# Exercises: Asynchronous Programming and Promises

Problems for exercises and homework for the [“JavaScript Applications” course @ SoftUni](https://softuni.bg/courses/javascript-applications). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/361/>.

## Forecaster

Write a JS program that requests a weather report from a server and displays it o the user. Use the following HTML to test your code:

|  |
| --- |
| schedule.html |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Forecatser</**title**>  <**style**>  **#content** { **width**: 400**px**; }  **#request** { **text-align**: **center**; }  .**bl** { **width**: 300**px**; }  **#current** { **text-align**: **center**; **font-size**: 2**em**; }  **#upcoming** { **text-align**: **center**; }  .**condition** { **text-align**: **left**; **display**: **inline-block**; }  .**symbol** { **font-size**: 4**em**; **display**: **inline-block**; }  .**forecast-data** { **display**: **block**; }  .**upcoming** { **display**: **inline-block**; **margin**: 1.5**em**; }  .**label** { **margin-top**: 1**em**; **font-size**: 1.5**em**; **background-color**: **aquamarine**; **font-weight**: 400; }  </**style**>  <**script src="https://code.jquery.com/jquery-3.1.1.min.js"**></**script**> </**head**> <**body**> <**div id="content"**>  <**div id="request"**>  <**input id="location" class='bl' type="text"**>  <**input id="submit" class="bl" type="button" value="Get Weather"**>  </**div**>  <**div id="forecast" style="display:none"**>  <**div id="current"**>  <**div class="label"**>Current conditions</**div**>  </**div**>  <**div id="upcoming"**>  <**div class="label"**>Three-day forecast</**div**>  </**div**>  </**div**> </**div**> <**script src="forecaster.js"**></**script**> <**script**>  *attachEvents*(); </**script**> </**body**> </**html**> |

Submit only the attachEvents() function that attaches events to the **button** with ID "**submit**" and holds all program logic.

When the user writes the name of a location and clicks “**Get Weather**”, make a GET request to the server at address https://judgetests.firebaseio.com/locations.json. The response will be an array of objects, with structure:

{ name: locationName,

code: locationCode }

Find the object, corresponding to the name the user submitted in the input field with ID "**location**" and use its code value to make two more requests:

* For current conditions, make a GET request to https://judgetests.firebaseio.com/forecast/today/{code}.json (replace the highlighted part with the relevant value). The response from the server will be an object as follows:

{ name: locationName,

forecast: { low: temp,

high: temp,

condition: condition } }

* For a 3-day forecast, make a GET request to https://judgetests.firebaseio.com/forecast/upcoming/{code}.json (replace the highlighted part with the relevant value). The response from the server will be an object as follows:

{ name: locationName,

forecast: [{ low: temp,

high: temp,

condition: condition }, … ] }

Use the information from these two objects to compose a forecast in HTML and insert it inside the page. Note that the <div> with ID "**forecast**" must be set to **visible**. See the examples for details.

If an error occurs (the server doesn’t respond or the location name cannot be found) or the data is not in the correct format, display "Error" in the forecast section.

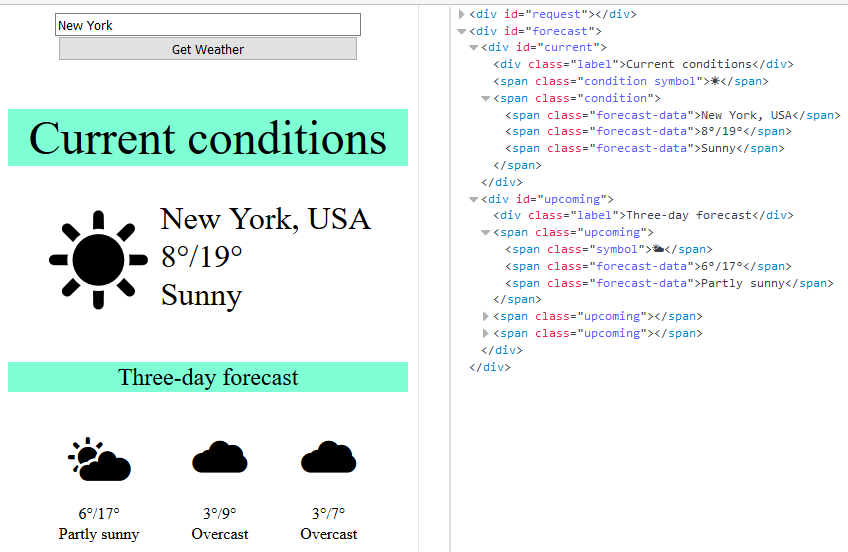
Use the following codes for the weather sumbols:

* Sunny &#x2600; // ☀
* Partly sunny &#x26C5; // ⛅
* Overcast &#x2601; // ☁
* Rain &#x2614; // ☂
* Degrees &#176; // °

### Examples

When the app starts, the forecast div is hidden. When the user enters a name and clicks submit, the requests being.





### Hints

The server at the address listed above will respond with valid data for location names "London", "New York" and "Barcelona".

## \*\*\*Secret Knock

Your task is to perform the Secret Knock. The Secret Knock is a secret knocking technique that is performed with requests, responses and promises. First, you will use Kinvey.

The app credentials are:

* App id / key: kid\_BJXTsSi-e
* App secret: 447b8e7046f048039d95610c1b039390

The guest user is:

* Username: guest
* Password: guest

You will need to log in before you perform any kind of action. Next you will have to send various requests **with queries**. Now a query is a list of parameters added to the URL of the request. Here is the base URL for the requests:

https://baas.kinvey.com/appdata/kid\_BJXTsSi-e/knock

And now you have to add the first query, which is “Knock Knock.” to the URL. Do it like this:

https://baas.kinvey.com/appdata/kid\_BJXTsSi-e/knock?query=Knock Knock.

If you send a **GET request** to this URL with this query, you will receive a response with an **answer** from the server, and the **next message**. Change the **query** with the **next message** in line, and continue this process until you receive a response **with no next message**. Print the **answer** and the **next message** after each successful request on the console, and you’ll be able to see the magic of the Secret Knock.

## Fisher Game

Create an application at **kinvey.com** Create a collection biggestCatches(angler, weight, species, location, bait, captureTime) to hold information about the largest fish caught.

* angler - string representing the name of the person who caught the fish
* weight - floating point number representing the weight of the fish in kilograms
* species - string representing the name of the fish species
* location - string representing the location where the fish was caught
* bait - string representing the bait used to catch the fish
* captureTime - integer number representing the time needed to catch the fish in minutes

### HTML Template

You are given an HTML template to test your code, your task is to attach handlers to the [Load], [Update], [Delete] and [Add] buttons, which make the appropriate GET, PUT, DELETE and POST requests respectively.

|  |
| --- |
| catch.html |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Biggest Catch</**title**>  <**script src="https://code.jquery.com/jquery-3.1.1.min.js"**></**script**>  <**script src="catch.js"**></**script**>  <**style**>  **h1** { **text-align**: **center**; }  **input** { **display**: **block**; }  **div** { **border**: 1**px solid black**; **padding**: 5**px**; **display**: **inline-table**; **width**: 24%; }  **div#aside** { **margin-top**: 8**px**; **width**: 15%; **border**: 2**px solid grey**; }  **div#catches**{ **width**:**auto**; }  **button** { **display**: **inline-table**; **margin**: 5% 0 5% 5%; **width**: 39%; }  **button**.**add** { **width**: 90%; }  **button**.**load** { **width**: 90%; **padding**: 10**px**; }  **button**.**load** { **vertical-align**: **top**; }  **fieldset** { **display**: **inline-table**; **vertical-align**: **top**; }  **fieldset#main** { **width**: 70%; }  </**style**> </**head**> <**body**> <**h1**>Biggest Catches</**h1**> <**fieldset id="main"**>  <**legend**>Catches</**legend**>  <**div id="catches"**>  <**div class="catch" data-id="<id-goes-here>"**>  <**label**>Angler</**label**>  <**input type="text" class="angler" value="Paulo Amorim"**/>  <**label**>Weight</**label**>  <**input type="number" class="weight" value="636"**/>  <**label**>Species</**label**>  <**input type="text" class="species" value="Atlantic Blue Marlin"**/>  <**label**>Location</**label**>  <**input type="text" class="location" value="Vitória, Brazil"**/>  <**label**>Bait</**label**>  <**input type="text" class="bait" value="trolled pink"**/>  <**label**>Capture Time</**label**>  <**input type="number" class="captureTime" value="80"**/>  <**button class="update"**>Update</**button**>  <**button class="delete"**>Delete</**button**>  </**div**>  </**div**> </**fieldset**> <**div id="aside"**>  <**button class="load"**>Load</**button**>  <**fieldset id="addForm"**>  <**legend**>Add Catch</**legend**>  <**label**>Angler</**label**>  <**input type="text" class="angler"**/>  <**label**>Weight</**label**>  <**input type="number" class="weight"**/>  <**label**>Species</**label**>  <**input type="text" class="species"**/>  <**label**>Location</**label**>  <**input type="text" class="location"**/>  <**label**>Bait</**label**>  <**input type="text" class="bait"**/>  <**label**>Capture Time</**label**>  <**input type="number" class="captureTime"**/>  <**button class="add"**>Add</**button**>  </**fieldset**> </**div**> <**script**>*attachEvents*()</**script**> </**body**> </**html**> |

