University of St Andrews School of Computer Science

CS5063 — Web Technologies — 2025/26

Assignment: P2 - YakYak
Deadline: 18-11-2025 21:00:00
Credits: 30% of module grade

MMS is the definitive source for deadline and credit details

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

1 Description

In this coursework, you have been commissioned to write the web client for a new microblogging website called **YakYak**. You have been provided with a test server that implements a Web API for you to call from the front-end web app. Messages on the website are called *Yaks*. Your task is to write an interactive single-page web application that acts as a client for the YakYak server. Users should be able to post their own Yaks and read Yaks posted by other people. They can also like Yaks and reply to other people's Yaks (called *ReYaking*).

The coursework is divided into three stages: **Primary**, **Secondary**, and **Tertiary** requirements. You are encouraged to complete them in order, ensuring that each stage is working before moving on to the next. The tasks build on each other and should be submitted as a single project.

Please see the README file inside the server directory for information about how to run the server.

You will be required to use the following technologies:

- HTML5 for the structure of the web page,
- CSS3 for the presentation of the web page,
- JavaScript for client-side programming to handle interaction and functionality on the web page.

You may optionally use **React.js** and/or **P5.js**, but you are not allowed to use any other third-party libraries, frameworks, or preprocessors.

2 Data Model

YakYak uses a data model based on **Yaks** (these are similar to Tweets or Facebook posts). Each Yak is represented as an object with the following properties:

- **id (integer):** A unique identifier for the Yak.
- **text (UTF-16 string):** The content of the Yak (maximum 280 characters).
- **created_at (UTF-16 string):** The date and time that the Yak was created in ISO 8601 UTC format.
- **author_name (UTF-16 string):** The name of the author of the Yak (maximum 64 characters).
- **likes (integer):** The number of likes that the Yak has received.
- **reply_to (integer):** An optional field specifying the ID of a Yak that this Yak was in reply to, or `null`.

3 The YakYak API Specification

The YakYak API exposes one resource: **Yaks**, available at the endpoint `/yaks`. Clients can retrieve paginated Yaks, post new Yaks, like Yaks, and post replies (ReYaks).

3.1 Pagination Model

Endpoints returning multiple Yaks use limit/offset pagination.

- **limit (number):** Number of Yaks to return.
  - Default: 10
  - Maximum: 20, if limit > 20, the server returns HTTP 400 Bad Request.
- **offset (number):** Number of Yaks to skip before the first result (default 0, must be ≥ 0).

Responses are wrapped in a paged envelope:

```
{
  "items": [ /* array of Yaks */ ],
  "limit": 10,
  "offset": 0,
  "total": 137,
  "links": {
    "self": "/yaks?limit=10&offset=0",
    "next": "/yaks?limit=10&offset=10",
    "prev": null
  }
}
```

3.2 GET /yaks

Retrieves a list of recent Yaks (newest first).

Query Parameters:

- `limit`, `offset`: Pagination control.
- `reply_to` (number, optional): Return only Yaks that reply to this Yak ID.
- `search` (string, optional): Case-insensitive substring match against Yak text or author name.

Example Request:

```
GET /yaks?limit=10&offset=0 HTTP/1.1
Accept: application/json
```

Example Error (limit too large):

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
{
  "error": "The 'limit' parameter must be 20 or below."
}
```

Example JSON Response (Pagination)

Page 1 (offset 0):

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
{
  "items": [
    {
      "id": 1012,
      "text": "Hello, YakYak!",
      "created_at": "2025-11-02T18:15:23Z",
      "author_name": "Ava",
      "likes": 2,
      "reply_to": null
    }
    /* 9 more Yaks ... */
  ],
  "limit": 10,
  "offset": 0,
  "total": 137,
  "links": {
    "self": "/yaks?limit=10&offset=0",
    "next": "/yaks?limit=10&offset=10",
    "prev": null
  }
}
```

3.3 GET /yaks/{id}

Retrieves a single Yak by ID.

Status Codes:

- 200: Successfully retrieved Yak.
- 404: Yak not found.
- 500: Server error.

3.4 POST /yaks

Creates a new Yak (or a reply if `reply_to` is provided).

Request Body:

```
{
  "text": "First Yak!",
```

```
  "author_name": "Dana",
  "reply_to": null
}
```

Response:

```
HTTP/1.1 201 Created
Content-Type: application/json
{
  "id": 1013,
  "text": "First Yak!",
  "created_at": "2025-11-02T18:20:01Z",
  "author_name": "Dana",
  "likes": 0,
  "reply_to": null
}
```

3.5 POST /yaks/{id}/like

Adds one like to the Yak with the given ID. Each call increments the count by one.

Status Codes:

- 200: Like recorded.
- 404: Yak not found.
- 500: Server error.

3.6 POST /yaks/{id}/reply

Creates a reply (ReYak) to Yak `{id}`. Equivalent to `POST /yaks` with `reply_to: id`.

3.7 DELETE /yaks/{id}

Deletes the Yak with the given ID.

Status Codes:

- 204: Yak deleted.
- 404: Yak not found.

- 500: Server error.

3.8 Error Format

All error responses use the following structure:

```
{ "error": "Short human-readable message" }
```

4 Requirements

4.1 Primary Requirements

- **Timeline:** Display a timeline showing the most recent Yaks using `GET /yaks`. The timeline should display up to 10 Yaks by default (newest first). Each Yak should show its author name, text, creation date/time, and like count.
- **Send a Yak:** Allow the user to post a new Yak using `POST /yaks`. After posting, the new Yak should appear immediately in the timeline (no refresh required). Note that the server has no notion of user account so the user can post using any username and this can be taken from an input field in the client.

