Music Hub

IT6038 Web Services

Dan Hayworth

92060016

Contents

[API Planning 2](#_Toc51091761)

[Benefits and uses: 2](#_Toc51091762)

[Methods and actions: 2](#_Toc51091763)

[API Design 2](#_Toc51091764)

[API Modelling: 2](#_Toc51091765)

[API Implementation 3](#_Toc51091766)

[Music Hub API structure: 3](#_Toc51091767)

[MongoDB: 3](#_Toc51091768)

[API folder structure: 4](#_Toc51091769)

[User Authentication 6](#_Toc51091770)

[User model: 6](#_Toc51091771)

[JWT authentication: 9](#_Toc51091772)

[API Testing 9](#_Toc51091773)

[Routes testing: 9](#_Toc51091774)

[Authentication tests: 9](#_Toc51091775)

[10 CRUD tests: 9](#_Toc51091776)

# API Planning

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Endpoint | Usage | Returns |
| GET | /hub | Get all artists and songs | Artist + Songs |
| GET | /hub/songs | Get song list | Songs |
| GET | /hub/artists | Get artists list | Artists |
| GET | /hub/song/:id | Get a song | Song |
| GET | /hub/artist/:id | Get an artist | Artist |
| GET | /hub/user/:id | Get the user | User |
| POST | /hub/song/:id/update | Update a song | Song |
| POST | /hub/artist/:id/update | Update an artist | Artist |
| POST | /hub/song/:id/delete | Delete a song | Song deleted |
| POST | /hub/artist/:id/delete | Delete an artist | Artist deleted |

## Benefits and uses:

Based on simple REST principles, this Music Hub API would benefit singers and music companies that would like to add new songs and/or remove them from our Music hub Database.

Creating this API will allow people to stay always updated with new songs if those were to be added into the database.

## Methods and actions:

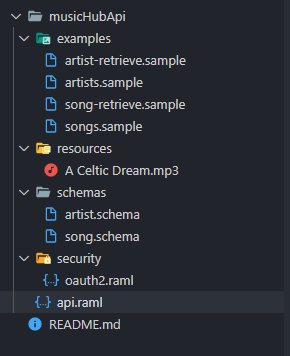
Methods and actions available to this API are based on REST principles with its resources being accessed through HTTP requests.

|  |  |
| --- | --- |
| Method | Action |
| GET | Retrieves the information |
| POST | Creates a new resource |
| UPDATE | Changes or replaces resources |
| DELETE | Removes resources |

# API Design

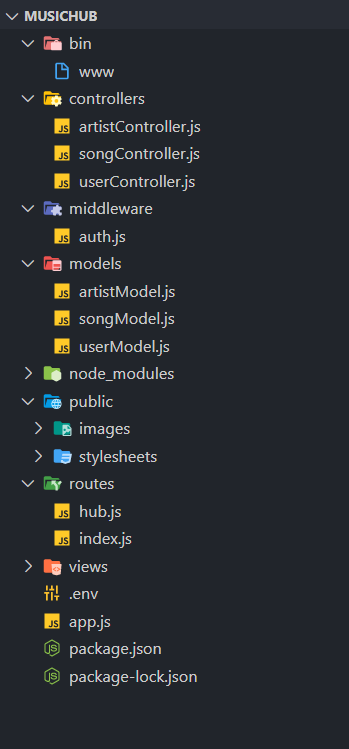
## API Modelling:

For this API we have used RAML API modelling language with Atom and API Workbench. The files are attached to this submission in the folder musicHubApi/api.raml