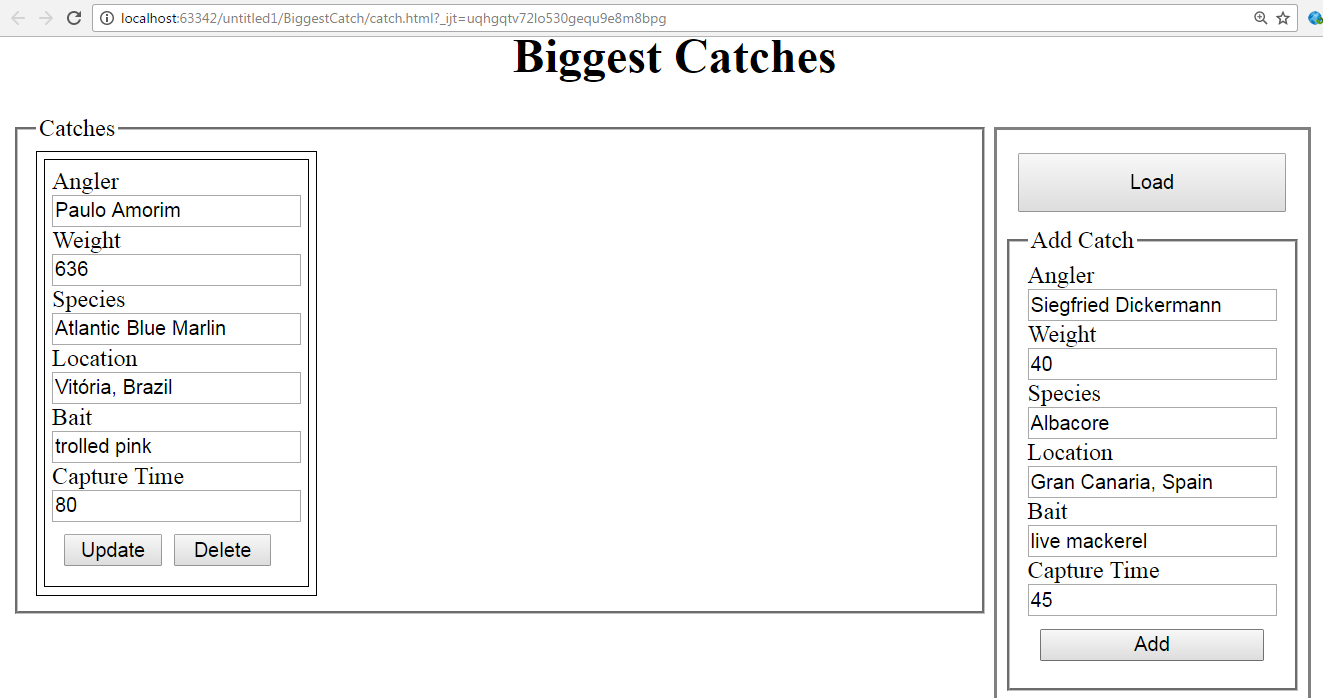
You are given an example catch in the template to show you where and how you should insert the catches. Notice that the div containing the catch has an attribute data-id that should store the \_id of the entry given by Kinvey.