4.2 Secondary Requirements

- **Like a Yak:** Allow users to "like" Yaks using `POST /yaks/{id}/like`. The like count should update dynamically.
- **Pagination:** Implement pagination so the user can view additional Yaks using the `limit` and `offset` query parameters. Use the `links.next` and `links.prev` fields to navigate.
- **Reply to a Yak:** Allow users to reply to an existing Yak using either `POST /yaks` with `reply_to` or `POST /yaks/{id}/reply`. Replies should include a small indication (e.g., " Reply to Ava") showing which Yak they respond to.
- **Delete a Yak:** Allow users to delete a Yak using `DELETE /yaks/{id}`. The deleted Yak should be removed from the timeline immediately.

4.3 Tertiary Requirements

- **View a Reply Thread:** Allow users to view a thread showing the original Yak and all replies using `GET /yaks?reply_to={id}`.
- **Search:** Allow users to search for Yaks by author name or text using `GET /yaks?search={term}`.
- **Real-Time Updates (Polling):** Implement periodic polling (e.g., every 10 seconds) using `GET /yaks` so that new Yaks and updated like counts appear automatically without requiring a manual refresh. Visually indicate new Yaks when they appear. To test this you will need to use two clients to connect to the same server.

5 Report

The report should include the following sections:

- Overview: A brief description of your version of YakYak. This section must state which of the requirements you have completed and to what extent.
- Design: Description of your design of your web application and the intent behind the choices that you made when designing and implementing it.
- Testing: A description of how you tested your web application, including any issues you encountered and how you resolved them. You should include screenshots of your application in action and any error messages you encountered in the console.
- Evaluation: This section should include one or two paragraphs evaluating the success of your submission compared to what you were asked to do.
- Conclusion: This section should include one or two paragraphs addressing what you have achieved, what you found difficult, and what you would have done if you had more time.

Word Limit: The report should be no longer than 2,000 words. The word limit for this assignment is advisory, and excludes references and appendices. No automatic penalties will be applied based on report length, but your mark may still be affected if the report is short and lacking in detail or long and lacking focus or clarity of expression. A word count must be provided at the start of the report.

6 Hints

- We recommend an incremental approach: make sure that each part of your program works before moving on to the next part. Make a safe copy of whatever you have got working so far so that you always have a working solution to submit while you are developing the next part of your solution. This should allow you to experiment with different ways of organising your code and implementing functionality without worrying of breaking things.
- Use an HTML validator and a CSS validator to validate your code.
- Use strict mode for JavaScript and check for errors/warnings in the JavaScript console of your web browser.
- It is good practice to write your HTML code, CSS code, and JavaScript code in separate files and link to the CSS and JavaScript files from within the HTML file as appropriate.
- Use the Live Server plugin for Visual Studio Code (or something similar for your editor or IDE of choice) to run your web application when developing it.
- Any requirement requiring CSS is designed to assess your ability to code using CSS rather than your graphic or interactive design skills. You should aim for a clean, simple, and consistent design but we will be grading the CSS based on the quality of the code rather than the visual appearance.

Submission

A single file containing your code and PDF report in ZIP format must be submitted electronically via MMS by the deadline. Submissions in any other format will be rejected.

The specific mark descriptors for this piece of coursework are described below.

Marking

Submissions will be marked in accordance with the School's General Mark Descriptors, which can be found in the student handbook at:
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

The following examples illustrate the typical characteristics of submissions that would fall into each mark band.

- **1–3:** Little or no working code, and little or no understanding of web technologies demonstrated.
- **4–6:** Some attempt at implementing basic functionality, such as displaying part of the catalogue or handling a user action, but the code does not run reliably or fails to meet the Primary requirements. The report shows limited understanding of the technologies used.
- **7–10:** A partially working web page that demonstrates some understanding of the main concepts but is missing major functionality from the Primary requirements. Code may be inconsistent or poorly organised, and testing is minimal or absent. The report is brief and lacks technical depth.
- **11–13:** A working implementation that successfully fulfils the Primary requirements. The report demonstrates reasonable understanding of the implementation and some evidence of testing.
- **14–16:** A complete and well-structured implementation of the Primary and Secondary requirements. Functionality is correct with only minor issues or missing features. The report includes thoughtful discussion of design decisions and systematic testing with evidence such as screenshots and validator output.
- **17–18:** A complete and well-structured implementation of the Primary and Secondary requirements, with significant progress on the Tertiary requirements. Code demonstrates excellent quality. The report is well structured and written and includes thoughtful discussion of design decisions and systematic testing with evidence such as screenshots and validator output.
- **19–20:** All required functionaity implemented correctly. The work shows exceptional design and implementation, with comprehensive documented testing. The report is extremely well structured and written and includes thoughtful discussion of design decisions and systematic testing with evidence such as screenshots and validator output.

Policies and Guidelines

Marking
See the standard mark descriptors in the School Student Handbook:

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

Lateness penalty
The standard penalty for late submission applies (Scheme A: 1 mark per 24-hour period, or part thereof):
https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good academic practice
The University policy on Good Academic Practice applies:
https://www.st-andrews.ac.uk/education/handbook/good-academic-practice/