

**GCP – HOL -Session 12**

### **Node.js App**

During this HOL session You will learn how to integrate Datastore and Cloud Storage into a node.js application and use OAuth 2.0 to authenticate users!

### **What will be covered**

* Using the google-cloud library to access Google Cloud Datastore
* Using the google-cloud library to access Google Cloud Storage
* Using the googleapis library to authenticate users via OAuth 2.0
* Using the googleapis library to fetch user profiles from the Google+ API

In Cloud Shell on the command-line, run the following command to clone the Github repository:

**git clone https://github.com/googlecodelabs/cloud-nodejs.git**

Change directory into cloud-nodejs/start.

**cd cloud-nodejs/start**

The sample has the following layout:

app.js */\* Express.js web application \*/*

books.js */\* Code for creating, deleting, and listing books \*/*

auth.js */\* Code for user authentication \*/*

config.js */\* Application configuration variables \*/*

package.json */\* npm package file including dependencies \*/*

views/

index.jade */\* HTML template \*/*

public/

style.css */\* CSS stylesheet \*/*

To run the sample application in Cloud Shell, let's perform the following steps:

1. Install dependencies. Enter the following command:

npm install

2. Run app.js to start the node.js server:

node app.js

3. Click the "Web preview" icon that appears at the top left side of the cloud shell window and select "Preview on port 8080" to see the app in a web browser.

## Set up your project

In this step, you will set up Google Cloud Datastore and configure the application to query from it.

### **Create Credentials**

For the node.js application to access this project's services, eg. Datastore and Cloud Storage, it needs to be authenticated. Create a Service Account for this project which will be used to authenticate the application.

1. In the Cloud Console, navigate to **APIs & Services > Credentials**
2. Make sure your project is selected and click **Continue**
3. Next, click **Create credentials**
4. Select "**Service account key**" from the drop-down menu
5. Under the **Service Account** dropdown, select **"Compute Engine default service account"**and ensure that **Key type**is set to**JSON**. Then click the "**Create"**button
6. Once the JSON key file automatically downloads, locate it (it should have a name of the form) <project name>-<hash string>.json and open it in a text editor.
7. Back in the Cloud Shell window, in the "cloud-nodejs/start" directory, copy and paste the entire contents of the key file into a new file called key.json.

### **Enable Datastore API**

The credentials you created allow your application to communicate with Google APIs that you enable for this project. Enable the Datastore API so the application can access Datastore.

1. Under **APIs & Services**, click **Enable APIs and Services** and search for "Cloud Datastore API"
2. Click on **Cloud Datastore API**
3. Click **Enable** (at the top of the page).

### **Update configuration variables**

To configure the node.js sample application to authenticate with your project, go back to your Cloud Shell instance and edit the config.js file in the "cloud-nodejs/start" directory and replace the placeholder value for projectId with the Project ID of your project:

### [**config.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/config.js)

module.exports = {

projectId: '[your Google Developers Console project ID]',

keyFilename: './key.json',

*// ...*

};

## List all books

The sample application's home page lists all books.

Books are retrieved by calling books.getAllBooks from the books module.

### [**app.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/app.js)

var config = require('./config');

var books = require('./books')(config);

*// ...*

*/\* Fetch all books and display them \*/*

app.get('/', function(req, res, next) {

books.getAllBooks(function(err, books, key) {

if (err) return next(err);

var keyBooks = books.map((book) => Object.assign(book, { id: book.id || book[key].id }));

res.render('index', { books: keyBooks, user: req.session.user });

});

});

The current implementation of getAllBooks found in books.js simply returns a fake book.

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/books.js)

function getAllBooks(callback) {

var error = null;

var books = [

{ id: 12345, title: 'Fake Book', author: 'Fake Author' }

];

callback(error, books);

}

In this step, you will write the code for getAllBooks to query for Book entities from Datastore.

### **Configure Datastore**

To begin, install the google-cloud npm package, which you will use to interact with Cloud Datastore.

npm install google-cloud@^0.57.0 --save

The google-cloud node package is an idiomatic node.js client library for Google Cloud Platform services.

