CS571 - Summer 25 Midterm Rubrics

Please look at the table on page 2 and create rubrics for the assigned questions. Once we are done with the rubric, we'll assign the grading allocation in this table.

Grading Allocation

TA	Assigned Questions	Total
Dipsy Desai	Q1: 1–134 + Q2: 1–67	201
Syeda Tasnim Fabiha	Q2: 68–134 + Q3: 1–134	201
Sandeep Zechariah	Q4: 1–134 + Q5: 1–67	201
Shefali Tushar Kulkarni	Q5: 68–134 + Q6: 1–134	201
Tuan Ngo	Q7: 1–134 + Q8: 1–67	201
Yuehan Qin	Q8: 68–134 + Q9: 1–134	201
Pengda Xiang	Q10: 1–134 + Q11: 1–67	201
Xinyi Yu	Q11: 68–134 + Q12: 1–134	201

Rubric Allocation:

Midterm Rubrics Creation	Create a rubric for the question	Review the rubric for the question
Dipsy Desai	1	8, 12
Syeda Tasnim Fabiha	2✅	3, 9
Sandeep Zechariah	3✅, 9 ✅	1
Shefali Tushar Kulkarni	4	5, 10
Tuan Ngo	5✅, 10✅	2
Yuehan Qin	6✅, 11✅	4
Pengda Xiang	7	6, 11
Xinyi Yu	8✅, 12✅	7

Overall, let us be lenient, rather than strict. Wrong answers don't get points of course, but partially correct ones, incomplete ones, not-well-explained-but-on-the-right-track ones do get partial scores [use your judgment, and be consistent and fair].

Q1. JavaScript ('JS') is the usual (most common) language whose code runs in the browser. But we looked at, or talked about, alternatives (ie. other languages whose code can also run in the browser).

a. [2 points] name TWO such (non-JS) languages.

Answer [covered in class]: Python, Dart, LOGO. We can also accept: 'wasm' (WebAssembly), TypeScript, CoffeeScript, SQL, etc.

b. [1 point] how does this work, ie. how are non-JS languages able to be used?

Q2.

Answer: Via a 'script' tag, an included 'transpiler' [which compiles one language into another']written in JS, converts the non-JS code (eg. Python) into JS code, which the browser can execute. Simpler versions the answer (eg. that don't mention 'transpiler') are ok too.

Q2.
a. [1 point] What are 'XHR' and 'fetch'?

Solution:
XHR: XMLHttpRequest, legacy API for HTTP requests.
fetch: Modern Promise-based API for HTTP requests.

Rubrics: +.5 for each valid explanation of XHR and fetch.
The answer may not be as it is mentioned in the solution. Any correct answer should get full points. Potential correct answers may evolve around the following main idea:

Both perform **asynchronous HTTP requests** to a server **without reloading** the webpage. They let JavaScript **send and receive data** (e.g., JSON, text, HTML) in the background. XHR is older callback based where as fetch() is modern, promise-based

b. [1 point] What is their history, ie how did they originate?

Solution:
XHR: Enabled AJAX call for the first time, which is the key to dynamic web apps. It was introduced by Microsoft.
fetch: Modern alternative to XHR, which is promise-based and more powerful. It became the fetch living standard supported by all modern browsers.

Rubrics: +.5 for each valid history of XHR and fetch.
The answer may not be as it is mentioned in the solution. Any relevant answer should get full points.

c. [1 point] Discuss a use case for them [ie. how we can put them to use].
Solution:

Use case: Making asynchronous API calls to update webpage content without full reload (e.g., loading search results, submitting forms via AJAX).

XHR and fetch are used to **send/receive data asynchronously** from a server. **fetch() or XHR** sends a request for search results → response received → **JavaScript updates the DOM** without reloading the page.

Eg. we can use it to fetch a slideshow, a catalog of products, a restaurant's menu, a bunch of auction items, 3D scene data for rendering or for a videogame; we can also use it to obtain raw data (eg Big Data) for browser-based analysis or display.

Rubrics: +1 for any valid use case of XHR and fetch.
The answer may not be as it is mentioned in the solution. The solution gives only an overall idea. Any relevant answer should get full points.

Q3. HTTP 1.0 has been steadily evolving over the years. Name, and briefly discuss, three evolved versions of HTTP [ie. what feature was introduced in each].

Solution:

1. HTTP/1.1 - Persistent connections, chunked tx encoding, host headers
2. HTTP/2 - Multiplexing, server push, header compression, binary protocol
3. HTTP/3: QUIC protocol, built on UDP, improved performance on unstable n/w, loss recovery
4. HTTPS (Only if explained as a security evolution) SSL/TLS

Rubrics:

- 1 pt / version (3 version => 3 pts)
    - 1pt: Names a valid version and provides a correct feature / improvement
    - 0.5pt: Names a valid version but description is incomplete or wrong

Q4. What are three ways to style HTML elements using CSS? Provide a small example for each [OK if your CSS syntax isn't perfect but the overall answer does need to be precise].

Answer:

a. by linking to an external CSS file in <head>, eg. <link rel="stylesheet" type="text/css" href="path/to/styles.css">
b. by including a <style> 'block' in <head> eg.
   <style>
     body {
       background-color: #ff5510;
     }
   </style>
c. doing it 'inline', ie. in an HTML element, eg. ...

We can accept as correct answers, ones that don't have the exact syntax for the examples.

Q5.

a. [1 point] What are 'MIME types' for?

Solution:

- (Optional) Multipurpose Internet Mail Extensions (MIME) types are two-part identifiers that specify file and content formats (e.g., text/html, image/jpeg).
- Using MIME types is a standard way to label the nature and format of files and data transmitted over Internet protocols (to ensure that clients and servers know how to process/display the received content properly)

Rubrics:

- +1 if mention MIME types are for labelling file/data formats transmitted over Internet (e.g., between client and server).
- +0.5 if only mention MIME types are for file/data formats.
- +0 if otherwise

b. [2 points] Name two MIME types and state their purpose (ie. what they enable).

Solution:
There may be multiple MIME types, so any two from the list below (which is not exhaustive) would be acceptable, or even ones not on the list (eg. model/vrml):

- **text/plain:** Server streams raw text; client displays unchanged; and vice versa.
- **text/html**: Server sends HTML markup; client parses and renders a webpage.
- **image/jpeg**: Server streams compressed image data; client decodes; and vice versa. *[Similar for PNG or other image formats.]*
- **application/json**: Server returns structured data; client parses into objects; and vice versa. *[Similar to XML and others]*
- **multipart/form-data**: Server expects mixed parts for fields/files; client encodes inputs into parts; and vice versa.
- **application/javascript**: Server sends executable code; client parses and runs it; and vice versa.

** Other MIME types not listed above are also acceptable if correct. A full list can be found at: https://www.iana.org/assignments/media-types/media-types.xhtml*

Rubrics:

- +0.5 for each correct MIME type **and** its description
- +0.25 for each correct MIME type if **either** their name (e.g., text/plain) **or** its description is correct.
- +0 if neither is correct

Q6. At first, all HTML sent over by the server was 'static' - but today it is almost all, 'dynamic'.

Discuss three ways/techniques using which the server is able to respond with dynamic HTML to a client request.

Solution

1. Server-Side Scripting Languages
    a. PHP
    b. Python (Django, Flask)
    c. Node.js (JavaScript on the server)
    d. ASP.NETs
2. Template Engines and Server-Side Rendering (SSR)
    a. Express with EJS or Pug
    b. Ruby on Rails
    c. Django
    d. Traditional template engines (Jinja2 (Pyhton), Handlebars(JavaScript), Twig(PHP))
    e. Hybrid approaches that combine static generation with dynamic elements
    f. Edge computing
3. Content Management Systems (CMS) and Database Integration
    a. Relational database (MySQL, PostgreSQL)
    b. Content management system (WordPressx, Drupal, custom CMS)
    c. Object-Relational Mapping (ORM)
    d. Real-time data integration from APIs, web services, or live data feeds

Another way of answering this would be: via response to fetch()/XHR() requests; by doing a database lookup; by computing its response (eg random # distributions, digits of pi, ML calculations, eg. classification or LLM token gen), or

even measuring from the real-world (eg. satellite imagery, telescope images, weather, altitude...).

ANY answer that is about non-static page serving is acceptable [but should contain 3 ways, for full points].

Rubric
1 for each subpoint

Q7.

a. ‘Graphics in the browser’ - what enables this?
b. In the early days, how was graphics handled by the browser?
c. What is a reason to not prefer graphics on the browser?

Answer:

a. the ‘canvas’ HTML element
b. graphics were rendered by the browser (in a non-canvas (pre-canvas) manner, ie very simply) if the content was .gif or .jpg; for all others (eg. animation, video clips), browser PLUGINS [which the users needed to download and install] were used - including Java [where JVM code ran in an applet tag], MPEG, Flash etc.
c. graphics on the browser could slow down the page’s response drastically, if it is too intense [too much to render, slow processor, low memory etc].

Answers that express the above in simpler terms is quite acceptable as long as they are still correct.

Q8.

a. 'Frontend' for web tech is an evolving entity - eg. smartphones were not frontends when the web originated in the 1993. What is a promising newer frontend?