# API Implementation

## Music Hub API structure:

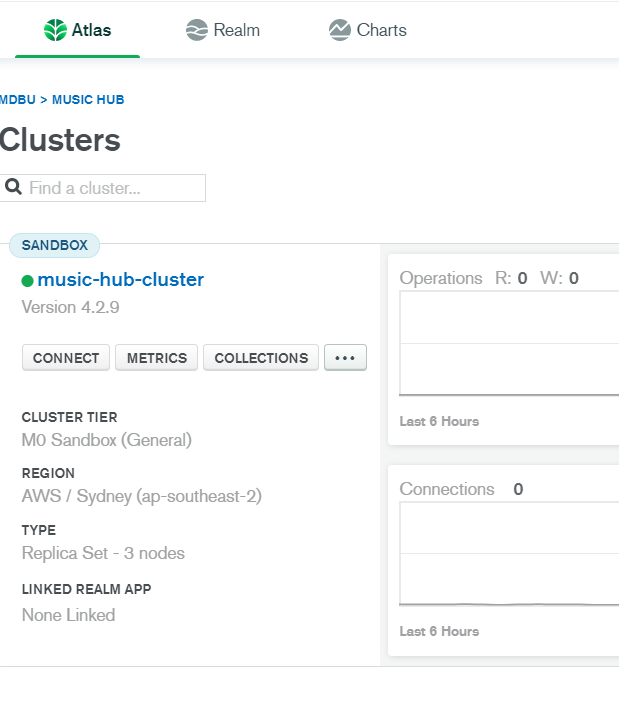
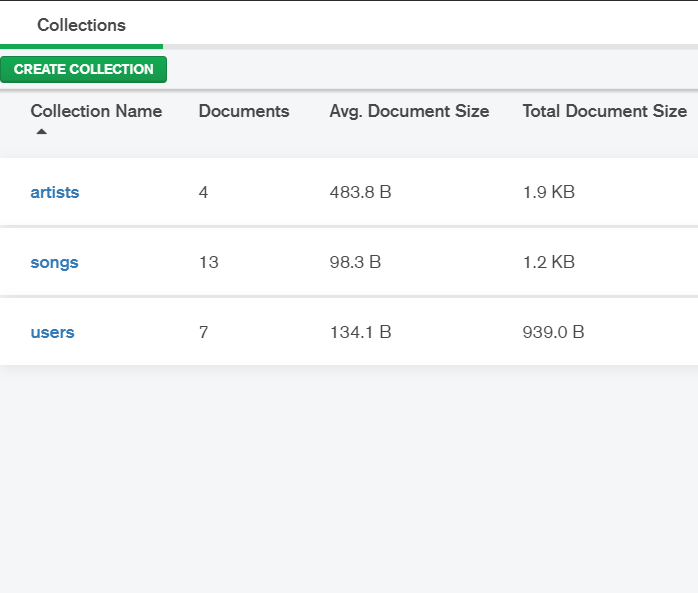
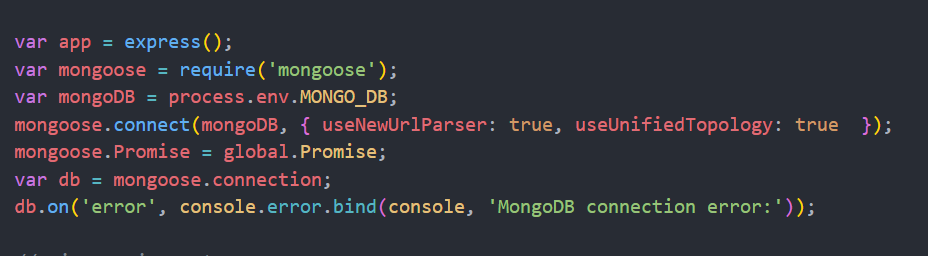


I have created a Node JS project with Express and install the required Node packages to be able to achieve the outcome that is required. Some of these packages are;

* Bcrypt it will be used to has the password adding an extra layer of security.
* Dotenv will be used to load the variables from .env package into process.env to be able to hide the sensitive information
* Jsonwebtoken JWT is used to create tokens between the parties for security purpose.
* Mongoose it will be installed to help us communicate with Mongo DB
* Helmet I will be using it to secure the Express project by setting various HTTP headers.

## MongoDB:

I have created a Mongo Database in the cloud using Atlas and with mongoose packet I am connecting the API to our database.

## API folder structure:

For our purpose I have created 2 controller files and 2 model files; one controller and one model for songs and one controller and one model for artists.

I have created a new file for routes and have redirected the index page of Express to our own routes in hub.js.

Artist Controller has the following functions:

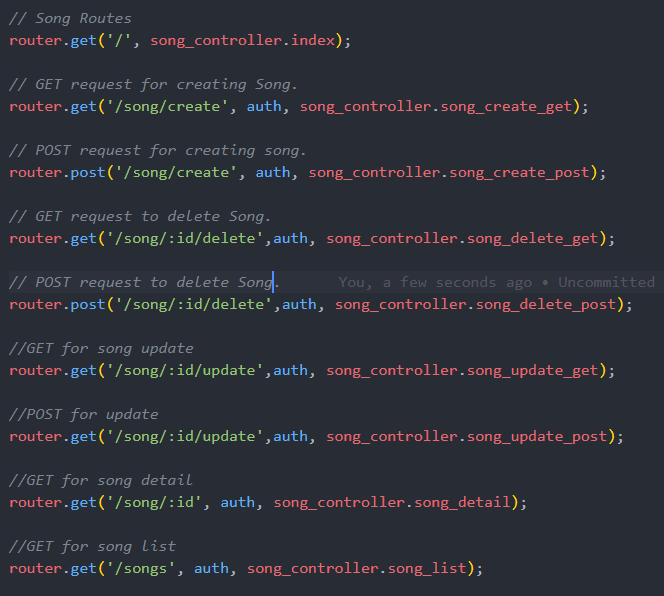
* artist\_list – to return a list of artists
* artist\_detail – to return details about an artist
* artist\_create\_get – to render the form for creating an artist
* artist\_create\_post – insert newly created artist into the database
* artist\_delete\_get – to retrieve details about an artist that is about to be deleted
* artist\_delete\_post - which gets the artist that has been retrieved and deletes from database if there are no songs related with it.
* artist\_update\_get - gets the information about an artist and adds it to the updating form
* artist\_update\_post – updates the artist in the database.

