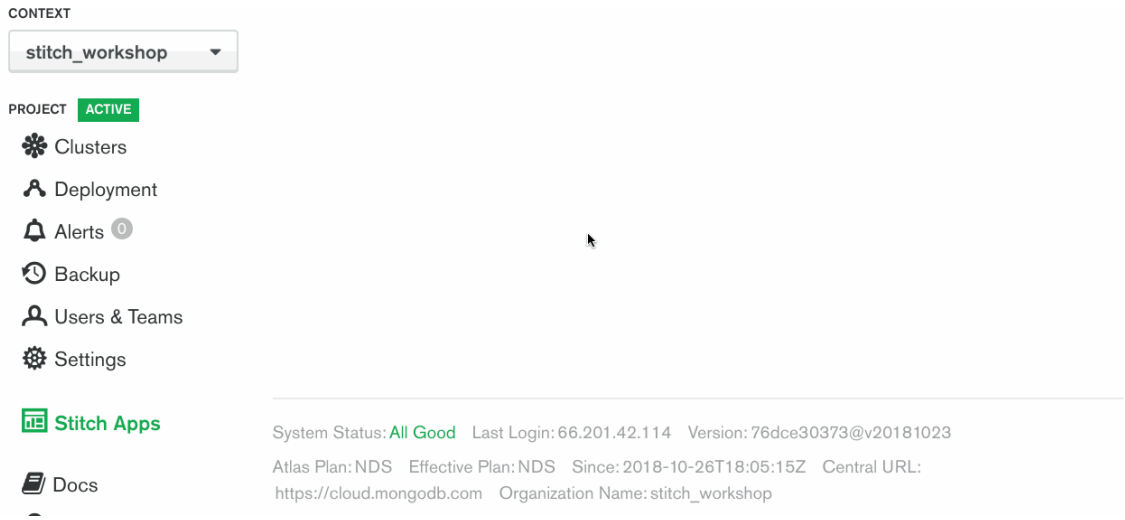
http://localhost:3000 (http://localhost:3000)



Create a Stitch App

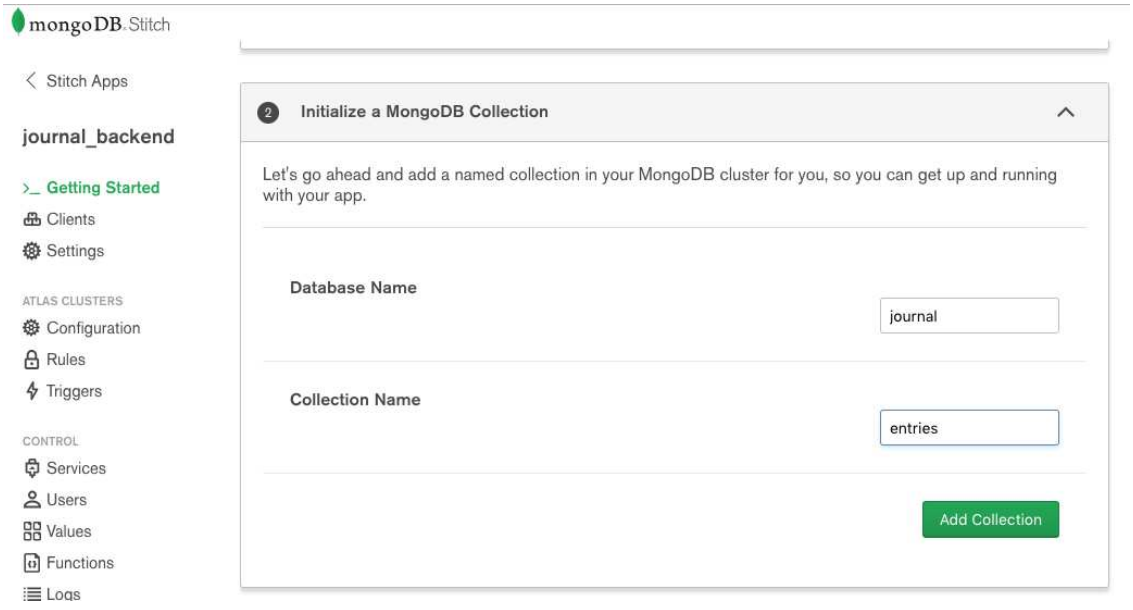


Stitch App UI



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.