While still in the project directory cloud-nodejs/start edit the books.js file.

Copy the following code block:

var gcloud = require('google-cloud');

var datastore = gcloud.datastore({

projectId: config.projectId,

keyFilename: config.keyFilename

});

Then paste the code block to the books.js file:

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/books.js)

module.exports = function(config) {

*// Add the copied code block here*

function getAllBooks(callback) {

*// ...*

The gcloud.datastore object allows you to interact with Google Cloud Datastore

The projectId and keyFilename are read from the config.js file that you edited earlier.

### **Query for entities**

Now replace the current getAllBooks function with the following:

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-1-retrieve-books/books.js)

function getAllBooks(callback) {

var query = datastore.createQuery(['Book']);

datastore.runQuery(query, (err, books) => callback(err, books, datastore.KEY));

}

Datastore queries are built with datastore.createQuery and run via datastore.runQuery. datastore.KEYis a special value that names a property pointing to the entity's key.

createQuery accepts an array containing the kinds of entities to query and returns a [Query](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.14.0/datastore/query) object.

In our case, we want to query for all 'Book' entities.

runQuery runs a query and returns its results as a list of entities in the following format:

{

title: 'A Tale of Two Cities',

author: 'Charles Dickens',

imageUrl: 'http://books.google.com/books...'

}

### **Create Entity using the Developers Console**

To test the Datastore query and see data displayed, add a Book entity from the [Google Cloud Console](http://console.cloud.google.com/).

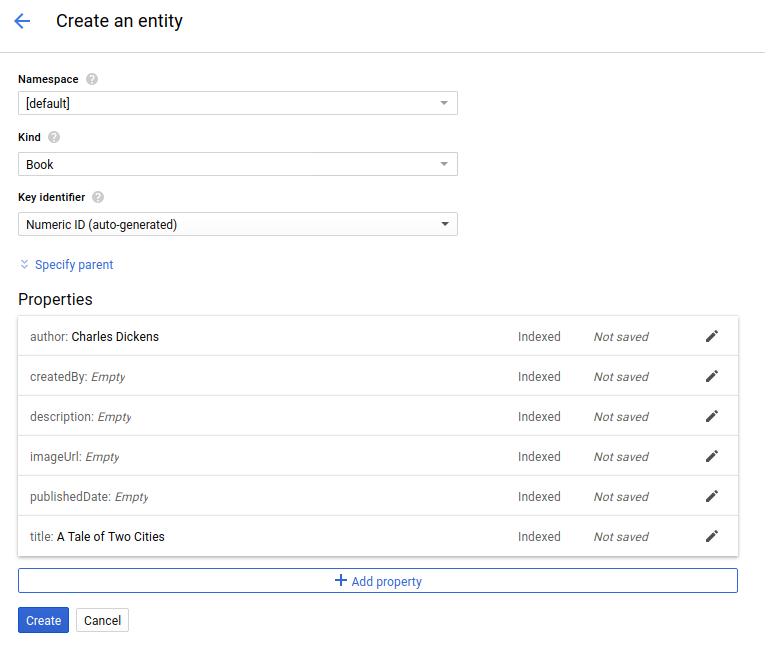
1. Open a new web browser tab and visit Google Cloud Console
2. Select to your Google Cloud Platform project using the dropdown at the top.
3. From the left navigation menu under "Storage", click **Datastore**
4. Click **Create Entity**

Fill out the **Create an entity** form with the following:

1. **Namespace:**Leave this set as [default]
2. **Kind:**Select Book from the dropdown. If Book is not present, type it in manually and add the following properties to the entity. (All of these properties should be Strings.)