Song Controller has the following functions:

* index – that return a list of all artists and songs
* song\_list – returns a list of songs
* song\_detail – return details about a specific song
* song\_create\_get – returns the song creation form
* song\_create\_post – adds the song to the database
* song\_delete\_get – gets details about specific song to be deleted
* song\_delete\_post – checks the details and removes the song from database
* song\_update\_get – gets details about a song based on id and autofill the form to be able to make changes
* song\_update\_post – saves changes to the song into the database

For each of the functions I have create a HTTP request.

Song Routes



Artist Routes:



# User Authentication

## User model:

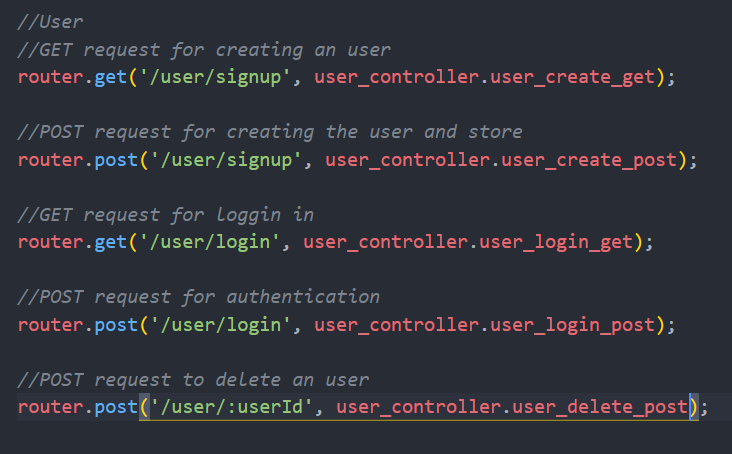
I have created a user model that has the following fields as requested:



User Controller has the following functions:

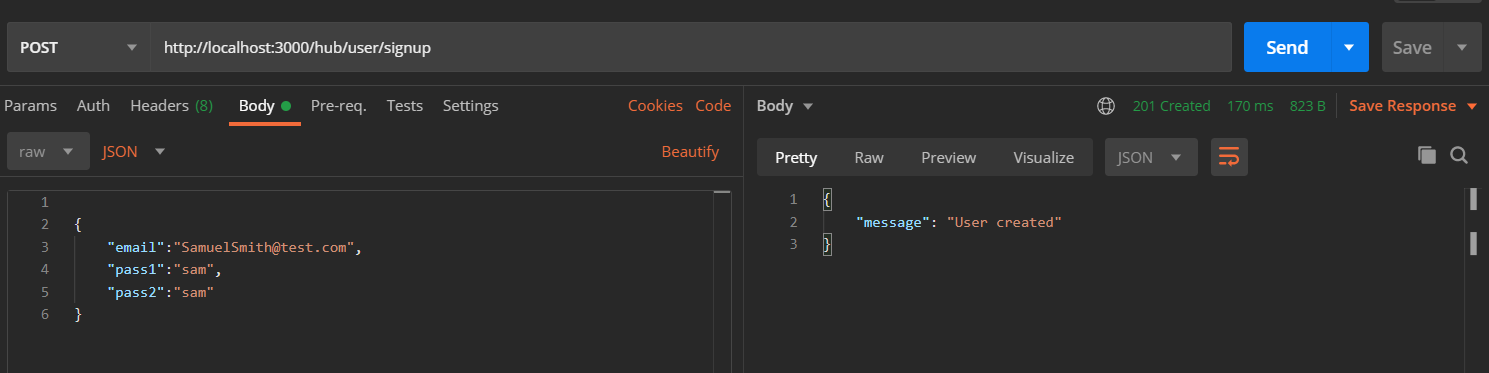
* user\_create\_get – returns the view with the form to create a new user
* user\_create\_post – saves the user to database
* user\_login\_get – renders the form for the user to login
* user\_login\_post – checks details and performs authentication
* user\_delete\_post – deletes an user and returns message.

