Exercises: Asynchronous Programming

1. Github Commits

Write a JS program that loads all commit messages and their authors from a github repository using a given HTML.

HTML Template

You are given the following HTML:

commits.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Github Commits</title>
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <style>
        @import url(https://fonts.googleapis.com/css?family=Open+Sans);
        body {
          font-family: "Open Sans", serif;
        }
        input[type=text] {
            padding: 5px 10px;
            margin: 8px 0;
            display: inline-block;
            border: 1px solid #ccc;
            border-radius: 4px;
        }
        button {
          background-color: #4caf50;
          color: white;
          padding: 10px 14px;
          margin: 8px 0;
          border: none;
          border-radius: 4px;
          cursor: pointer;
        }
    </style>
</head>
```

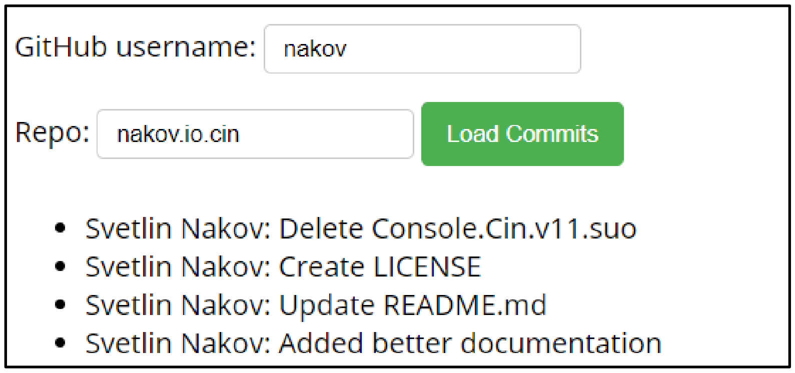
```
<body>
   GitHub username:
   <input type="text" id="username" value="nakov" /> <br>
   Repo: <input type="text" id="repo" value="nakov.io.cin"/>
   <button onclick="loadCommits()">Load Commits</button>
   <ul id="commits"></ul>
   <script>
       function loadCommits() {
           // AJAX call …
       }
   </script>
</body>
</html>
```

The **`loadCommits()`** function should get the **username** and **repository** from the **HTML** textboxes with ids **"username"** and **"repo"** and make a **GET** request to the **Github API**:
"https://api.github.com/repos/<username>/<repository>/commits"

Swap **`<username>`** and **`<repository>`** with the ones from the HTML:

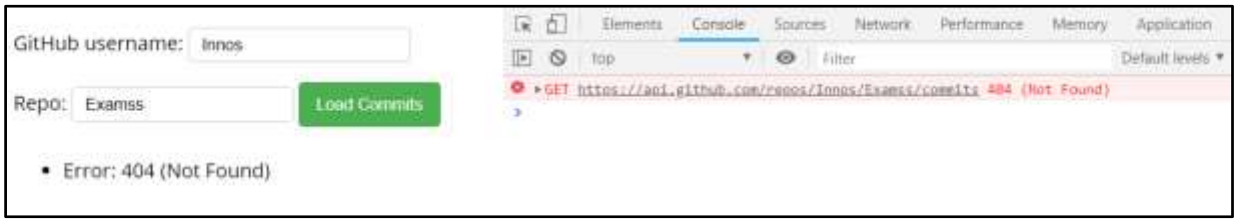
- In case of **success**, for **each** entry add a **list item** (`li`) in the **unordered list** (`ul`) with **`id="commits"`** with text in the format:
  `"<commit.author.name>: <commit.message>"`
- In case of an **error**, add a single **list item** (`li`) with text in the format:
  `"Error: <error.status> (<error.statusText>)"`

Screenshots:


GitHub username: nakov

Repo: nakov.io.cin

- Svetlin Nakov: Delete Console.Cin.v11.suo
- Svetlin Nakov: Create LICENSE
- Svetlin Nakov: Update README.md
- Svetlin Nakov: Added better documentation


GitHub username: Innos

Repo: Examss

- Error: 404 (Not Found)

```
GET https://api.github.com/repos/Innos/Examss/commits 404 (Not Found)
```

Submit only the **`loadCommits()`** function in Judge System.

2. Blog

Write a program for reading blog content. It needs to make **requests** to the **server** and display **all blog posts** and their **comments**. Use the following HTML to test your solution:

blog.html

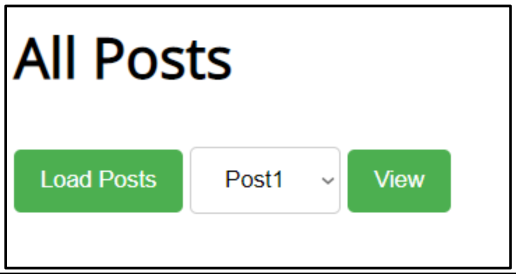
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Blog</title>
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <style>
    @import url(https://fonts.googleapis.com/css?family=Open+Sans);
    body {
        font-family: 'Open Sans', serif;
    }
    select {
        padding: 10px 15px;
        margin: 8px 0;
        display: inline-block;
        border: 1px solid #ccc;
        border-radius: 4px;
    }
    button {
        background-color: #4CAF50;
        color: white;
        padding: 10px 15px;
        margin: 8px 0;
        border: none;
        border-radius: 4px;
        cursor: pointer;
    }
    </style>
</head>
<body>
    <h1>All Posts</h1>
    <button id="btnLoadPosts">Load Posts</button>
    <select id="posts"></select>
    <button id="btnViewPost">View</button>
    <h1 id="post-title">Post Details</h1>
    <ul id="post-body"></ul>
    <h2>Comments</h2>
    <ul id="post-comments"></ul>
    <script src="solution.js"></script>
    <script>
       attachEvents();
    </script>
</body>
</html>
```

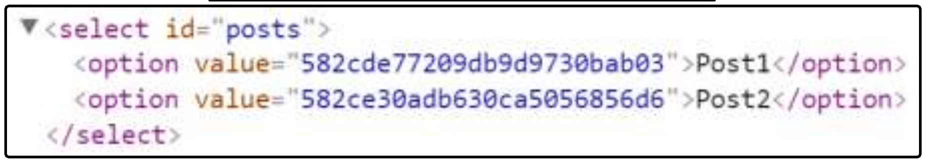
Submit only the **`attachEvents()`** function that attaches events to the buttons and contains all program logic. You will need to create a **Kinvey database** to test your code (instructions below).

The button with **`id`**="**`btnLoadPosts`**" should make a **GET** request to "**/posts**". The **response** from the **server** will be an **array of objects** in the following format:

```
{
  _id: "postId",
  title: "postTitle",
  body: "postContent"
}
```

Create an **`<option>`** for each post using its **_id** as value and **title** as text inside the node with **`id="posts"`**.



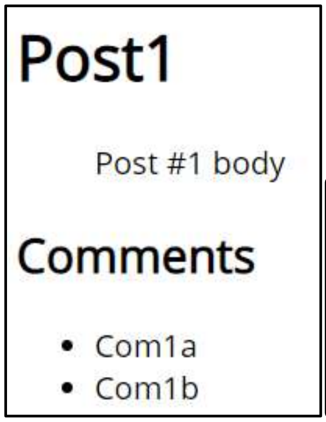


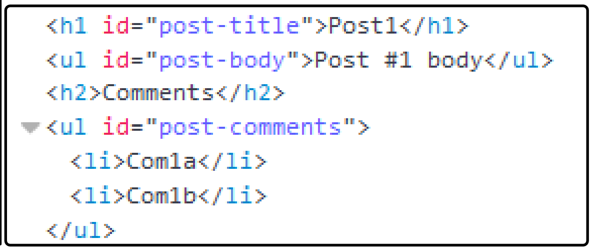
```
▼<select id="posts">
    <option value="582cde77209db9d9730bab03">Post1</option>
    <option value="582ce30adb630ca5056856d6">Post2</option>
  </select>
```

When the button with **`id="btnViewPost"`** is clicked, a **GET** request should be made to **`"/posts/{postId}"`** to obtain the selected post (from the dropdown menu with **`id="posts"`**) and another **request** to **`"/comments/?query={"post_id":"{postId}"}"`** to obtain all comments. The **first request** will return **a single object** as described above, while the **second** will return an **array of objects** in the format:

```
{
  _id: "commentId",
  text: "commentContent",
  post_id: "postId"
}
```

Display the post title inside **"#post-title"** and the post content inside **"#post-body"**. Display **each comment** as a **<li>** inside **"#post-comments"** and don't forget to clear its content beforehand.

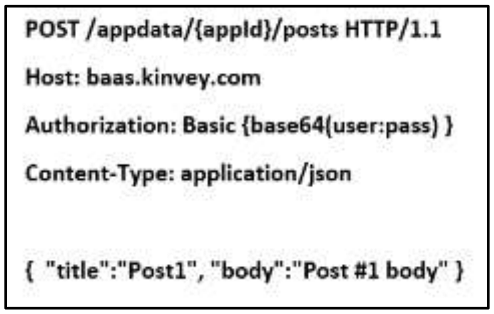




```
<h1 id="post-title">Post1</h1>
<ul id="post-body">Post #1 body</ul>
<h2>Comments</h2>
<ul id="post-comments">
  <li>Com1a</li>
  <li>Com1b</li>
</ul>
```

Hints

- Create a **Kinvey database** with the required content.
- Then create a **user** and a **password**. You will need these, along with your **app ID** for authentication.
- Use the following **POST** request to **create** blog posts through **Postman**:

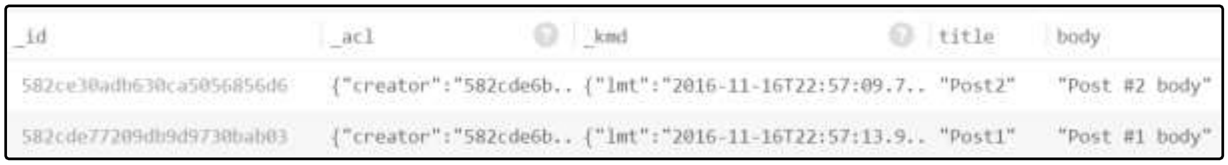


```
POST /appdata/{appId}/posts HTTP/1.1
Host: baas.kinvey.com
Authorization: Basic {base64(user:pass) }
Content-Type: application/json

{ "title":"Post1", "body":"Post #1 body" }
```

Note the **empty line** between the **header** and the **content** - the **request won't work** without it. The authorization string consists of the **username** and **password** appended together with a **colon** between them, hashed with the **btoa()** function (built into the browser). The resulting post will have an **_id** automatically assigned by Kinvey. You will then use this **ID** when creating comments for each blog post.

```
POST /appdata/{appId}/comments/ HTTP/1.1

Host: baas.kinvey.com

Authorization: Basic {base64(user:pass) }

Content-Type: application/json


{  "text":"Com1a", "post_id":"{postId}"  }
```

After the posts and comments are created, your database should look like this:



_id	_acl	_kmd	title	body
582ce30adb630ca5056856d6	{"creator":"582cde6b..	{"lmt":"2016-11-16T22:57:09.7..	"Post2"	"Post #2 body"
582cde77209db9d9730bab03	{"creator":"582cde6b..	{"lmt":"2016-11-16T22:57:13.9..	"Post1"	"Post #1 body"