* author
* createdBy
* description
* imageUrl
* publishedDate
* title

1. **Key Identifier Type:**leave the default value of "Numeric ID (auto-generated)"
2. Under the property list, change the value of the title property to A Tale of Two Cities
3. Then, change the value of the author property to Charles Dickens
4. Click **Save**

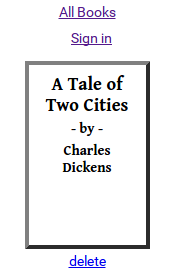


From Cloud Shell, make sure your running node.js web server is stopped. You should have done it in a previous step, but if have not, stop it now by pressing CTRL + C. Now run it again.

node app.js

To view your changes, click the "Web preview" icon that appears at the top left side of the cloud shell window and select "Preview on port 8080" to see the app in a web browser.

Now you should see the book listed!



## Create and delete books

### **Create books**

The sample application includes a form for adding books. When the form is submitted, the application creates the book by calling books.addBook.

### [**app.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/app.js)

*/\* Add a new book \*/*

app.post('/books', function(req, res, next) {

*// ...*

books.addBook(req.body.title, req.body.author, coverImageData, userId, function(err) {

if (err) return next(err);

res.redirect(req.get('Referer') || '/');

})

});

The book title and author come from the 'title' and 'body' fields in the form.

You can ignore coverImageData and userId for now - we'll come back to these later :)

Currently, if you try to add a book, you receive an error:

Error: books.addBook [Not Yet Implemented]

Let's fix that!

### **Creating books**

Press **CTRL + C** in Cloud Shell to exit the running Node.js app. Then, in the project directory, edit the books.js file.

The current addBook function is a placeholder that simply returns an error.

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-2-create-and-delete-books/books.js)

function addBook(title, author, coverImageData, userId, callback) {

if (coverImageData)

return callback(new Error("books.addBook image saving Not Yet Implemented"));

return callback(new Error("books.addBook datastore saving Not Yet Implemented"));

}

To save a new Book entity in Datastore, replace the current addBook function with the following.

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-2-create-and-delete-books/books.js)

function addBook(title, author, coverImageData, userId, callback) {

if (coverImageData)

return callback(new Error("books.addBook image saving Not Yet Implemented"));

var entity = {

key: datastore.key('Book'),

data: {

title: title,

author: author

}

};

datastore.save(entity, callback);

}

datastore.save inserts or updates entity objects.

Entity objects are represented the following format:

{

key: datastore.key('Book', [Numeric entity ID]) },

data: {

title: 'A Tale of Two Cities',

author: 'Charles Dickens'

*// ... properties ...*

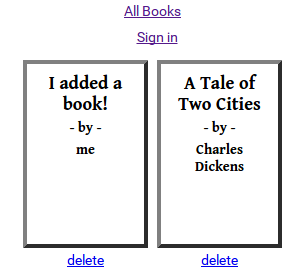
}

}

If an entity ID is provided, calling save will update the entity in Datastore. In our example, the book entity is being added to Datastore for the first time, so no ID is necessary (an ID will be auto-generated by Datastore).

Restart your webserver (node app.js) and try submitting the form again with different values to add a new book.

You should now see your added book!



### **Deleting books**

Now try deleting a book by clicking the delete link below it. You will receive an error:

Error: books.deleteBook [Not Yet Implemented]

Press **CTRL-C** to stop the webserver, and take a look at books.js. The delete link in the sample application deletes books by calling books.deleteBook with the ID of the book to delete.

### [**app.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/app.js)

*/\* Delete book by key \*/*

app.get('/books/delete', function(req, res, next) {

books.deleteBook(req.query.id, function(err) {

if (err) return next(err);

res.redirect('/');

});

});

The sample implementation of books.deleteBook is also a placeholder which simply returns an error:

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-2-create-and-delete-books/books.js)

function deleteBook(bookId, callback) {

callback(new Error("books.deleteBook datastore deletion Not Yet Implemented"));

}

To delete the book with the provided ID, replace the current deleteBook function with the following.

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-2-create-and-delete-books/books.js)

function deleteBook(bookId, callback) {

var key = datastore.key(['Book', parseInt(bookId, 10)]);

datastore.delete(key, callback);

}

Restart the node application and try deleting book again.