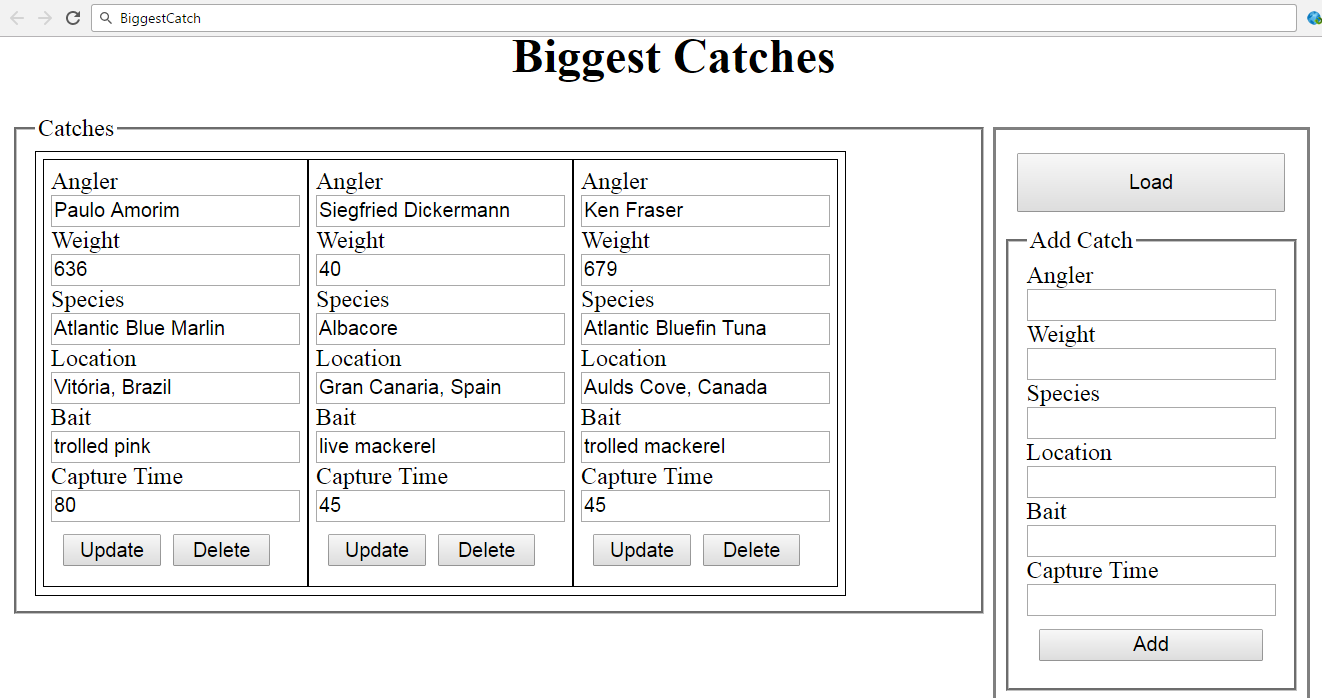
Kinvey will automatically create the following REST services to access your data:

* **List All Catches**
  + Endpoint: [https://baas.kinvey.com/appdata/[:appId]/biggestCatches](https://baas.kinvey.com/appdata/%5b:appId%5d/biggestCatches)
  + Method: GET
  + Headers:
    - Basic Authorization with **user credentials**
  + Returns (JSON)
* **Create a New Catch**
  + Endpoint: [https://baas.kinvey.com/appdata/[:appId]/biggestCatches](https://baas.kinvey.com/appdata/%5b:appId%5d/biggestCatches)
  + Method: POST
  + Headers:
    - Basic Authorization with **user credentials**
    - Content-type: application/json
  + Request body (JSON): {"angler":"…", "weight":…, "species":"…", "location":"…", "bait":"…", "captureTime":…}
* **Update a Catch**
  + Endpoint: [https://baas.kinvey.com/appdata/[:appId]/biggestCatches/[:catchId]](https://baas.kinvey.com/appdata/%5b:appId%5d/biggestCatches/%5b:catchId%5d)
  + Method: PUT
  + Headers:
    - Basic Authorization with **user credentials**
    - Content-type: application/json
  + Request body (JSON): {"angler":"…", "weight":…, "species":"…", "location":"…", "bait":"…", "captureTime":…}
* **Delete a Catch**
  + Endpoint: [https://baas.kinvey.com/apdata/[:appId]/biggestCatches/[:catchId]](https://baas.kinvey.com/apdata/%5b:appId%5d/biggestCatches/%5b:catchId%5d)
  + Method: DELETE
  + Headers:
    - Basic Authorization with **user credentials**
    - Content-type: application/json