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";
```

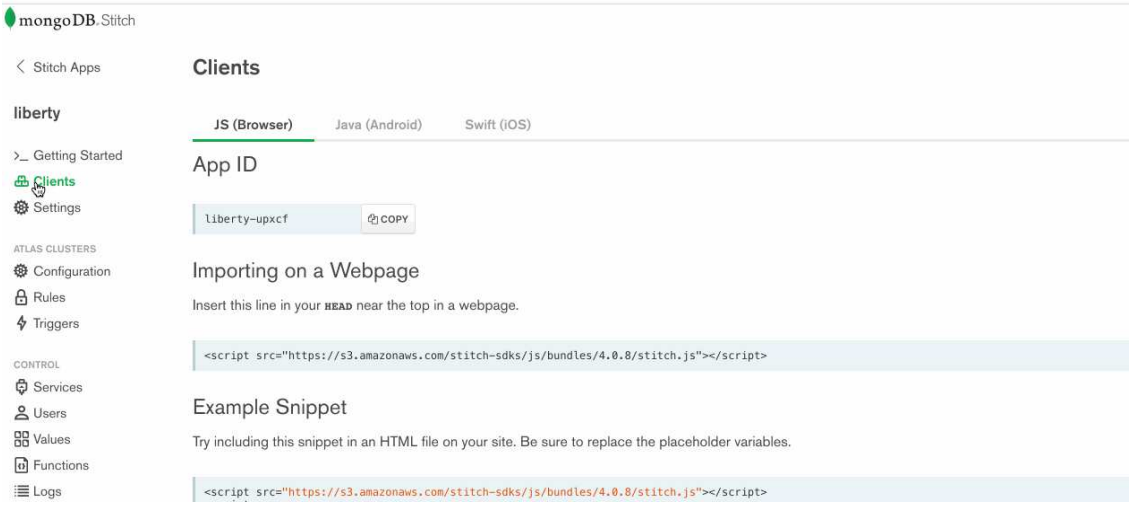
Configuring Stitch `appId`

Replace with the `App ID` of your Stitch App.

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

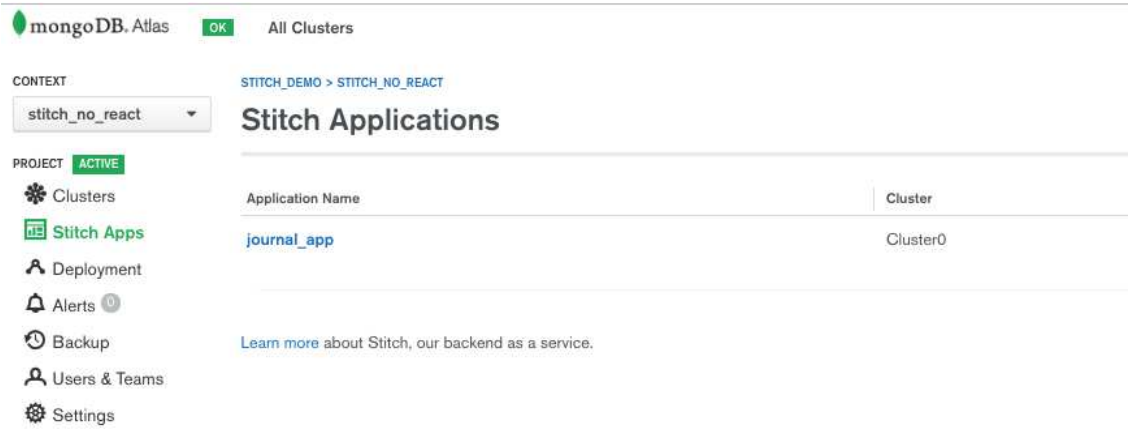
We are going to pass the unique id of the Stitch App that you created earlier.

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!



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



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

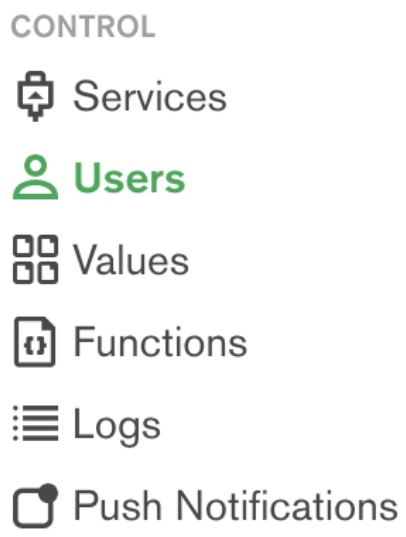
What is an Authentication Provider in Stitch?

- 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

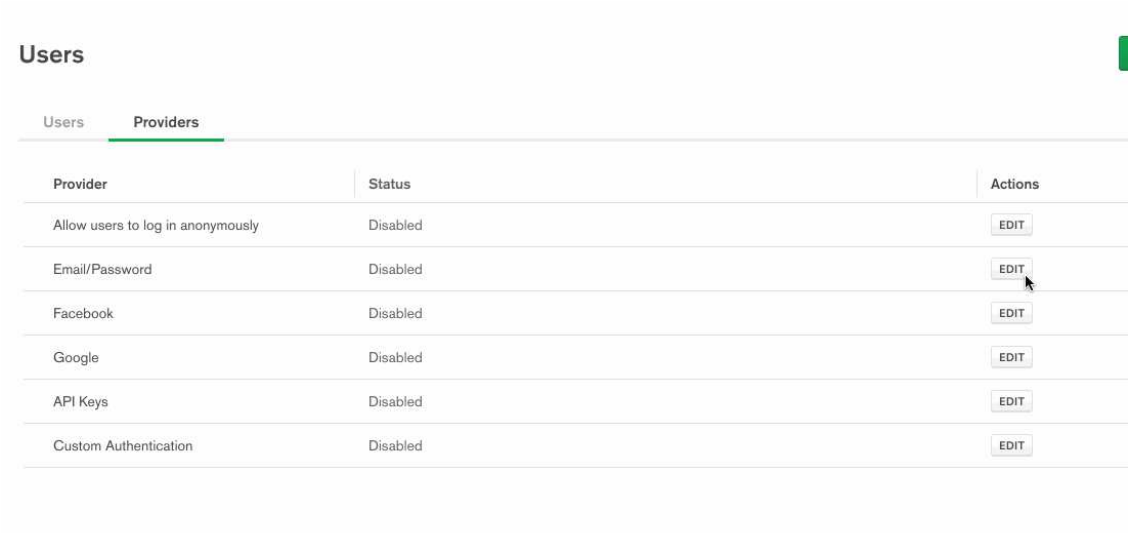
For this app, we will use *Email/Password* authentication

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



Enable Authentication Provider

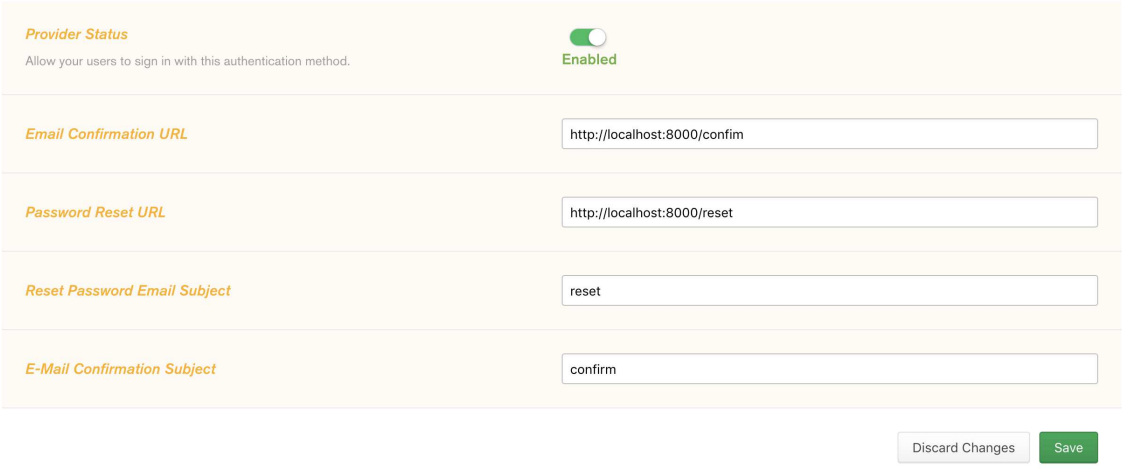
Next, click the "*Providers*" tab in the UI



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