Routes:

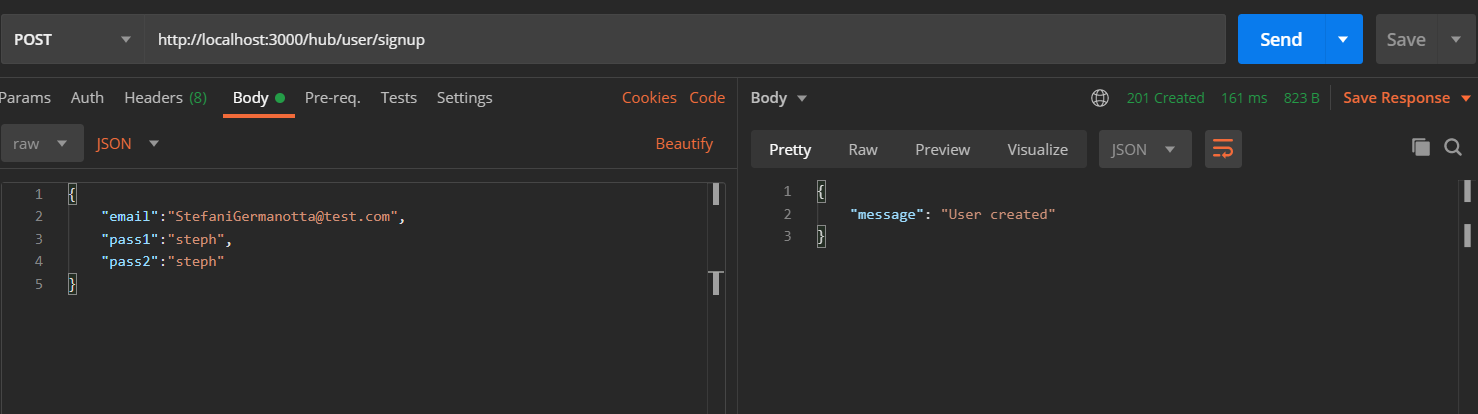


For creation of users I have added double password field that will have to match to create a user:

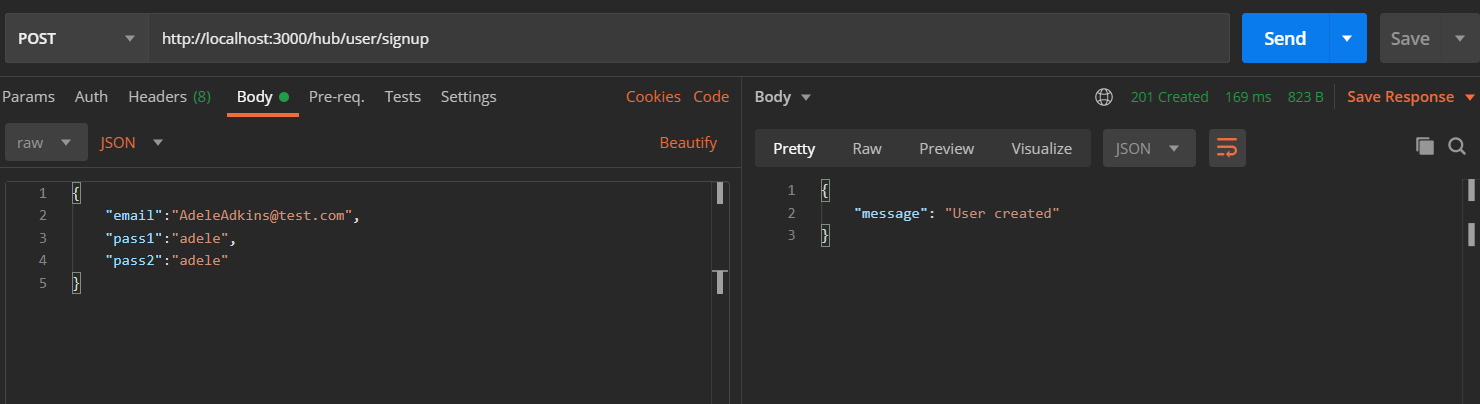
[SamuelSmith@test.com](mailto:SamuelSmith@test.com) – sam