Pressing the [Load] button should list all catches, pressing a catch's [Update] button should send a PUT requests updating that catch in kinvey.com. Pressing a catch's [Delete] button should delete the catch both from kinvey and from the page. Pressing the [Add] button should submit a new catch with the values of the inputs in the Add fieldset.

### Screenshots



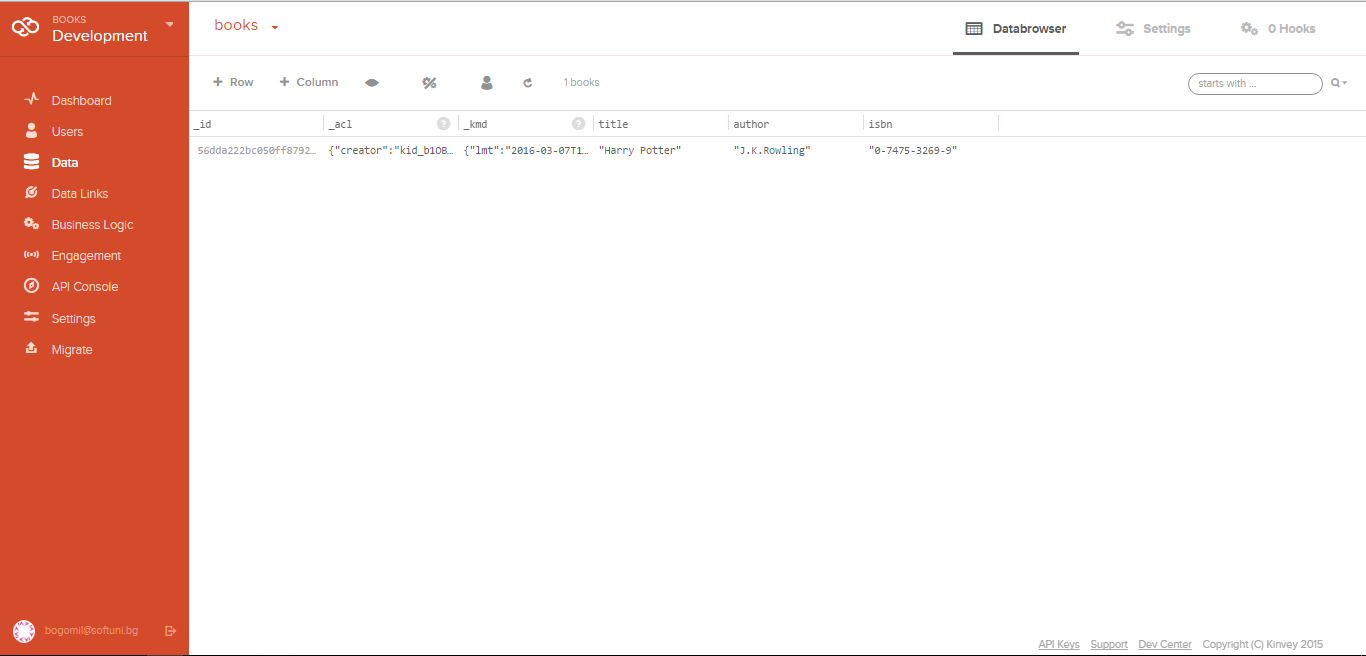


# Extra tasks

The following tasks don't have automated tests in the Judge system, they are for practicing.

