# Exercises: AJAX with 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/362/>.

# Practice

The following tasks don't have automated tests in the Judge system, you are expected to check the problems yourself.

## Students

Your task is to create functionality for creating and listing students from a database in Kinvey.

Here are the Kinvey application key and secret:

* APP KEY: kid\_BJXTsSi-e
* APP SECRET: 447b8e7046f048039d95610c1b039390

Here is also a test user for you to log in with:

* Username: guest
* Password: guest

There is collection called students there, in the application.

The student entity has:

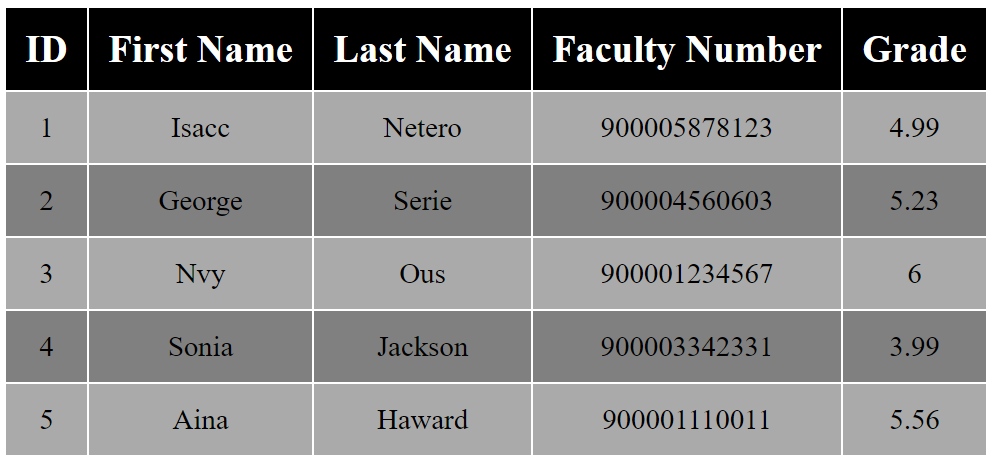
* ID – a number, **non-empty**
* FirstName – a String, **non-empty**
* LastName – a String, **non-empty**
* FacultyNumber – a String of **numbers**, **non-empty**
* Grade – a number, **non-empty**

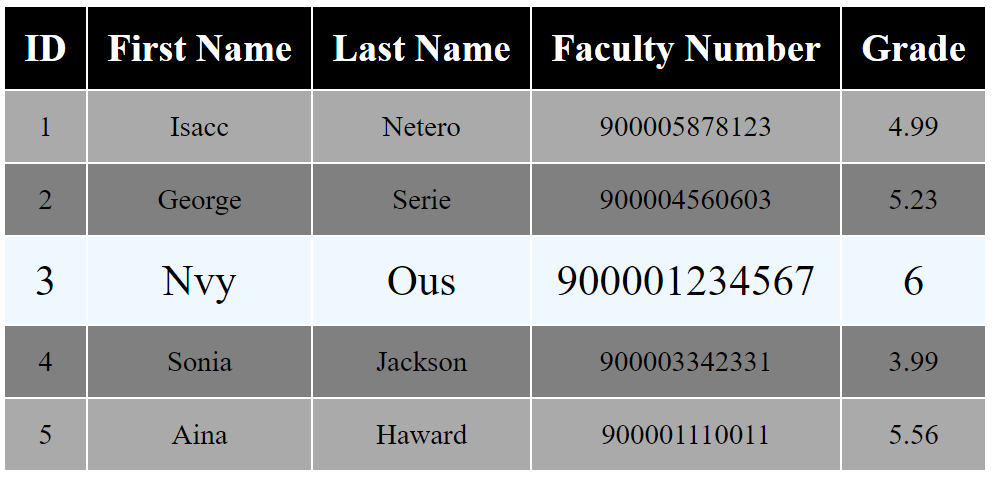
You need to write functionality for creating students. When you are creating a new student, make sure you name these properties perfectly. Create at least one student to test your code.