## Upload book images

The form for adding books also allows you to attach a cover image for a book.

If you try to create a new book with an attached image, you receive an error:

Error: books.addBook image saving Not Yet Implemented

In this step, you will write the code to save book cover images in Google Cloud Storage.

### **Set up Google Cloud Storage**

Press **CTRL-C** to stop the webserver.

Then, create a Cloud Storage bucket to store book cover images:

1. Open a new web browser tab and visit Google Cloud Console
2. Select your Google Cloud Platform project from the dropdown menu
3. From the left navigation menu, find the Storage section and click **Storage**
4. Click **Create bucket**
5. Choose a unique **bucket name** and click **Create**
6. Note the **bucket name** since you'll need it for configuration

Go back to the tab with Cloud Shell in it. In the project directory, edit the config.js file and replace the placeholder value for bucketName with the name of the bucket you created:

### [**config.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/config.js)

module.exports = {

*// ...*

bucketName: '[your Google Cloud Storage bucket name]',

*// ...*

};

### **Upload Images to Cloud Storage**

When the form is submitted with an image attached, the image data from the cover form field is passed to the books.addBook function:

### [**app.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/app.js)

app.use(multer({ inMemory: true })); *// store upload data in-memory*

*// ...*

*/\* Add a new book \*/*

app.post('/books', function(req, res, next) {

*// ...*

var coverImageData;

if (req.files['cover'])

coverImageData = req.files['cover'].buffer;

*// ...*

books.addBook(req.body.title, req.body.author, coverImageData, userId, function(err) {

if (err) return next(err);

res.redirect(req.get('Referer') || '/');

})

});

[Multer](https://github.com/expressjs/multer) is a node.js middleware for handling multipart/form-data forms for uploading files.

Currently, addBook returns an error when image data is provided:

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-3-book-cover-images/books.js)

function addBook(title, author, coverImageData, userId, callback) {

if (coverImageData)

return callback(new Error("books.addBook image saving Not Yet Implemented"));

*// ...*

}

Let's fix this!

When coverImageData is passed in, we will save the book in Datastore with an imageUrl property specifying the URL to the uploaded book cover image.

To handle this scenario, replace the current addBook function in books.js with the following.

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-3-book-cover-images/books.js)

function addBook(title, author, coverImageData, userId, callback) {

var entity = {

key: datastore.key('Book'),

data: {

title: title,

author: author

}

};

if (coverImageData)

uploadCoverImage(coverImageData, function(err, imageUrl) {

if (err) return callback(err);

entity.data.imageUrl = imageUrl;

datastore.save(entity, callback);

});

else

datastore.save(entity, callback);

}

We have not yet implemented uploadCoverImage so trying to add a book with an image still results in an error:

ReferenceError: uploadCoverImage is not defined

Let's fix this now!

### **Uploading to Google Cloud Storage**

Edit the books.js file and add storage and bucket variables in the location shown below:

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-3-book-cover-images/books.js)

Copy the following code block:

var storage = gcloud.storage({

projectId: config.projectId,

keyFilename: config.keyFilename

});

var bucket = storage.bucket(config.bucketName);

Then, paste it into books.js as shown below:

module.exports = function(config) {

var gcloud = require('google-cloud');

var datastore = gcloud.datastore({

projectId: config.projectId,

keyFilename: config.keyFilename

});

*// Add your copied code here*

*// ...*

bucketName is read from the config.js file that you edited earlier.

The [bucket](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.14.0/storage/bucket) object provides the API you will use to interact with your Google Cloud Storage bucket.

To upload the image to Cloud Storage and return a publicly accessible URL for displaying the image, add the following uploadCoverImage function to books.js:

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-3-book-cover-images/books.js)

function uploadCoverImage(coverImageData, callback) {

*// Generate a unique filename for this image*

var filename = '' + new Date().getTime() + "-" + Math.random();

var file = bucket.file(filename);

var imageUrl = 'https://' + config.bucketName + '.storage.googleapis.com/' + filename;

var stream = file.createWriteStream();

stream.on('error', callback);

stream.on('finish', function() {

*// Set this file to be publicly readable*

file.makePublic(function(err) {

if (err) return callback(err);

callback(null, imageUrl);

});

});

stream.end(coverImageData);

}

[bucket.file](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.34.0/storage/bucket?method=file) returns a [File](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.34.0/storage/file) object providing the API for a file in Cloud Storage.

The publicly accessible URL for the image file will be https://<bucket name>.storage.googleapis.com/<filename>

Restart the node application and try adding a book with an image again.

It should work!

### **Deleting images**

Right now, if you delete a book that has a cover image, the image will remain in your Cloud Storage bucket.

Let's fix this by updating deleteBook to also delete saved cover images.

Replace the current deleteBook function with the following:

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-3-book-cover-images/books.js)

function deleteBook(bookId, callback) {

var key = datastore.key(['Book', parseInt(bookId, 10)]);

datastore.get(key, function(err, book) {

if (err) return callback(err);

if (book.imageUrl) {

var filename = url.parse(book.imageUrl).path.replace('/', '');

var file = bucket.file(filename);

file.delete(function(err) {

if (err) return callback(err);

datastore.delete(key, callback);

});

} else {

datastore.delete(key, callback);

}

});

}

Before deleting a book, the book is retrieved from Datastore by ID via [datastore.get](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.34.0/datastore?method=get) to determine whether or not it has a cover image.

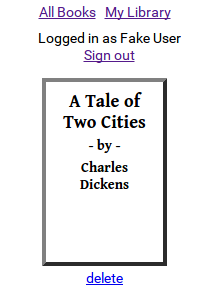
If the book has a cover image, the file is deleted from Cloud Storage via [file.delete](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.34.0/storage/file?method=delete) before deleting the entity from Datastore.

Now deleting books with cover images will also delete the images from your bucket!

## Authenticate users

The sample application includes a ‘Sign in' link so that users can login and see a list of only the books they have added.

Currently, clicking ‘Sign in' logs you in as "Fake User."



Let's fix this!

When a user logs in, the sample application should redirect the user to an authentication URL - in this case, the Google login screen.

Next, you will associate users with books

Once the user is authenticated, they will be redirected back to your application with an authorization code you can use to get their profile information.

The application gets the authentication URL to redirect to by calling auth.getAuthenticationUrl from auth.js.

### [**app.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/app.js)

*/\* Redirect user to OAuth 2.0 login URL \*/*

app.get('/login', function(req, res) {

var authenticationUrl = auth.getAuthenticationUrl();

res.redirect(authenticationUrl);

});

Right now, getAuthenticationUrl is a placeholder that simply redirects to the callback URL.

### [**auth.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-4-user-login/auth.js)

function getAuthenticationUrl() {

return "/oauth2callback";

}

Let's fix this by setting up Google OAuth 2.0 authentication and redirecting the user to the real Google sign in screen.

### **Setup OAuth 2.0 client**

First, you need to create a web application client for authentication.

Then, copy the URL that you've been using to access the app from the browser. This should be of the form

https:*//8080-dot-<9 digit number>-dot-devshell.appspot.com/*

Use the following steps to configure your OAuth 2.0 setup:

1. Visit Google Cloud Console. If necessary, select the project you created earlier from the **Choose Project**dropdown.
2. From the left navigation, click **APIs & Services > Credentials**.
3. Click **Create Credentials > OAuth client ID**.
4. Click the **Configure consent screen** button.
5. Choose an **email address**.
6. Configure a **Product name** which identifies your application to users when they sign in
7. Click **Save**.
8. Choose **Web application**for the Application type.
9. Paste the URL you copied at the start of this section into **Authorized redirect URIs**and append /oauth2callback, for example:  
   https://8080-dot-<9 digit number>-dot-devshell.appspot.com/oauth2callback
10. Click **Create**.
11. Make note of the **Client ID** and **Client secret** that are displayed. You'll need them for configuration later.