## Create "Books" REST Service

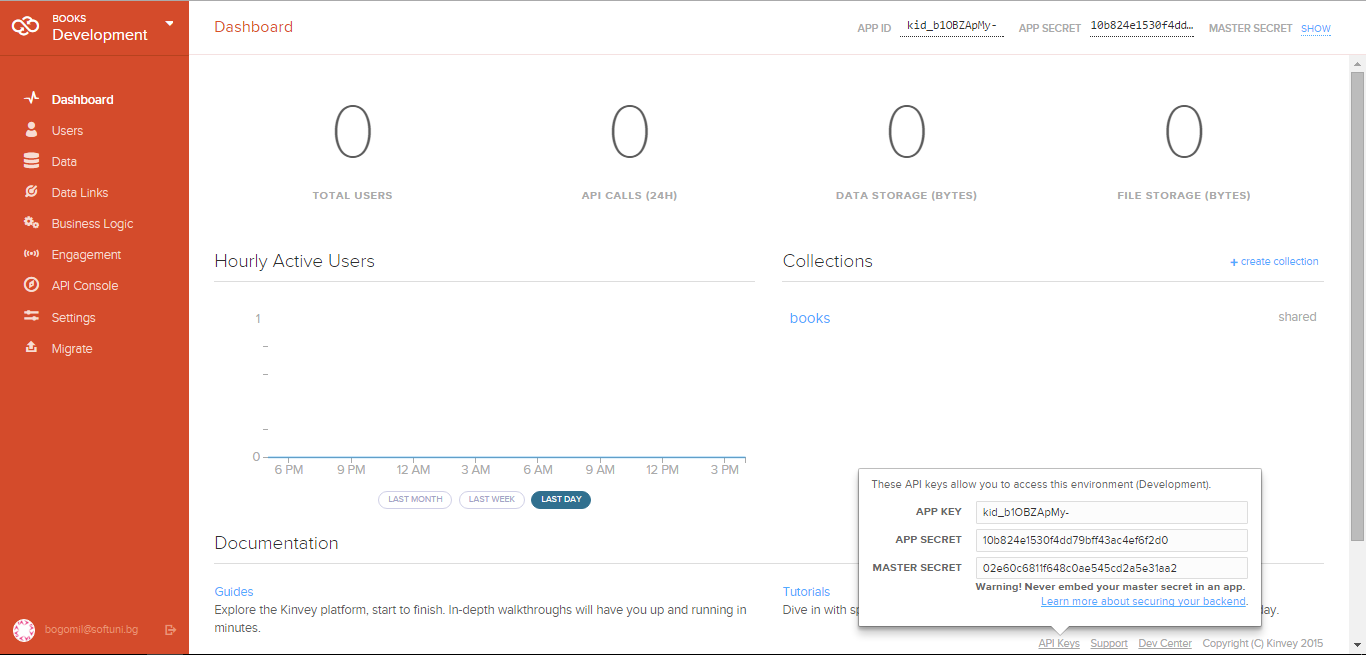
Register at **kinvey.com** and create an application. Create a class **Book(title, author, isbn)** to hold book objects. Fill a few sample books:



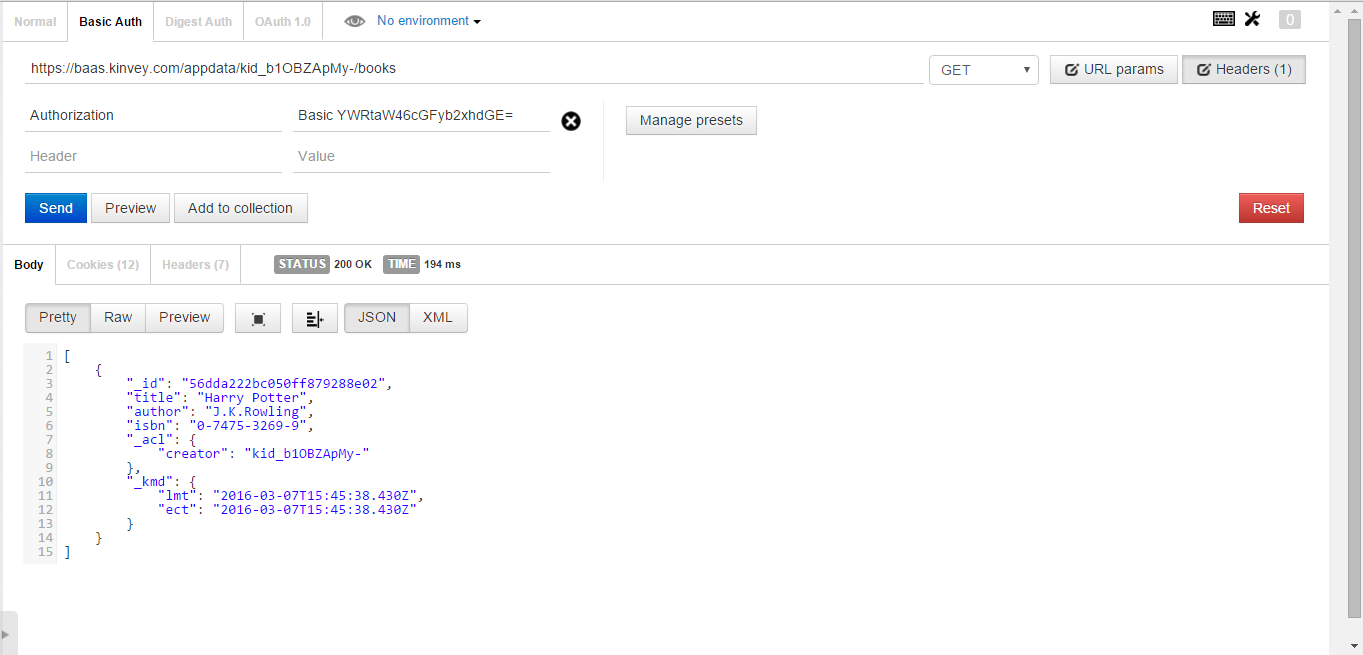
kinvey.com will automatically create the following REST services to access your data:

* **List All Books**
  + Endpoint: [https://baas.kinvey.com/apdata/[:appId]/books](https://baas.kinvey.com/apdata/%5b:appId%5d/books)
  + Method: GET
  + Headers:
    - Basic Authorization with **user credentials**
  + Returns (JSON)
* **Create a New Book**
  + Endpoint: [https://baas.kinvey.com/apdata/[:appId]/books](https://baas.kinvey.com/apdata/%5b:appId%5d/books)
  + Method: POST
  + Headers:
    - Basic Authorization with **user credentials**
    - Content-type: application/json
  + Request body (JSON): {"title":"…", "author":"…", "isbn":"…"}
* **Update a Book**
  + Endpoint: [https://baas.kinvey.com/apdata/[:appId]/books/[:bookId]](https://baas.kinvey.com/apdata/%5b:appId%5d/books/%5b:bookId%5d)
  + Method: PUT
  + Headers:
    - Basic Authorization with **user credentials**
    - Content-type: application/json
  + Request body (JSON): {"title":"…", "author":"…", "isbn":"…"}
* **Delete a Book**
  + Endpoint: [https://baas.kinvey.com/apdata/[:appId]/books/[:bookId]](https://baas.kinvey.com/apdata/%5b:appId%5d/books/%5b:bookId%5d)
  + Method: DELETE
  + Headers:
    - Basic Authorization with **user credentials**
    - Content-type: application/json

To view your kinvey.com access keys, go to your application dashboard 🡪 bottom-right menu 🡪 API keys:



Test your REST Service, e.g. using **Postman** Chrome Extension (<http://www.getpostman.com>). Try to list all books in JSON format with an HTTP GET request to the REST API of kinvey.com.



## List All Books

Create a HTML5 project consisting of HTML, CSS and JS files. Add an AJAX call that takes all books from your application in kinvey.com as JSON object and displays them at page load.

## Create a Book

Add a HTML form with submit button for adding a new book. When the button is pressed, create a new book at kinvey.com using its REST API with an AJAX request.

## Edit a Book

Implement "Edit a Book" functionality. Clicking on a book should load its data in a HTML form. By clicking the submit button, the book data at kinvey.com should be updated at the server side with and AJAX request.

## Delete a Book

Implement "Delete a Book" functionality. Each book should have "Delete" button. Clicking on it should delete the book at the server side with and AJAX request to the REST service.

## \* Add Tags for Each Book

Implement tags for the books. Tags should be stored at kinvey.com in the Book class in a column "**tags**" as arrays of strings. List the tags for each book. Implement add / edit / delete for tags when a book is created / updated.