You will also need to extract the students. You will be given an **HTML template** with a table in it. Create an AJAX request which extracts all the students. Upon fetching all the students from Kinvey, add them to the table each on a new row, **sorted** in **ascending order** by ID.

|  |
| --- |
| students.html |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Shit</**title**>  <**script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"**></**script**>  <**style**>  **#results** {  **background-color**: **#FFFFFF**;  **display**: **flex**;  **flex-direction**: **column**;  **text-align**: **center**;  }   **#results tr** {  **background-color**: **#AAAAAA**;  **padding**: 5**vh**;  **font-size**: 1.5**vw**;  }   **#results tr**:**nth-child**(**odd**) {  **background-color**: **#808080**;  }   **#results tr**:**first-child** {  **background-color**: **#000000**;  **color**: **#FFFFFF**;  **font-weight**: **bold**;  **font-size**: 2**vw**;  }   **#results tr th** {  **padding**: 1**vw**;  }   **#results tr td** {  **padding**: 1**vw**;  **transition**: **font-size** 0.2**s**;  }   **#results tr**:**not**(:**first-child**):**hover** {  **background-color**: **#F0F8FF**;  **color**: **#000000**;  **font-size**: 2.25**vw**;  }   </**style**> </**head**> <**body**> <**table id="results"**>  <**tr**>  <**th**>ID</**th**>  <**th**>First Name</**th**>  <**th**>Last Name</**th**>  <**th**>Faculty Number</**th**>  <**th**>Grade</**th**>  </**tr**> </**table**> *<!--<script src="script.js"></script>-->* </**body**> </**html**> |

### Screenshots





## Countries & Towns

### Create a Backend at kinvey.com

Register at **kinvey.com**, create an application and two classes: **Country(name)** and **Town(name, country)**. Fill sample data for further use. Submit in the homework screenshots of your classes from kinvey.com.

### List Countries

Create a JavaScript application (HTML + CSS + JS + jQuery) that loads and displays all countries from your application at Kinvey.com into a HTML page.

### Edit Countries

Extend the previous application with add / edit / delete for countries.

### List Towns by Country

Extend the previous application to show all towns when a certain country is selected.

### CRUD Towns

Extend the previous application to implement add / edit / delete for towns.

## Venuemaster

Write a JS program that displays information about venues and allows the user to buy a ticket. Use the following HTML to test your code:

|  |
| --- |
| venuemaster.html |
| <!DOCTYPE **html**>  <**html lang="en"**>  <**head**>  <**meta charset="UTF-8"**>  <**title**>Venue Master</**title**>  <**script src="https://code.jquery.com/jquery-3.1.1.min.js"**></**script**>  <**style**>  **#content** { **width**: 800**px**; }  .**venue** { **border**: 1**px solid black**; **margin**: 0.5**em**; }  .**venue-name** { **background-color**: **beige**; **padding**: 0.1**em**; **display**: **block**; **font-size**: 2**em**; **padding-left**: 1**em**; }  .**venue-name input** { **margin-right**: 1**em**; }  .**head** { **background-color**: **beige**; **padding**: 0.1**em**; **padding-left**: 1.5**em**; **display**: **block**; **font-size**: 1.5**em**; }  .**description** { **margin**: 2**em**; }  **table** { **text-align**: **center**; **margin**: 2**em**; }  **td**, **th** { **width**: 100**px**; }  .**purchase-info span** { **display**: **inline-block**; **width**: 100**px**; **margin-left**: 2**em**; **margin-right**: 2**em**; }  .**ticket** { **border**: 1**px solid black**; **text-align**: **center**; **overflow**: **hidden**; }  .**bl** { **display**: **inline-block**; **font-size**: 1.5**em**; **margin**: 1**em** 3**em** 1**em** 3**em**; }  .**left** { **width**: 600**px**; **float**:**left**; }  .**right** { **float**:**right**; }  </**style**>  </**head**>  <**body**>  <**div id="content"**>  <**div id="date-control"**>  <**input type="text" id="venueDate" placeholder="Enter date"**>  <**input type="button" id="getVenues" value="List Venues"**>  </**div**>  <**div id="venue-info"**>  </**div**>  </**div**>  <**script src="venuemaster.js"**></**script**>  <**script**>  *attachEvents*();  </**script**>  </**body**>  </**html**> |

Submit only the attachEvents() function that attaches events and holds all program logic.

You can use the following Kinvey database and credentials to test your solution:

App ID: kid\_BJ\_Ke8hZg

User: guest

Password: pass

You can consult previous lectures about making requests to Kinvey databases: [Asynchronous Programming](https://softuni.bg/downloads/svn/js-core/Sept-2016/JS-Apps-Nov-2016/4.%20JS-Apps-Async-Programming-Promises/4.%20JS-Apps-Async-Programming-and-Promises.pptx). Note that certain requests are made to rpc/ instead of appdata/ – take care when copy/pasting code!

When the user clicks on the **button** with ID "**getVenues**", take the value of the input field with ID "**venueDate**" and make a POST request to rpc/kid\_BJ\_Ke8hZg/custom/calendar?query={date} (replace the highlighted part with the relevant value). The server will respond with and **array**, containing the IDs of available venues for that date. Use those IDs to obtain information from the server about **each** of the venues – make a GET request to appdata/kid\_BJ\_Ke8hZg/venues/{\_id} (replace the highlighted part with the relevant value). The server will respond with an object in the following format:

{ name: *String*,

description: *String*,

startingHour: *String*,

price: *Number* }

Compose a list with all venues and display it on the page inside the <div> with ID "**venue-info**". Use the following HTML to format the results:

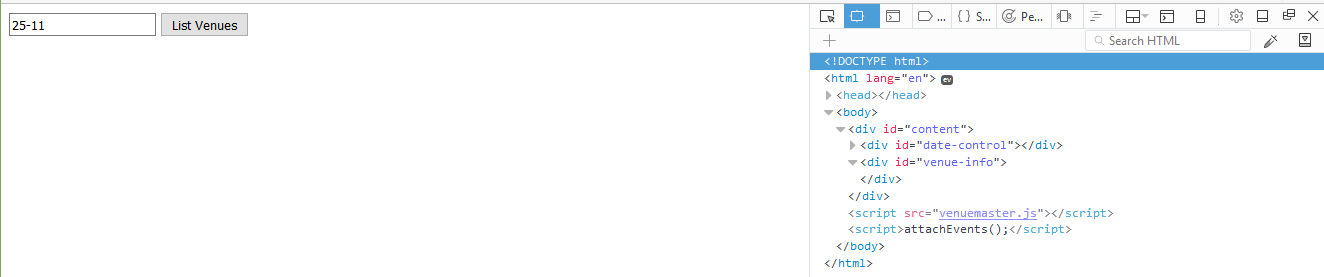
|  |
| --- |
| Venue Template |
| <**div class="venue" id="**{venue.\_id}**"**>  <**span class="venue-name"**><**input class="info" type="button" value="More info"**>{venue.name}</**span**>  <**div class="venue-details" style="display**: **none**;**"**>  <**table**>  <**tr**><**th**>Ticket Price</**th**><**th**>Quantity</**th**><**th**></**th**></**tr**>  <**tr**>  <**td class="venue-price"**>{venue.price} lv</**td**>  <**td**><**select class="quantity"**>  <**option value="1"**>1</**option**>  <**option value="2"**>2</**option**>  <**option value="3"**>3</**option**>  <**option value="4"**>4</**option**>  <**option value="5"**>5</**option**>  </**select**></**td**>  <**td**><**input class="purchase" type="button" value="Purchase"**></**td**>  </**tr**>  </**table**>  <**span class="head"**>Venue description:</**span**>  <**p class="description"**>{venue.description}</**p**>  <**p class="description"**>Starting time: {venue.startingHour}</**p**>  </**div**>  </**div**> |

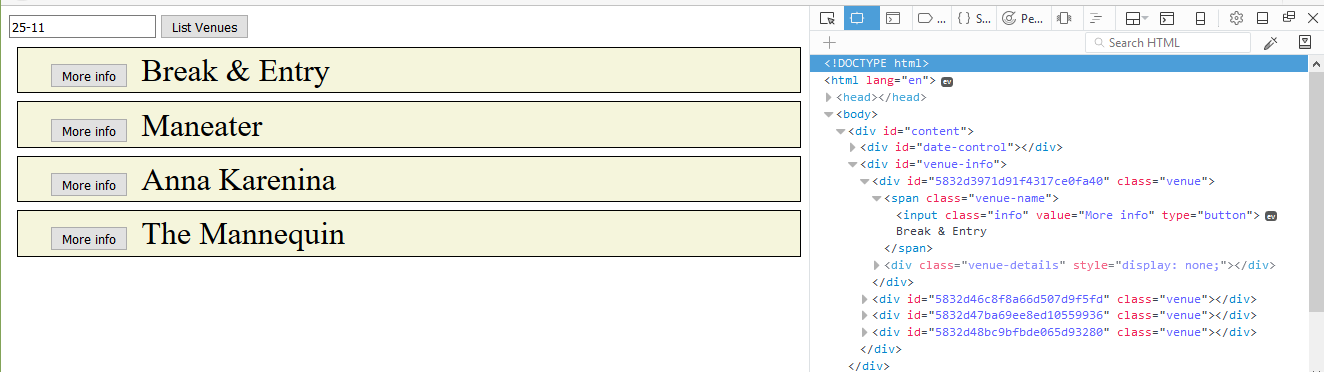
Each item in the list has a button "**More info**", that changes the visibility of the detailed description for the corresponding venue – hide all descriptions (set style to "**display: none**") and show the current description (set style to "**display: block**"). The detailed view has a numeric drop-down and a button "**Buy tickets**". When this button is clicked, take the user to the confirmation page – change the contents of the "**#venue-info**" div, using the following HTML:

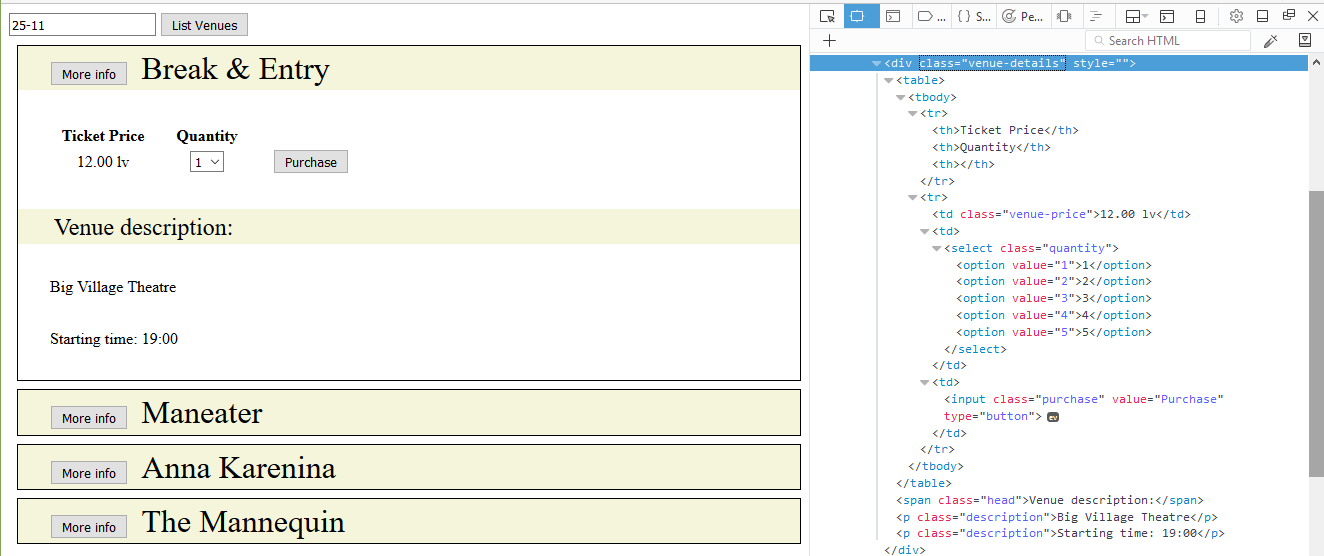
|  |
| --- |
| Confirmation Template |
| <**span class="head"**>Confirm purchase</**span**>  <**div class="purchase-info"**>  <**span**>{name}</**span**>  <**span**>{qty} x {price}</**span**>  <**span**>Total: {qty \* price} lv</**span**>  <**input type="button" value="Confirm"**>  </**div**> |

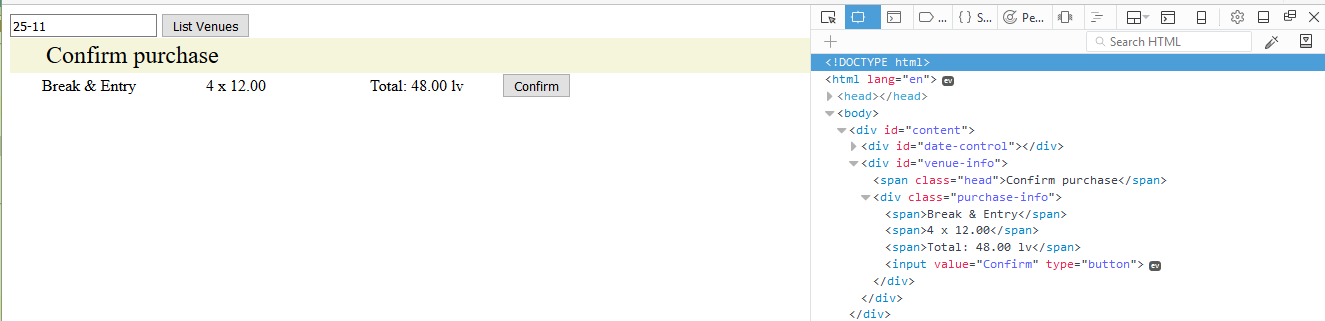
The final step is the confirmation of the purchase – when the user clicks on the button with ID "**confirm**", make a POST request to rpc/kid\_BJ\_Ke8hZg/custom/purchase?venue={\_id}&qty={qty} – the server will return an object, containing an HTML fragment in it’s html property. Display that fragment inside "**#venue-info**" along with the text "**You may print this page as your ticket**".

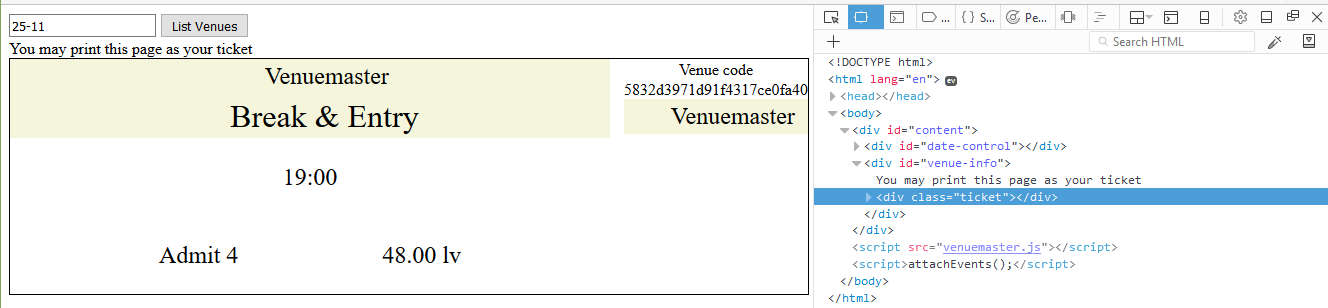
### Examples











## Wild Wild West

Write the REST services for a simple Western game. Create a collection players(name, money, bullets) to hold information about the players in the game.

* name - string representing the name of the current player.
* money - integer number representing the current player’s money.
* bullets - integer number representing the current bullets of the player.

### HTML and JS

You will be provided with a skeleton project containing an HTML template and some JS files, the loadCanvas.js is simple implementation for the game, your job is to attach events to all the buttons and make the needed AJAX requests. You are provided with a file to write your code:

|  |
| --- |
| attachEvents.js |
| **function** *attachEvents*() {  *//****TODO*** } |

When the page is loaded a GET request should be sent to the server to get all players and load them in the #players div, an example entry is left in the HTML to demonstrate the HTML representation of a player and his placement.

Whenever the [Save] button is pressed the progress of the current player (if there is one) should be saved (a PUT request sent to the server with the new data), the canvas and buttons [Save] and [Reload] should be hidden and the clearInterval should be called on the canvas.intervarId property (used for the main loop of the game).

Whenever the [Reload] button is pressed the player’s money should be **reduced by 60** and his bullets should be **set to 6**.

Whenever the [Add Player] button is clicked a new Player with the name specified in the corresponding input should be created and the players should be reloaded to display the new entry. Each new player starts with 500 Money and 6 bullets.

Pressing the [Play] button on a player should first call the **[Save]** button, then display the canvas, [Save] and [Reload] buttons and after that call the loadCanvas function (from the loadCanvas.js) and pass to it the **new player** as an **object** (containing properties name, money and bullets).

When a player’s [Delete] button is pressed the player should be deleted (both from the HTML and from the server).

### Examples

