COMP1111 Programming (Gold) 20-21

Coursework Resit

Markers: Suncica Hadzidedic (suncica.hadzidedic@durham.ac.uk)
Amitabh Trehan (amitabh.trehan@durham.ac.uk)

Hand-out date: 16 July 2021

Expected workload: 20 days, 3-4 h/day => 60 h
Components marked: code, video, documentation

Total marks: 100
Weight: 100% of module mark

Coursework submission

Deadline	**16 August 2021, 2 p.m.**
Format	Via **DUO**: Source code (all zipped in a directory with correct file structure), including: - README.txt with execution instructions and link to API documentation (if documentation is online) - HTML and CSS and any media - client and server-side JavaScript - *package.json* including test and pretest scripts - *.eslintrc* - jest test cases e.g., app.test.js - dataset - API documentation (if not online) - demonstration video You should **not** include the **directory *node_modules*** in the submission
Plagiarism, collusion	Students suspected of plagiarism, either of published work or work from unpublished sources, including the work of other students, or of collusion will be dealt with according to Computer Science and University guidelines - https://www.dur.ac.uk/learningandteaching.handbook/6/2/4/

1. Learning Outcomes

Subject-specific Knowledge

- Interaction between JavaScript programme and the Document Object Model (DOM).
- Using control statements to loop and make decisions.

- Building collections of data within a program and using JavaScript Object Notation (JSON)
- Making programs robust through the use of exceptions and exception handling
- Knowledge and understanding of good programming practice (e.g. reuse, documentation and style).

Subject-specific and Key Skills - students will be able to demonstrate:

- an ability to realise solutions to problems as working JavaScript programs
- an ability to apply reuse by exploiting predefined components
- an ability to use software tools related to programming (e.g., documentation tools)
- an ability to recognise and apply the principles of abstraction and modelling
- an ability to communicate technical information.

2. Outline of requirements

Design and develop a **dynamic website** in **JavaScript** applied to the domain of **music** (for details see Section 3 - *Instructions*). This can be a website/web application such as: digital music service, music artist's web portfolio, music contest website (e.g., Eurovision), music instruments online shop, etc.

For details about the expected website functionality see the **Instructions** section. You are to:

I. Develop a **dynamic website** in **JavaScript**. Demonstrate knowledge and understanding of the Programming module by using skills and tools taught in the module, particularly:
   - include control statements, objects and functions
   - include JavaScript classes with constructors
   - showcase the principles of abstraction and prototypal inheritance
   - apply DOM and data visualisation
   - use static HTML pages loading dynamic JSON content from server via AJAX (Fig.1)
   - write server in nodejs to provide JSON through REST API (Fig.1).

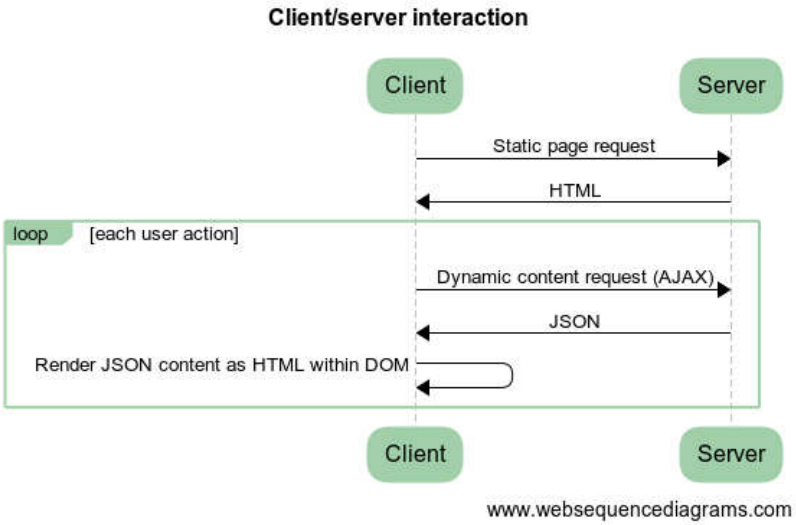

Fig 1: Server providing JSON through a REST API

II. Apply **good programming practices** by:
- using tools that support software development, e.g. web frameworks.
- demonstrating code reuse
- writing documentation
- accounting for code quality.

III. See section 3 – Instructions - for specific output/functionality/skills for each marking criterion.

3. Instructions

Overview of steps:

1. Select dataset (JSON)
2. Design HTML
3. Design web service
4. Join with Fetch

Choose a public dataset (or create one) that fits your music website scope and target audience. To make it easier on yourself, choose a smaller dataset, or reduce the dataset size.

- See examples of music datasets on: https://www.kaggle.com/search?q=music. Alternatively, search online resources for other music datasets that correspond better to your website, e.g., on:
  - AMiner; Registry of Open Data on AWS; Google Dataset Search; Awesome public datasets; opendatasoft; Microsoft Research Open Data
- Convert (the whole or part of) the dataset to JSON (if not already so) and add the JSON object to your (server side) source code. You are strongly suggested to **not use a database system.**

Your music website must have a clear purpose which is evident from/highlighted on the main/home web page. It should have one web page and be in the single page style. If you consider necessary, you can add another page for related but different functionality e.g. an admin page.

Your website (main web page) is supposed to present **interactive data visualisations** using the dataset you have selected. It should have objects/**entities** of at least **two types** e.g., songs, artists, listeners, events, places, comments, etc.

Client-side

- **Functionality**:
  - Use the music dataset to present different types of interactive data visualisations:
    - **input interface** - user should be able to input information (input text, select from options, etc.) that will influence (e.g., filter) data visualisations.
    - **output interface** - present **two** different types of interactive data visualisations, e.g.:
      a. bar or pie charts (aggregating data per e.g., music genre, artist, etc.);

b. line graphs for time trends (e.g., listening trends of a song in the last 10 years)
c. interactive maps.
- both input and output should be appearing on the same web page of your website.

- Use the music dataset to manipulate **entities** (two of your choice):
  - It should be possible to search and list/present data about the entities. You can do this through the interactive data visualisations (see above) or by adding a separate functionality for listing/searching entity data.
  - Additionally, it should be possible to **add** and **edit/update** the entities.
  - Asynchronous updates: The entities should be updated asynchronously without needing to reload the page (e.g., your visualisations should automatically change when you update the data).

- **Quality**:
  - Page(s) written in HTML+CSS (compliant with HTML5)
  - To manipulate and present the data, demonstrate skills in and use of:
    - control statements, objects and functions
    - JavaScript classes with constructors
    - principles of abstraction and prototypal inheritance
    - DOM
    - D3 and SVG
    - embedding/importing data from JSON
    - retrieving/editing data using Fetch (with GET/POST etc.)
  - Apply user interface/experience **design guidelines** (see: **link1, link2**), with focus on:
    - usability – ease of use; minimal clicks/entry required
    - responsiveness - adjusting for different devices (e.g., laptop and smartphone) and window size browsing.
    - gracefully handling server disconnection (useful error messages, recommencing on server restart).

Server-side

- **Functionality**:
  - More than one entity type, with relationships
  - REST API: Both GET and POST methods for two entities:
    - Each entity type (e.g., song) must have:
      - GET methods to list/search/get individual details (e.g., returns a list of ids and names, or other attributes/details of related entities – this can be directly reflected by updating the data visualisations).
      - POST method to add new entity
  - Installs with `npm install`
  - Starts with `npm start`

- **Quality**:
  - Should be written in *nodejs*
  - Use *npm* for management

- Make sure you use --save or --save-dev option with packages you add
- Write jest test cases: **must** run with `npm test (add to your package.json)`
- Use *Express*
- Successful jest tests with good coverage (**must** run with `npm test`):
    - Testing includes content-type and HTTP code
- Completeness of API documentation. Document your API in the style of the Twitter API (https://developer.twitter.com/en/docs/twitter-api/api-reference-index)
- Response of API calls provided as JSON:
    - Content-type needs to be correct
    - HTTP codes should be correct: use 200, 400 or 403 (if using authentication)

Code quality (client- and server-side)

- Evaluate the quality of all your code quality using **ESLint** and **fix** the identified errors.
    - Use the following to configure .eslintrc.js

```
module.exports = {
    "extends": "standard",
    "rules": {
      "semi": [2, "always"],
      "indent": "off"
    }
};
```

    - Eslint **must** run with `npm run pretest(add to your package.json)`
- **Comment** your code for maintenance purposes:
    - The main methods (objects, functions, classes) and parameters should be clearly explained in the comments;
    - Include references and acknowledgment of the source of original code and/or data (including licenses).
- Use appropriate web development **tools** and **frameworks** (Fetch, Express, Bootstrap, etc).

Video Presentation

- Submit a 2-minute (max) video demonstrating your software
- Include demonstration of how to start the program
- **All functionality (client- and server-side)** will be assessed by what is demonstrated in the video
    - If it is not demonstrated in the video, you will not get a mark for it.
- Quality of video presentation will be marked separately from functionality:
    - Structure: Visual Presentation; Audio explanation
- **You will lose 10% of marks** for video presentation (10 marks), for every block of 10 seconds over 2 minutes. That is, if your video is 2 mins 1 second long, you lose 10%, if it is 2 mins 11 seconds long, you lose 20%, and so on.

4. Marking Criteria

	Mark /100
Client-side functionality	25
Client-side quality	20
Server-side functionality	20
Server-side quality	20
Code quality	10
Video presentation	5

Testing Environment

- Windows 10 with Chrome and Mac OSX with Firefox
- Visual Studio Code
- Standard packages covered in class (npm, express etc)
    - note: you need to include *package.json* (but not the directory node_modules)