Solution: Augmented-reality (AR) headsets

Rubrics:

- 1 pt: Clearly explains a similar result.
- 0.5pt: vague or incomplete

b. If the web evolves to cater to AI agents too, what would that look like (ie in terms of clients and how they interact with servers)?

Solution:

- Headless AI agents as clients (no GUI, run in code)
- Semantic, machine-readable APIs (JSON-LD, GraphQL schemas)
- Hypermedia-driven workflows (HATEOAS links guiding next actions)
- Asynchronous callbacks/Webhooks for long-running tasks
- Authentication via agent credentials (OAuth client-credentials flow)

Rubrics:

- 1 pt: Clearly explains at least one.
- 0.5pt: vague or incomplete

c. What were the web's (ie WWW's) influences? You can name one, or two.

Solution:

- Vannevar Bush's "Memex" concept
- Ted Nelson's Xanadu hypertext system
- Bill Atkinson's HyperCard (on the Mac)

Rubrics:

- 1 pt: Clearly explains at least one.
- 0.5pt: vague or incomplete

Q9.

[1 point] What are 'web frameworks' (eg. React, Flutter) for [what do they provide, over raw JS]?

Solution:
Web frameworks provide abstractions, structure, and tools that simplify development over raw JavaScript. They offer component-based architecture, state management, routing, and development efficiency. Keywords to look for:

- Component reusability
- State management
- Simplified DOM manipulation
- Development speed/efficiency
- Structure/organization
- Built-in tools and utilities
- Abstraction of complex tasks

Simpler answer (also ok): they simplify HTML element creation, modification (**content** populating and updating) and deletion (state mgmt), element **styling**, element **behavior**.

Rubrics:

- 1 pt: Clearly explains atleast one major benefit of frameworks.
- 0.5pt: Attempts to explain but vague or incomplete

[2 points] What was one of the very first frameworks [it is still in use although not heavily)? What notational simplification did it provide (ie. code shortcut)?

Solution:
jQuery was one of the first widely-adopted frameworks. Its notational simplification was the $ function for element selection and manipulation (e.g.,

$('#id') instead of document.getElementById('id')). Other examples of very first frameworks that are also acceptable:

- Prototype
- MooTools
- Dojo

Rubrics:

- 1 pt: Correctly identifies jQuery (or another valid early framework like Prototype, MooTools), look for “$” if there is a “$” its a good answer.
- 0.5pt: Names a framework that's older but not among the first
- 0 pt for modern frameworks like react / angular

Q10. ‘Literate programming’ involves notebooks (mix of static ie non-code content, and code). What are 3 literate programming examples we covered [including non-web ones]?

Solution:
(Optional explanation) Literate programming refers to notebook-style documents that mix prose/markdown, media/math, and executable code with results.

Three examples covered in CSCI 571’s lecture (i.e., on D2L website, CSCI 571 Lecture Video 7/15/2025):

- Jupyter Notebooks
- Google Colab
- Wolfram/Mathematica Notebooks

Add’l answers that are ok: LaTeX (including Overleaf and ASCIIMathML), MATLAB notebooks, ObservableHQ (JS notebooks).

Rubrics:

- +0.5 for each correct example (no description required or graded)
- +0 if no example is provided or if all the examples are incorrect

Q11.

[1 point] How do 'URL queries' [sending parameter values] work?

Solution

A URL query **appends key–value pairs to the end of a URL after a question mark (?)**. These **pairs are separated by &**. When the client requests this URL, the **server receives those parameter values** and can use them to decide what data or page to return.

Rubric

+0.5 for each mentioning of [append key-val pairs after question mark], [paris separated by &], [server receive param vals], 1 max

[2 points] What is the name given to a wildly popular form of URL-based request (query) sent to a server? Provide an example (no need for exact syntax).

Solution:
'REST API' calls.
Eg: https://api.openweathermap.org?lat=35&lon=-118
Or https://weather.com/weather/today/l/Los+Angeles+CA

Rubric

1. +1 for 'REST' or 'REST API' [0.5 for GET or POST]
2. +2 for any example that contains a ? [which separates the URL and the REST part] and at least one key=value; OR an example that uses / instead

Q12. The ‘Holy Trinity’ (or ‘triumvirate’) of web tech is what what? Name them AND discuss their purpose (ie what each provides).

Solution:

- HTML (HyperText Markup Language)
  Purpose: Defines the structure and semantics of web content.
- CSS (Cascading Style Sheets)
  Purpose: Controls the presentation—layout, colors, typography, spacing—of HTML elements.
- JavaScript
  Purpose: Implements behavior and interactivity on the client side.

Rubrics:

- 0.5pt each for correctly pointing out three items
- 0.5pt each for correctly pointing out their purposes