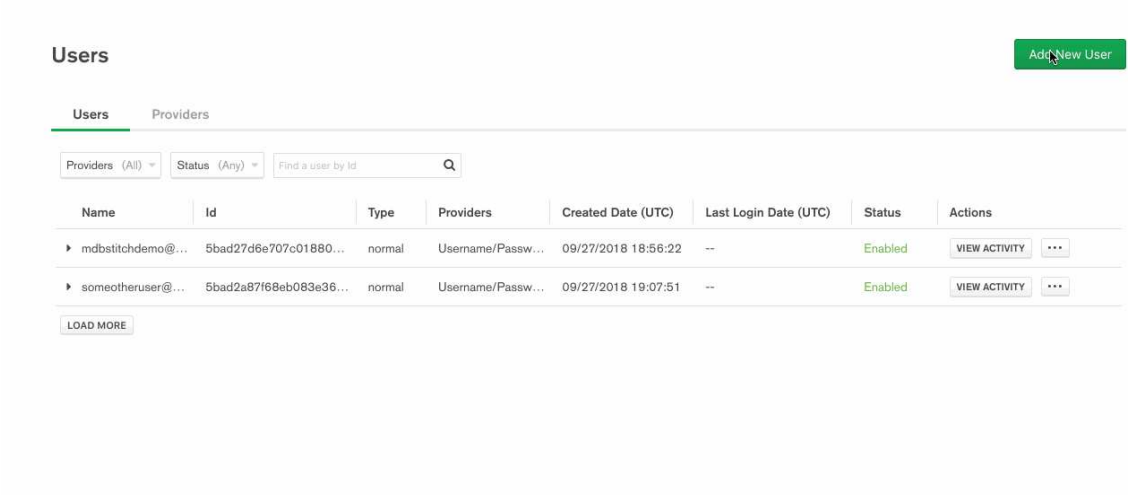
Configure Authentication Provider settings (cont)

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 Pasword Subject: `reset`
- E-Mail Confirmation Subject: `confirm`

Create Application Users

Create two users



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

This is where we will be working next...

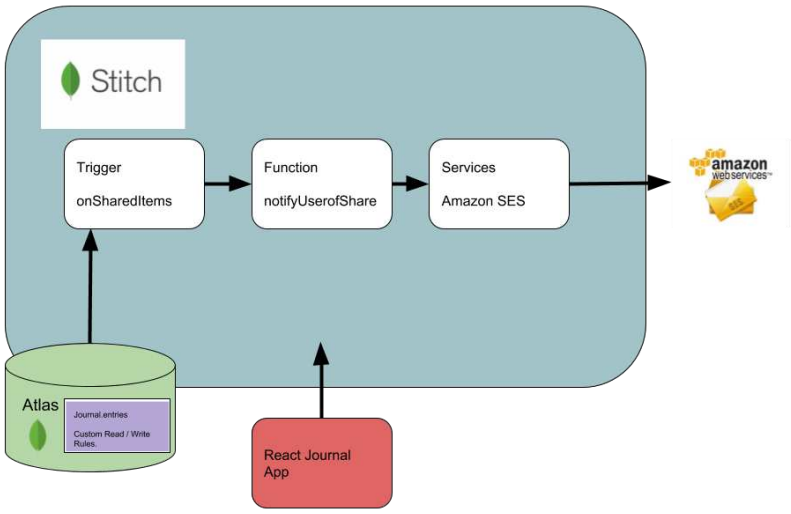
The `Journal` component

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({}).asArray();
```

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...

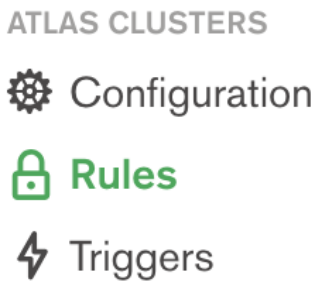
We need to grant additional privileges from the server side!

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

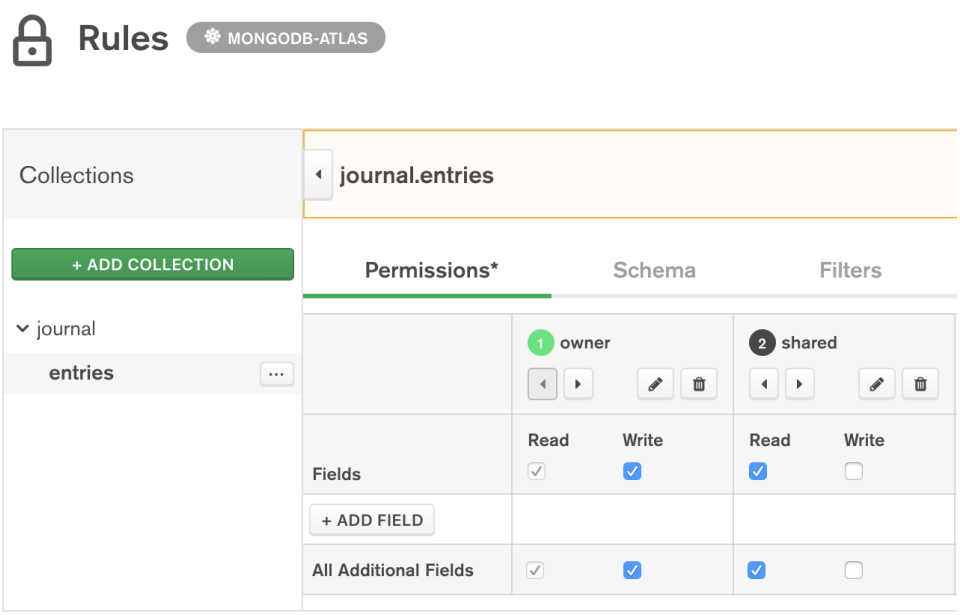


MongoDB Rules allow you to

- 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

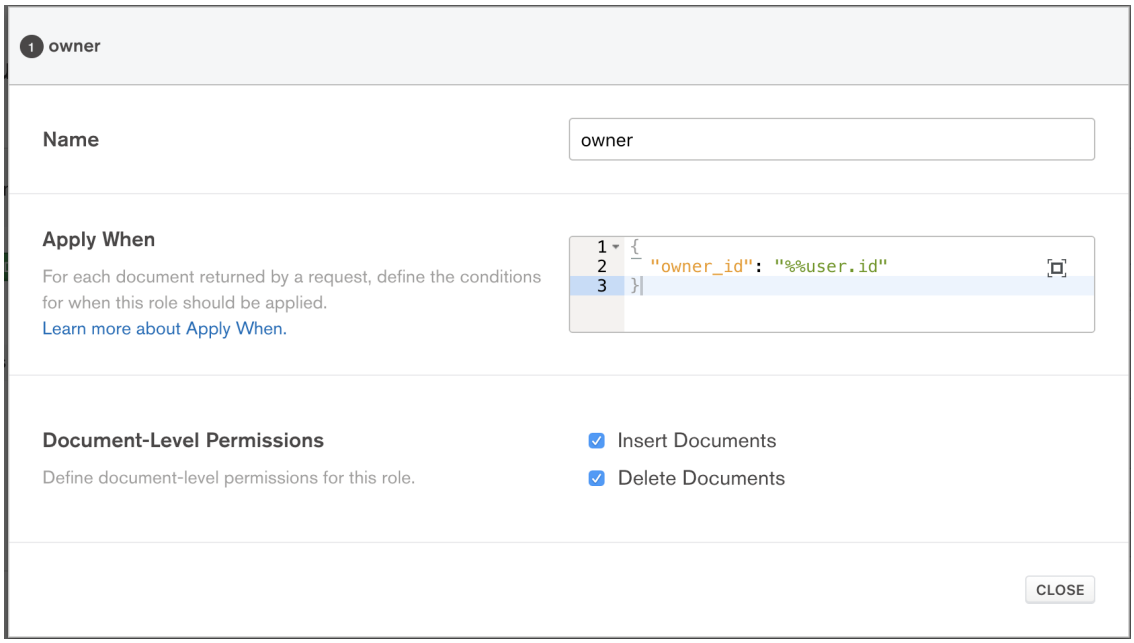
Adding a Stitch role

Our rules should look like this once configured



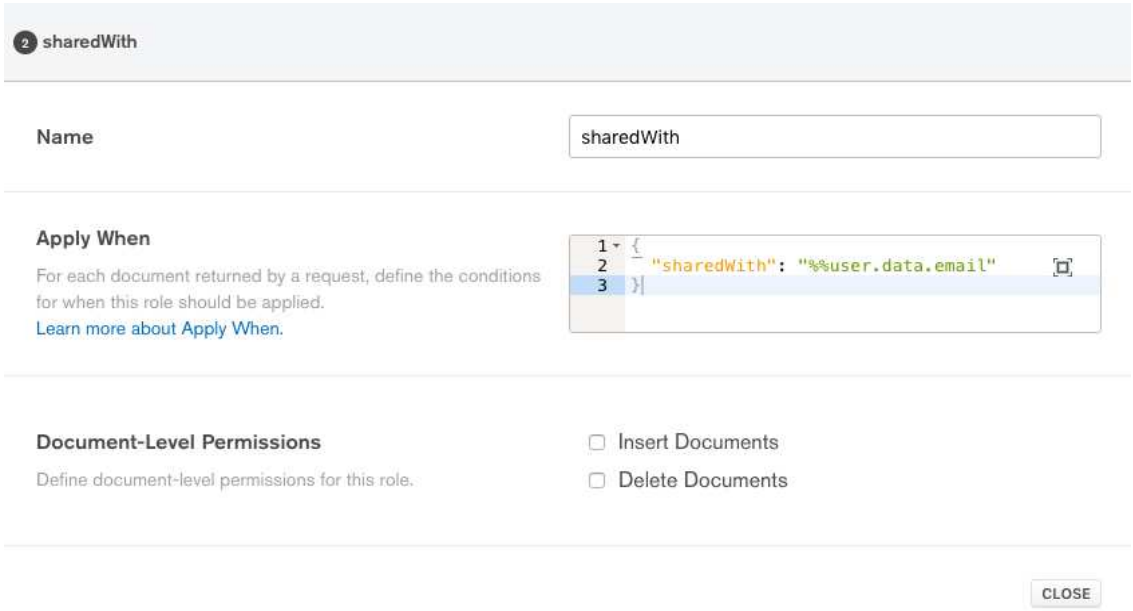
Default `owner` role

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



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:



Create a `sharedWith` Role

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

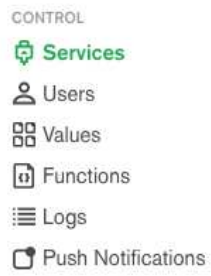
```
{
    "sharedWith": "%%user.data.email"
}
```

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:

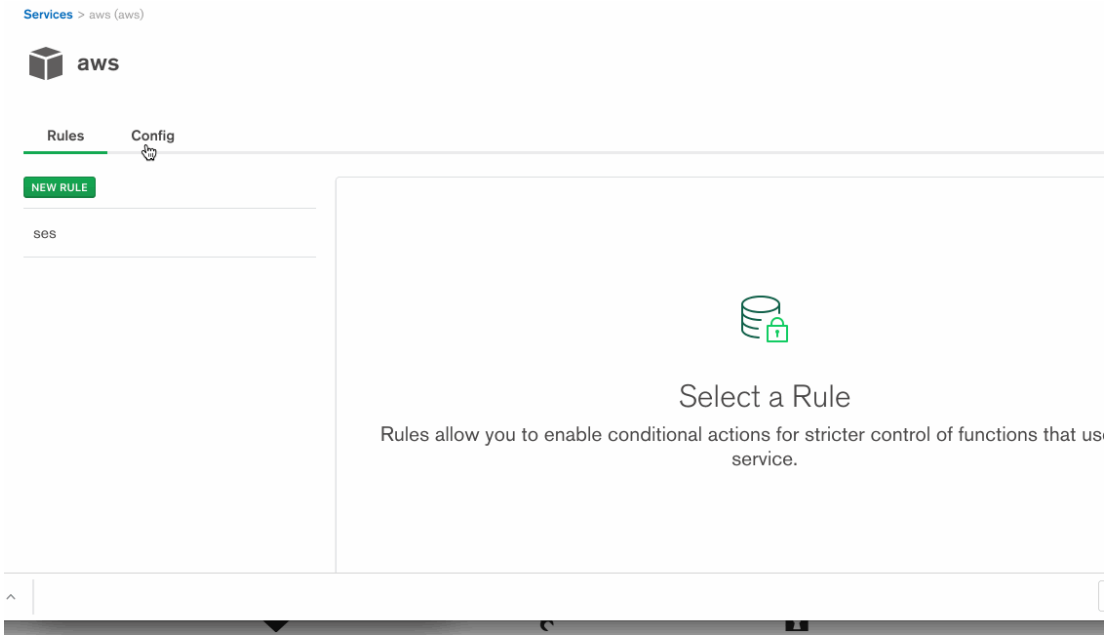
`AKIAI6MY7Z4VLO232BNA`

Secret access key:

`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`

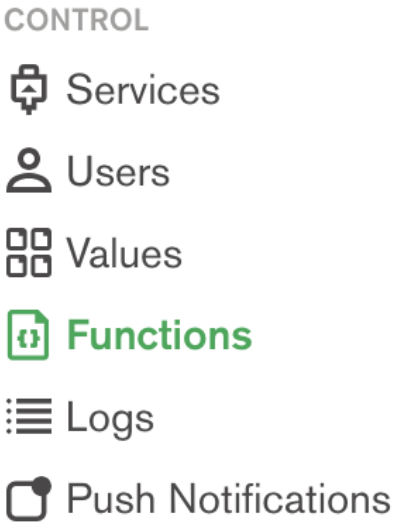
Configure AWS Simple Mail Service (SES)

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

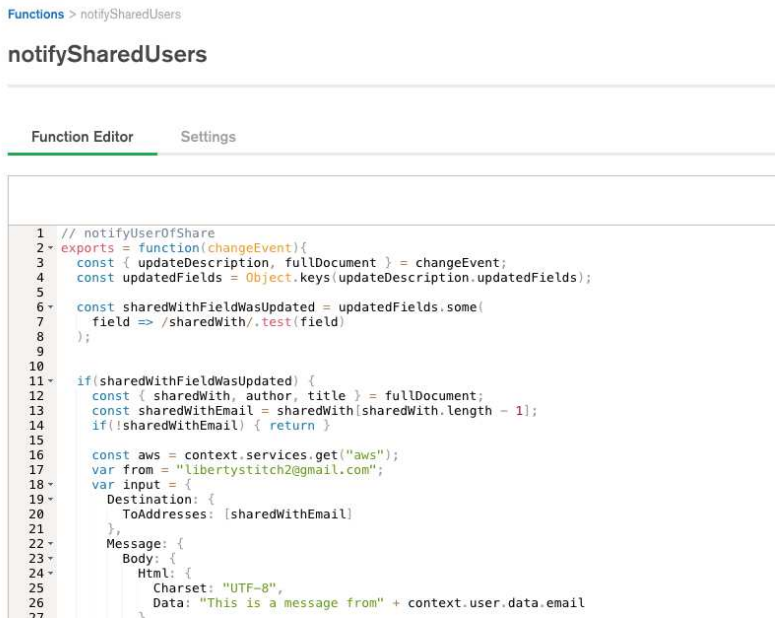


To create the service go to *Services* page and select an **AWS** service. Once created got to the *Rules* tab and click on the *New Rule* button. Name the rule as **ses** and add a *Action* from the *API* dropdown named **ses** with *All Actions* selected There is nothing to provide in the *When* section

Stitch Functions



Stitch function editor



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)
  );
    ...
```

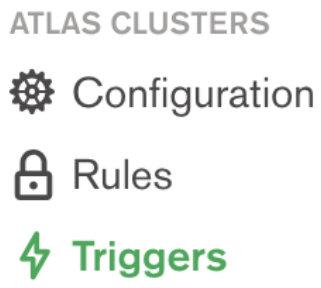
https://gist.github.com/nleite/8862266f693bb87f83570a80710306a8 (https://gist.github.com/nleite/8862266f693bb87f83570a80710306a8)

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");
};
```

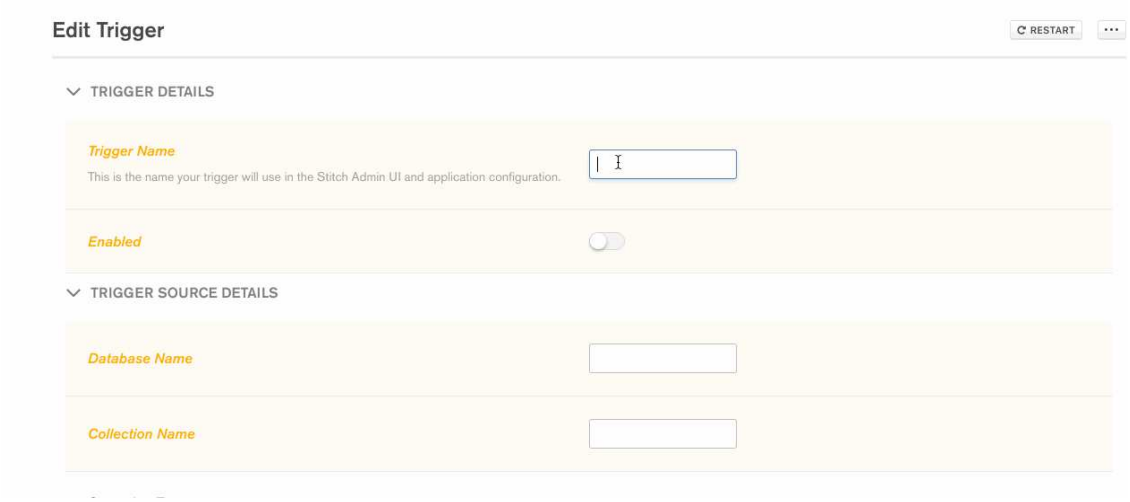
MongoDB Triggers



Triggers

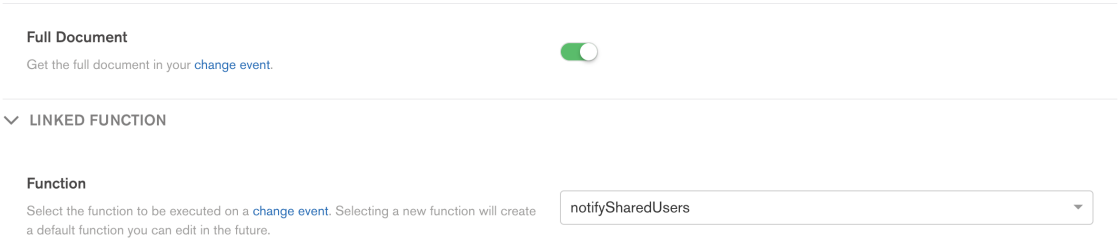
- Fire in Response to Data Changes
  - Built on MongoDB Change Streams
  - Pass Change Events to Functions
  - Use Multiple Triggers per Collection

Creating Trigger in the Stitch UI



To create a trigger we need to perfrom 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 dropbox.

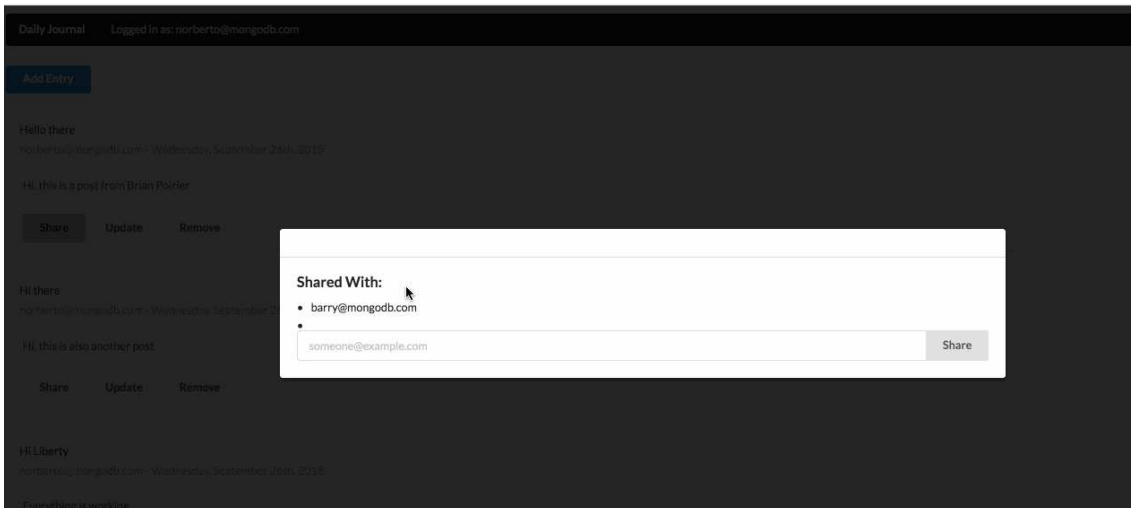


Notify a user when an `Entry` is shared

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

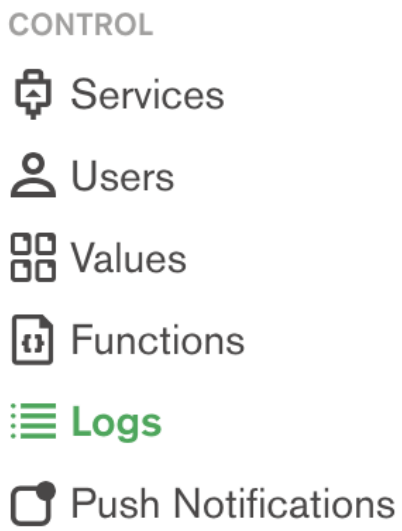
Share an `Entry` with another user

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:



Stitch function log

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

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
@MongoDB