### **Update configuration**

The node.js application needs to be configured to use the Client ID you created.

In the project directory, edit the config.js file and do the following:

1. Replace the placeholder values for clientId and clientSecret
2. Replace <http://localhost:8080/oauth2callback> with the URL that you copied at the start of the **Setup OAuth 2.0 client** section. Don't forget to include the /oauth2callback.

### [**config.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/config.js)

module.exports = {

*// ...*

oauth2: {

clientId: '[Client ID for web application credentials]',

clientSecret: '[Client Secret for web application credentials]',

redirectUrl: process.env.REDIRECT\_URL || 'http://localhost:8080/oauth2callback'

}

};

### **Redirect to sign in screen**

To begin, install the [googleapis](https://www.npmjs.com/package/googleapis) npm package, which you will use to generate the authentication URL and fetch profile information for the logged in user.

npm install googleapis@^30.0.0 --save

The [googleapis](https://github.com/google/google-api-nodejs-client/) node package is Google's officially supported node.js client library for using Google APIs.

In the project directory, edit the auth.js file and add the following code to require the googleapis package:

### [**auth.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-4-user-login/auth.js)

module.exports = function(config) {

*// TODO: define this variable below*

var {google} = require('googleapis');

function getAuthenticationUrl() {

*// ...*

Now, replace the current getAuthenticationUrl function with the following:

### [**auth.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-4-user-login/auth.js)

function getAuthenticationUrl() {

var client = new google.auth.OAuth2(

config.oauth2.clientId,

config.oauth2.clientSecret,

config.oauth2.redirectUrl

);

*// Use 'profile' scope to authorize fetching the user's profile*

return client.generateAuthUrl({ scope: ['profile'] });

}

The clientId and clientSecret are read from the config.js file that you edited earlier.

Restart the node application and click ‘Sign in'. You should be redirected to the Google sign in page.

If you get a redirect\_uri\_mismatch error, you may need to wait a minute for the Web application client ID to become active.

If you login, you will be redirected back to the sample application, but you will still appear to be signed in as "Fake User" because the callback is not implemented to fetch the user profile.

Let's fix that!

### **Fetch user profile**

Enable the Google+ API so the application can call the API to fetch user profiles.

1. Under [**APIs & Services**](https://console.cloud.google.com/project/_/apiui/api)[**> Library > Social APIs**](https://console.developers.google.com/project/_/apiui/api)search for the "Google+ API"
2. Click on **Google+ API**
3. Click **Enable**

To fetch the profile of the authenticated user, the /oauth2callback route that Google OAuth 2.0 redirects to calls auth.getUser from auth.js, passing it the provided ?code query string that can be used to fetch the user's profile.

The returned user profile is stored in the application session (via cookies).

### [**app.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/app.js)

var session = require('cookie-session');

app.use(session({ signed: true, secret: config.cookieSecret }));

*// ...*

*/\* Use OAuth 2.0 authorization code to fetch user's profile \*/*

app.get('/oauth2callback', function(req, res, next) {

auth.getUser(req.query.code, function(err, user) {

if (err) return next(err);

req.session.user = user;

res.redirect('/');

});

});

[cookie-session](https://github.com/expressjs/cookie-session) is a node.js middleware package for providing signed cookie-based sessions.

Cookies are signed with the value of config.cookieSecret to protect against forgery.

Currently, auth.getUser always returns "Fake User."

### [**auth.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-4-user-login/auth.js)

function getUser(authorizationCode, callback) {

var error = null;

var fakeUser = { id: 123, name: 'Fake User' };

callback(error, fakeUser);

}

Let's fix this!

Replace the current getUser function with the following:

### [**auth.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-4-user-login/auth.js)

function getUser(authorizationCode, callback) {

var client = new google.auth.OAuth2(

config.oauth2.clientId,

config.oauth2.clientSecret,

config.oauth2.redirectUrl

);

*// With the code returned from OAuth flow, get an access token*

client.getToken(authorizationCode, function(err, tokens) {

if (err) return callback(err);

*// Configure this Google API client to use the access token*

client.setCredentials(tokens);

*// Call the Google+ API to get the profile of the user who authenticated*

google.plus('v1').people.get({ userId: 'me', auth: client }, function(err, profile) {

if (err) return callback(err);

var user = {

id: profile.data.id,

name: profile.data.displayName,

imageUrl: profile.data.image.url

};

callback(null, user);

});

});

}

This code gets an access token for the user who authenticated and then calls the [People.get](https://developers.google.com/+/api/latest/people/get) method of the [Google+ API](https://developers.google.com/+/api/) using the user's access token.

The sample application expects a user object in the following format:

{

id: <unique ID of user>,

name: <display name>,

imageUrl: <publicly accessible URL to user profile image>

}

Restart the node application the try and Sign in again (you may need to Sign out first).

You should see your name and profile image displayed!

## Display a user's books

When a user is signed in, the sample application passes the ID of the signed in user to books.addBook when new books are created.

### [**app.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/app.js)

*/\* Add a new book \*/*

app.post('/books', function(req, res, next) {

*// ...*

var userId;

if (req.session.user)

userId = req.session.user.id;

books.addBook(req.body.title, req.body.author, coverImageData, userId, function(err) {

if (err) return next(err);

res.redirect(req.get('Referer') || '/');

})

});

Currently, we're not tracking this userId when saving book entities.

To fix this, update the entity object in the books.addBook function of books.js to include the userId when provided:

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-5-user-books/books.js)

if (userId)

entity.data.userId = userId;

function addBook(title, author, coverImageData, userId, callback) {

var entity = {

key: datastore.key('Book'),

data: {

title: title,

author: author,

}

};

*// Add the code here*

if (coverImageData)

*// ...*

Now, books created by authenticated users will be associated with the books they add. This means we can query Datastore to list only the books added by that user.

The sample application displays a **My Library** link when a user is signed in. When clicked, only books owned by the user are displayed.

A user's books are fetched by calling books.getUserBooks in books.js.

### [**app.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/start/app.js)

*/\* Fetch books created by the currently logged in user and display them \*/*

app.get('/mine', function(req, res, next) {

if (! req.session.user) return res.redirect('/');

books.getUsersBooks(req.session.user.id, function(err, books) {

if (err) return next(err);

res.render('index', { books: books, user: req.session.user });

});

});

Sign into the application and click on **My Library**. You will receive an error:

Error: books.getUsersBooks [Not Yet Implemented]

Let's fix that!

Currently, the getUserBooks function in books.js is a placeholder that simply returns an error:

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-5-user-books/books.js)

function getUserBooks(userId, callback) {

callback(new Error('books.getUserBooks [Not Yet Implemented]'));

}

In the project directory, edit the books.js file and replace the getUserBooks function with the following:

### [**books.js**](https://github.com/googlecodelabs/cloud-nodejs/blob/master/step-5-user-books/books.js)

function getUserBooks(userId, callback) {

var query = datastore.createQuery(['Book']).filter('userId', '=', userId);

datastore.runQuery(query, (err, books) => callback(err, books, datastore.KEY));

}

To refine queries, the Datastore [Query](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.34.0/datastore/query) object provides functions such as [filter](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.34.0/datastore/query?method=filter), [groupBy](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.34.0/datastore/query?method=groupBy), [limit](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.34.0/datastore/query?method=limit), [offset](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.34.0/datastore/query?method=offset), and [hasAncestor](http://googlecloudplatform.github.io/gcloud-node/#/docs/v0.34.0/datastore/query?method=hasAncestor).

Restart the node application, sign in, and click on the **My Library** link. This should now list only the books you've added!

If you haven't added any books, you will see "There are no books!". In this case, sign in (if you haven't already), add a book, and then click the ‘My Library' link again to view the list of only books you have added.