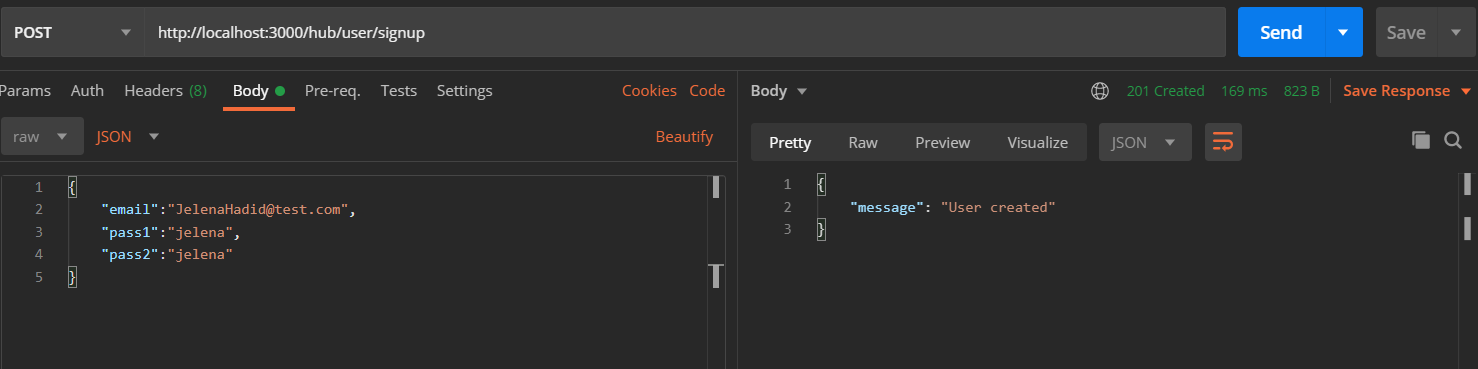
[StefaniGermanotta@test.com](mailto:StefaniGermanotta@test.com) – steph



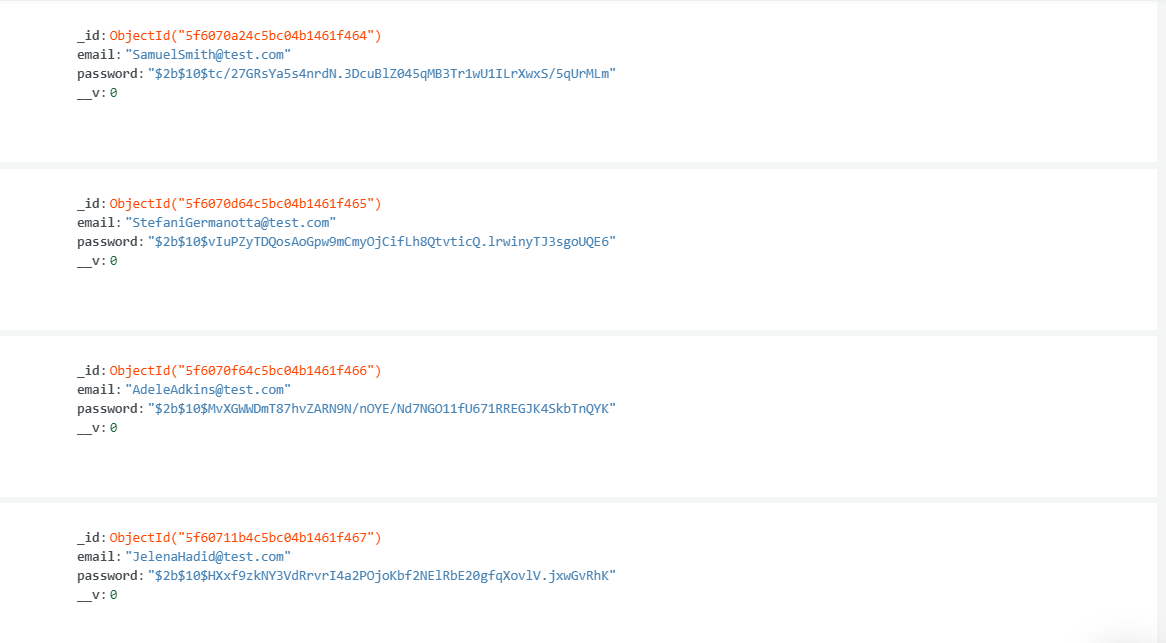
[AdeleAdkins@test.com](mailto:AdeleAdkins@test.com) – adele



[JelenaHadid@test.com](mailto:JelenaHadid@test.com) – jelena



Mong DB



## JWT authentication:

# API Testing

## Routes testing:

## Authentication tests:

## 10 CRUD tests: