

Exercises: Asynchronous Programming and Promises

1. Forecaster

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

schedule.html
<pre><!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <title>Forecaster</title> <style> #content { width: 400px; } #request { text-align: center; } .bl { width: 300px; } #current { text-align: center; font-size: 2em; } #upcoming { text-align: center; } .condition { text-align: left; display: inline-block; } .symbol { font-size: 4em; display: inline-block; } .forecast-data { display: block; } .upcoming { display: inline-block; margin: 1.5em; } .label { margin-top: 1em; font-size: 1.5em; 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></pre>

```

    </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 symbols:

- Sunny `☀ // ☀`
- Partly sunny `⛅ // ⛅`
- Overcast `☁ // ☁`
- Rain `☔ // ☔`
- Degrees `° // °`

Examples

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

<input type="text" value="New York"/> <button>Get Weather</button>	<pre><div id="request"></div> <div id="forecast" style="display: none;"> <div id="current"></div> <div id="upcoming"></div> </div></pre>
<div><div>Current conditions</div><div><div>New York, USA 8°/19° Sunny</div></div><div>Three-day forecast</div><div><div><div>6°/17° Partly sunny</div></div><div><div>3°/9° Overcast</div></div><div><div>3°/7° Overcast</div></div></div></div>	<pre><div id="request"></div> <div id="forecast"> <div id="current"> <div class="label">Current conditions</div> ☀ New York, USA 8°/19° Sunny </div> <div id="upcoming"> <div class="label">Three-day forecast</div> ⛅ 6°/17° Partly sunny </div> </div></pre>

Hints

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

2. 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
<pre><!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: 1px solid black; padding: 5px; display: inline-table; width: 24%; } div#aside { margin-top: 8px; width: 15%; border: 2px 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: 10px; } button.load { vertical-align: top; } fieldset { display: inline-table; vertical-align: top; } fieldset#main { width: 70%; } </style> </head> <body> <h1>Biggest Catches</h1></pre>

```

<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/[:appId]/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/[:appId]/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/[:appId]/biggestCatches/[:catchId])
 - 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/appdata/\[:appId\]/biggestCatches/\[:catchId\]](https://baas.kinvey.com/appdata/[:appId]/biggestCatches/[:catchId])
 - 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

The screenshot shows a web browser window with the address bar displaying 'localhost:53342/untitled1/BiggestCatch/catch.html?_ijt=uqhgqiv72fo530gequ9ellm8bpg'. The page title is 'Biggest Catches'. The main content area is titled 'Catches' and contains a single form with the following fields: Angler (Paulo Amorim), Weight (636), Species (Atlantic Blue Marlin), Location (Vitória, Brazil), Bait (trolled pink), and Capture Time (80). Below these fields are 'Update' and 'Delete' buttons. To the right of the 'Catches' section is an 'Add Catch' section with a 'Load' button at the top and an 'Add' button at the bottom. The 'Add Catch' section contains the following fields: Angler (Siegfried Dickermann), Weight (40), Species (Albacore), Location (Gran Canaria, Spain), Bait (live mackerel), and Capture Time (45).

The screenshot shows the same web browser window, but now the 'Catches' section displays three forms side-by-side. Each form has its own 'Update' and 'Delete' buttons. The first form is identical to the one in the previous screenshot. The second form has the following fields: Angler (Siegfried Dickermann), Weight (40), Species (Albacore), Location (Gran Canaria, Spain), Bait (live mackerel), and Capture Time (45). The third form has the following fields: Angler (Ken Fraser), Weight (679), Species (Atlantic Bluefin Tuna), Location (Aulds Cove, Canada), Bait (trolled mackerel), and Capture Time (45). The 'Add Catch' section on the right remains the same, with a 'Load' button at the top and an 'Add' button at the bottom. The 'Add Catch' section contains the following fields: Angler, Weight, Species, Location, Bait, and Capture Time.

Extra tasks

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

3. 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:

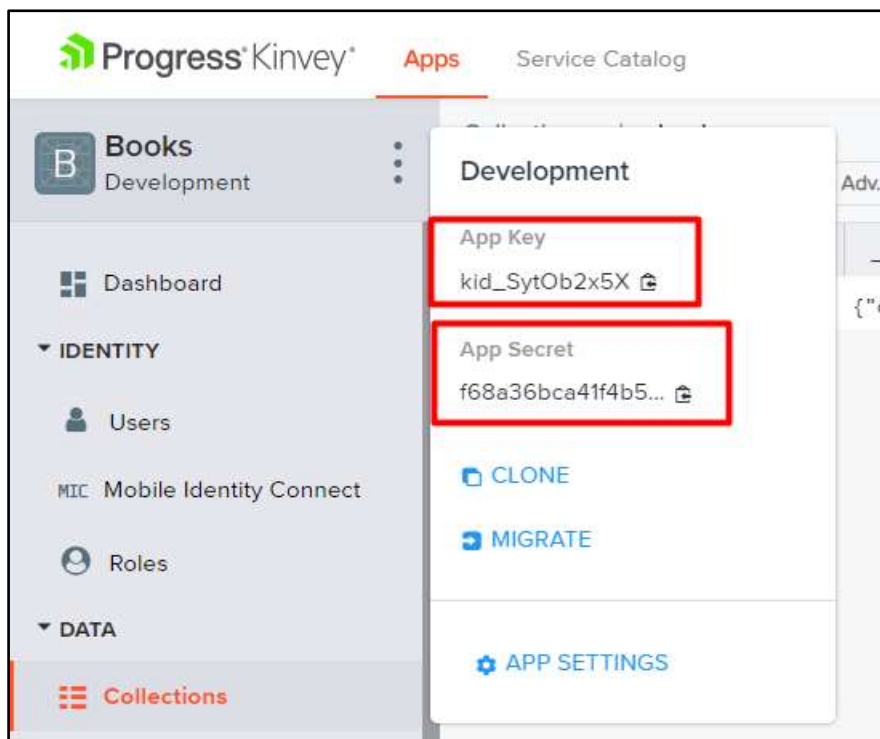


kinvey.com will automatically create the following REST services to access your data:

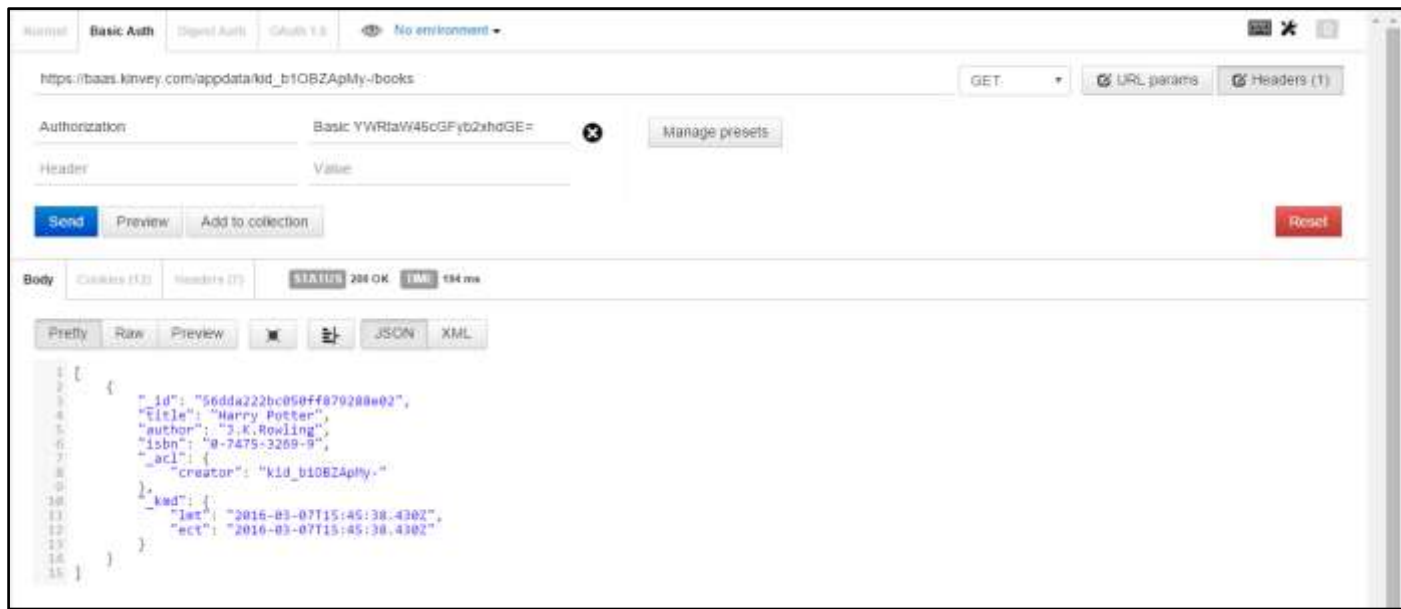
- **List All Books**
 - Endpoint: [https://baas.kinvey.com/apdata/\[:appId\]/books](https://baas.kinvey.com/apdata/[:appId]/books)
 - Method: GET
 - Headers:
 - Basic Authorization with **user credentials**
 - Returns (JSON)
- **Create a New Book**
 - Endpoint: [https://baas.kinvey.com/apdata/\[:appId\]/books](https://baas.kinvey.com/apdata/[:appId]/books)
 - 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/[:appId]/books/[:bookId])
 - 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/[:appId]/books/[:bookId])
 - Method: DELETE
 - Headers:
 - Basic Authorization with **user credentials**
 - Content-type: application/json

To view your kinvey.com access keys, go to your application dashboard → upper-left:



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 an 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 an 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.

Practice

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

4. 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**.

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
<pre><!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"></scr ipt></pre>

```

<style>
  #results {
    background-color: #FFFFFF;
    display: flex;
    flex-direction: column;
    text-align: center;
  }

  #results tr {
    background-color: #AAAAAA;
    padding: 5vh;
    font-size: 1.5vw;
  }

  #results tr:nth-child(odd) {
    background-color: #808080;
  }

  #results tr:first-child {
    background-color: #000000;
    color: #FFFFFF;
    font-weight: bold;
    font-size: 2vw;
  }

  #results tr th {
    padding: 1vw;
  }

  #results tr td {
    padding: 1vw;
    transition: font-size 0.2s;
  }

  #results tr:not(:first-child):hover {
    background-color: #F0F8FF;
    color: #000000;
    font-size: 2.25vw;
  }

</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

ID	First Name	Last Name	Faculty Number	Grade
1	Isacc	Netero	900005878123	4.99
2	George	Serie	900004560603	5.23
3	Nvy	Ous	900001234567	6
4	Sonia	Jackson	900003342331	3.99
5	Aina	Haward	900001110011	5.56

ID	First Name	Last Name	Faculty Number	Grade
1	Isacc	Netero	900005878123	4.99
2	George	Serie	900004560603	5.23
3	Nvy	Ous	900001234567	6
4	Sonia	Jackson	900003342331	3.99
5	Aina	Haward	900001110011	5.56

5. 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.

6. 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: 800px; }
    .venue { border: 1px solid black; margin: 0.5em; }
    .venue-name { background-color: beige; padding: 0.1em; display: block;
font-size: 2em; padding-left: 1em; }
    .venue-name input { margin-right: 1em; }
    .head { background-color: beige; padding: 0.1em; padding-left: 1.5em;
display: block; font-size: 1.5em; }
    .description { margin: 2em; }
    table { text-align: center; margin: 2em; }
    td, th { width: 100px; }
    .purchase-info span { display: inline-block; width: 100px; margin-left:
2em; margin-right: 2em; }
    .ticket { border: 1px solid black; text-align: center; overflow: hidden;
}
    .bl { display: inline-block; font-size: 1.5em; margin: 1em 3em 1em 3em;
}
    .left { width: 600px; 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](#). 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 an **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
<pre> <div class="venue" id="{venue._id}"> <input class="info" type="button" value="More info">{venue.name} <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"> </pre>


```

        <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
<pre> Confirm purchase <div class="purchase-info"> {name} {qty} x {price} Total: {qty * price} lv <input type="button" value="Confirm"> </div> </pre>

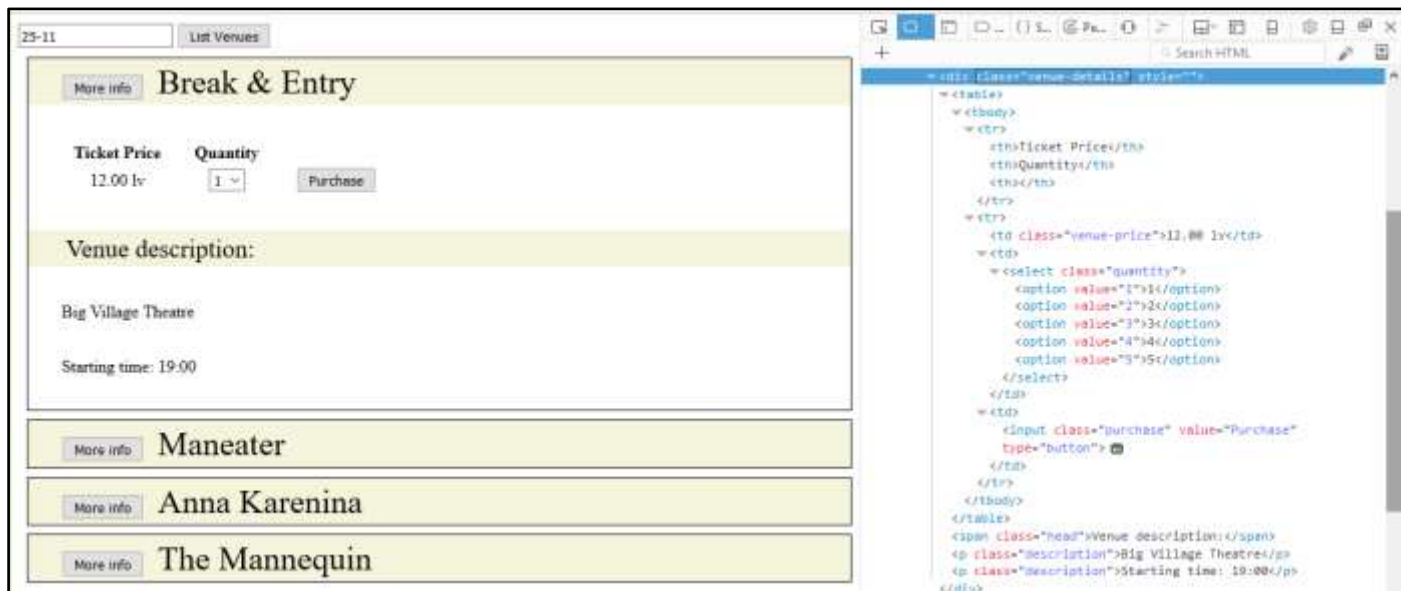
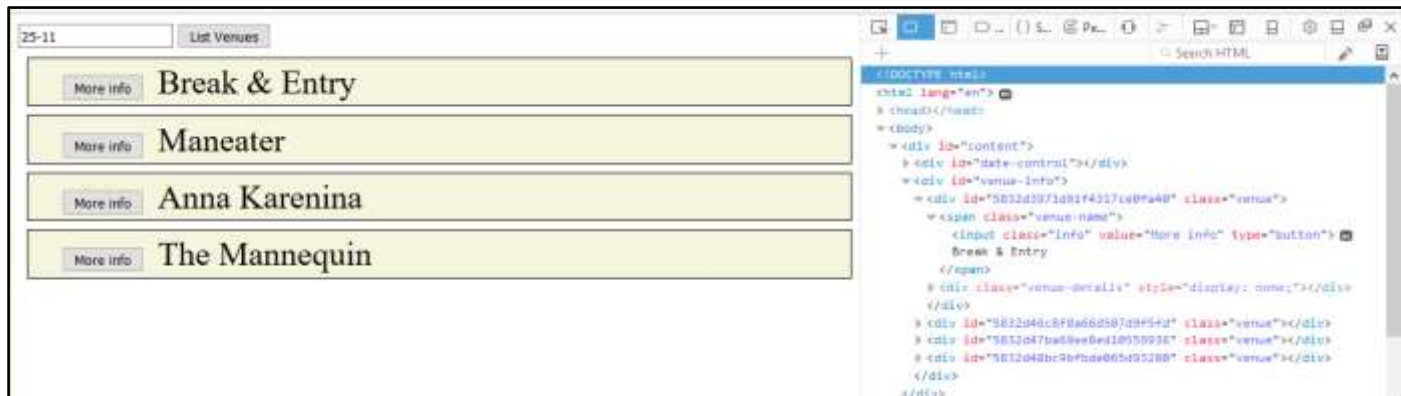
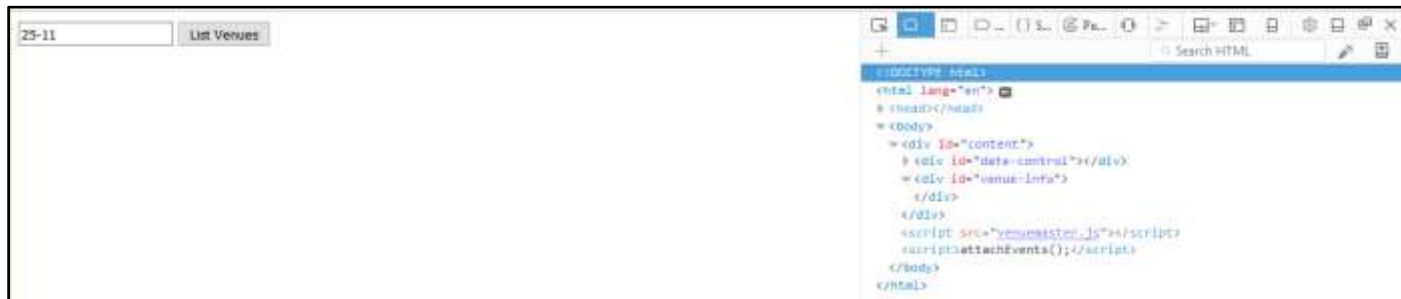
The final step is the confirmation of the purchase – when the user clicks on the button with ID "**confirm**", make a **POST** request to

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**".

Hints

The service at the given address will respond with valid information for dates "23-11", "24-11", "25-11", "26-11" and "27-11", in this exact format. Use these to test your solution (no validation is required).

Examples





7. ***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.

8. 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
<pre>function attachEvents() { //TODO }</pre>

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

Players

Name: Pesho
Money: 520
Bullets: 5
Play
Delete

Name: Ivan
Money: 600
Bullets: 1
Play
Delete

Add Player

Add Player

Save

Reload

Player: Pesho

Money: 520

Bullets: 5

Players

<div>Name: Pesho</div> <div>Money: 520</div> <div>Bullets: 5</div> <div>Play</div> <div>Delete</div>	<div>Name: Ivan</div> <div>Money: 600</div> <div>Bullets: 1</div> <div>Play</div> <div>Delete</div>
--	---

Add Player

Add Player