Web Programming

Tutorial 9

To begin this tutorial, please create a Node project or use an existing one. Also download `tut09-starter.zip` file to get the starter code. When you finish, zip all your source codes (excluding the `node_modules` folder) to submit to this tutorial's submission box. The zip file's name should follow this format: `tclass_sid.zip` where `tclass` is your tutorial class name (e.g. `tut01`, `tut02`, `tut03`, etc.) and `sid` is your student's ID (e.g. `2101040015`).

Activity 1 – Games DB

Get the `gamedb/setup.sql` file from the starter folder. Import the `games` table from the `setup.sql` file into a MySQL database. It's recommended that you use **phpMyAdmin** in your XAMPP installation, but you can also use **MySQL Workbench** if you are comfortable with this tool.

Complete the following tasks:

1. Write a SQL query that returns the names of all games that were developed by Nintendo (your result should not have duplicates).
   (*) Helpful keywords: `DISTINCT`
2. Write a SQL query that returns the names and release year of 20 of the video games released earliest as determined by their `release_year` in the `games` table.
   (*) Helpful keywords: `ORDER BY` / `ASC` / `DESC`
3. Write a SQL query that returns the name, platform and release year of all games with the word `'Spyro'` in their title and which don't include `'Skylanders'` in their title.
   (*) Helpful keywords: `AND` / `OR` / `NOT`
   Hint: Use `NOT` to negate a boolean expression.
4. Write a SQL query that returns the average `release_year` of games in the `games` table. Use the `ROUND` function to round the result to the nearest integer and rename the column with an alias `avg_release_year`.
   (*) Helpful keywords: `AVG`, `ROUND`, `AS`
5. Write a SQL query that returns the `name` and `release_year` of the Puzzle games released in the earliest year for Puzzle games in the `games` table.
   (*) New Concepts: `Subqueries`, `MIN`

Activity 2 – Games API

1. Open XAMPP Control Panel, start Apache and MySQL.

2. Open phpMyAdmin, import the `games` table from `games.sql` and `genres` table from `genres.sql` into a database of your choice (just create a new database and name it whatever you like).

3. Create an endpoint at `/games/genres` to return a list of game genres from the `genres` table, sorted by genre name alphabetically. Following is the format of the expected result:

```
[
    {"id": 10, "genre_name": "Action"},
    {"id": 12, "genre_name": "Adventure"}
    ...
]
```

4. Create an endpoint at `/games/list/:genreid/:year` to return a list of games that belong to the specified genre and were released in the specified year. The result should be limitted to 10 games. Following is the format of expected result:

```
[
    {"id": 32, "name": "…", "platform": "…", "publisher": "…"},
    {"id": 88, "name": "…", "platform": "…", "publisher": "…"},
    ...
]
```

Activity 3 – Games Client

In your Node.js application, host static files under the `public` folder. In the `public` folder, create an HTML page which has a form. This form contains:

- A drop-down menu (`<select>`) where user can select game genres. You need to use `fetch` to get a list of genres from the API built in **Activity 1** to populate this drop-down menu.
- A text field for user to enter a year.
- A button with the text “Show games”. When user clicks on this button, the page should fetch the API built in **Activity 2** to get a list of games that belong to the selected genre and were released in the year entered in the text field.

Once you get the API's response, use JS to create a `<table>` to dislay the received data, and add this table to the page, below the form.

You should apply proper input validation on both the client side and server side.

Activity 4 – Sign-up feature with MySQL

In this activity, you'll handle a form, display error and validation messages when suitable, and insert data into MySQL database.

- Create a `users` table which has the following fields; `id`, `username`, `password`, `created_at`.
- Create a sign-up form at `/register` (GET). It has 3 fields: *Username*, *Password* and *Re-enter password*. This form submits to `/register` (`action="/register"`).
- Create a `/register` (POST) endpoint to handle the form.

Use EJS or Handlebars to build a single template to be used for both pages. Do the following form validations:

1. When username exists, show an error message right above the username field.
2. When password is empty, show an error message right above the password field.
3. When re-enter password is not the same as password, show an error message right above the re-enter password field.

When there's no error, insert the user into the `users` table and show a successful message. Retain the entered values when the submitted form contains some error. Clear the form after the user has successfully signed up.