CS 142 Final Examination

Spring Quarter 2023

You have 3 hours (180 minutes) for this examination; the number of points for each question indicates roughly how many minutes you should spend on that question. Make sure you print your name and sign the Honor Code below. During the examination you may consult two double-sided pages of notes; all other sources of information, including laptops, cell phones, etc. are prohibited.

I acknowledge and accept the Stanford University Honor Code. I have neither given nor received aid in answering the questions on this examination.

____
(Signature)

Solutions
____
(Print your name, legibly!)

____@stanford.edu
(SUID - Stanford email account for grading database key)

Problem	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
Points	10	10	10	8	8	8	12	16	8	12
Problem	#11	#12	#13	#14	#15	#16	#17	#18	#19	Total
Points	10	10	8	8	8	8	8	10	8	180

Only the front side of the exam pages will be scanned. Do not write answers on the back of the pages.

Problem #1 (10 points)

As is widely done in computer networking protocols, the HTTP protocol is layered on top of the TCP/IP protocol. There are further protocol layers both below TCP/IP and above HTTP. For example, below TCP/IP are network and data link layers like Ethernet and WiFi.

A) Describe one of the layers above HTTP that were presented in class.

REST is a protocol layered on top of HTTP. REST specified model data operations in terms of HTTP requests/responses

B) Sometimes when you have layer A built on top of layer B, it is possible to implement layer B on top of layer A. For example, it won't be efficient but you could implement Ethernet by sending and receiving packets over a TCP/IP connection. Would it be possible to implement TCP/IP on HTTP? Explain your answer.
Given that TCP/IP is bidirectional (data transfers flow in both directions symmetrically), implementing it on top of a request-response protocol is going to be tough. In the client-to-server direction, it would be straightforward to transfer TCP/IP data in PUT or POST requests. The other direction (server-to-client), there isn't a way for an HTTP server to send data to the client unless there is a request outstanding it could add it to the response going back to the client. If you wanted to try to have a response outstanding you could do something like have the client continuously poll (e.g. GET requests) for data being transferred from the server to the client but that solution might be considered too resource expensive to be used.

Problem #2 (10 points)

Referential integrity was a serious concern in the early HyperText system. Broken references to pages that were deleted or moved were viewed as unacceptable. How did the HTTP designers envision handling references that:

A) Move to another location.
The HTTP designers envisioned handling references that moved to another location through the use of **HTTP redirection**. When a resource is moved to another location, the server can respond to requests for the original URL with an HTTP status code indicating the move ("301 Moved Permanently" status code). The server also includes the new URL where the resource can be found in the response headers. When the client receives the redirection response, it automatically follows the provided new URL to retrieve the resource from its new location. This redirection process ensures that references to the moved resource are updated, and subsequent requests are sent to the correct location.

B) Are deleted.
HTTP didn't attempt to maintain referential integrity so a client could send a request to a location that no longer has the content (i.e. was deleted). The HTTP protocol would return 404 Not Found in this situation.

Problem #3 (10 points)

The DOM's `XMLHttpRequest` class allows JavaScript to perform HTTP requests. In lecture, we presented the basic usage pattern:

```
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

A) The documentation for the `open` method reports an additional optional parameter named async (e.g. `open(method, URL, async)`). The documentation reads:

> `async` An optional Boolean parameter, defaulting to true, indicating whether or not to perform the operation asynchronously. If this value is false, the send() method does not return until the response is received.

The use of this feature of `XMLHttpRequest` is strongly discouraged. Describe the reason for this and describe what the problem would be if you modified your PhotoApp to use async=False rather than Axios to fetch model data.

Setting async to false can create unresponsive user interfaces and, thus poor user experience, especially when dealing with slow network, since it allows for an HTTP request to block all other operations on the PhotoApp until a response is received. For example, if uploading a photo to the PhotoApp takes a long time because it's a large file and the user's network is slow, this will make the app looks like it's frozen since other features like navigating to another page, or adding a comment won't work until the photo upload is complete.

B) Explain why URLs that would work if entered in the browser's location bar aren't guaranteed to work if passed to an `XMLHttpRequest` request.

XMLHttpRequest follows the same-origin policy, which restricts requests to only the same origin (protocol, domain, and port) from which the JavaScript code originates. This security measure is in place to prevent malicious scripts from accessing resources of different origins.

Problem #4 (8 points)

Although our simple React.js Photo App had components individually fetch their model data, it is possible with a JavaScript framework using model/view/controller decomposition like React.js to collect together all the component model data fetches of the current view and process them together. Would such an extension to collect the fetches together be more advantageous for REST or GraphQL backend APIs? Explain your answer.

GraphQL allows fetches of multiple resources in a single GraphQL query. This capability allows the extension to combine all the model data fetches into a single round-trip to the server. REST fetches a single resource per operation, so even with this extension, REST would have to loop through and perform the round-trip fetch for each model data fetch. GraphQL is in a position to take advantage of the extension, whereas REST isn't.

Problem #5 (8 points)

Assume you are given two JavaScript functions that return Promises named `f1` and `f2` and the following JavaScript function:

```
async function runIt(a) {
  let f1result = await f1(a);
  let b = await f2(f1result);
  console.log(b);
}
```

Your boss decides your code can no longer use the `async` and `await` keywords. Write a version of the function that doesn't use these keywords. Your code should print the same `console.log` statement when called with the same parameter.

We need to resolve f1 and then pass its results to f2. We need to resolve f2 and pass it to console.log:

```
function runIt(a) {
   f1(a).then(function (f1result) {
       return f2(f1result);
   }).then(function (b) {
      console.log(b);
   });
}
```

Problem #6 (8 points)

Node.js embedded a JavaScript runtime. JavaScript has a full-featured `String` data type and supports garbage collection but, until recently, didn't have good support for arrays of bytes, which are widely used by the system call interfaces of modern operating systems. Encoding an array of bytes into a JavaScript string in the JavaScript heap is a resource-intensive operation.

Much of the processing of web servers involves moving arrays of bytes between system calls. For example, processing a GET request for a file involves invoking a read system call to get an array of bytes and passing that array of bytes to a socket write system call. Explain how a web server written in Express.js can service GET requests without paying the cost of encoding the arrays of bytes into JavaScript strings.

Node.js introduced the Buffer class to the JavaScript environment to address this issue. The system calls all operations of Buffer objects that store the data outside of the JavaScript heap in the memory of the Node.js server. A web server processing a GET request reads the data into a Buffer and sends it out the network without having to convert it to JavaScript strings.

Problem #7 (12 points)

Our PhotoApp backend used multiple Express middleware modules. List three of the modules, including a description of what the module does and what functionality would break if the module wasn't used.

- express session - this module stores user's log-in credentials server-side and send back a cookie. if we took it out, we wouldn't be able to authenticate.
- express static - this module serves static files like JS and CSS files. if we just commented it out, we wouldn't be able to send code files to the browser to render (?)
- express body-parser -- this module parses requests coming in from the browser from bits to JS. we wouldn't be able to send non-empty bodies, so post requests like logging in and adding photos/comments wouldn't work.

Problem #8 (16 points)

A)
```
// A
app.put('/update', function (req, res) {
  // B
  User.findOne({_id: req.params.user_id}, function (err, user)
{
    // C
    user[req.body.prop] = req.body.value;
    // D
    user.save();
    // E
    res.send('success');
  });
  //F
});
// G
```

Although JavaScript has a single thread of control, not all JavaScript code can assume no other JavaScript execution can be performed between two points in the code. For each of the following pairs of letters, state if other JavaScript execution could be performed between the letters.

A,B Yes, B runs in the express handler and A is when the handler is registered. There is an opportunity for other functions to run.

B,C Yes, C runs when the Mongoose query finishes and C is when the query is launched. There is an opportunity for other functions to run.

C,D No, There is a single assignment between these points. No other JavaScript functions could run.

D,E No. This is tricky since the user.save() returns a Promise that we are not waiting for so no other JavaScript function could run.

A,G No. These points are before and after we register the express handler. No other JavaScript function could run.

B,F No. These points are before and after we launch the query. No other JavaScript function could run.

Problem #8 continued….

B) If there are no other requests active in the system, what can the requestor in the browser assume about the state of the database when the response is received? Can the requestor assume the database has been updated or might the database still hold the old value of the User object?

The HTTP response is sent out after the `user.save()` call returns. Since that call returns a Promise we know the database update operation has launched but we don't know if the Promise has been resolved and the update finished. Similarly, we don't know if the response gets to the requestor before or after the database update finishes. Hence the requestor can't know if the database holds the old or the new value of the User object. It would not be safe for the requestor to assume the database has been updated.

Problem #9 (8 points)

Describe the issues with the SQL databases that lead to web applications preferring no-SQL databases like MongoDB.

Traditional relational databases took a relatively static view of the schema with operations like adding or subtracting properties to/from a table assumed to be rare. Web developers prefer a more dynamic capability of being able to add properties to the model data returned for the views. The flexibility offered by databases like MongoDB that stores JavaScript-style objects attracted fans.

Problem #10 (12 points)

The REST APIs allow CRUD operations on single resources named by the URL. The HTTP verb specifies which of the CRUD operations to perform. A REST CRUD operation can result in the resource being changed in the database and potentially database indexes being updated.

For each of the HTTP verbs below, state if the operations would never update an index, update a single index only, or potentially update multiple indexes.

A) GET For security reasons, GET operations should not cause mutations, so you would not expect app data indexes to be updated.

B) PUT A PUT operation in REST updates a resource which could lead to multiple indexes being updated.

C) POST A POST operation in REST creates a resource which could lead to multiple indexes being updated.

D) DELETE A DELETE operation in REST deletes a resource which could lead to multiple indexes being updated.

Problem #11 (10 points)

Both the UI design using the MVC structure and the Mongoose ODL used the same word, "model", to describe a collection of data. Even if the two parts (MVC and Mongoose) are referring to the same concept (e.g User in our PhotoApp) they might not have the same property.

A) Is there ever a possibility that the MVC model is a subset of the properties of the Mongoose model? If so, give an example.

Certainly, the /user/list GET operation in the PhotoApp is an example. The User model in /user/list only contains a subset of the User model in Mongoose. The MVC model for example may only return id, first name, last name whereas the Mongoose model contained that and username, location, description, etc.

B) Is there ever a possibility that the MVC model is a superset of the properties of the Mongoose model? If so, give an example.

Yes, The /photosOfUser/ is an example. The MVC Photo object has more properties than the Mongoose Photo model. The user photos object in project 8 for example can be augmented with information such as “is_favorited” or “is_deleted” making the MVC model a superset.

Problem #12 (10 points)

Assume you download a new version of the Chrome browser and it has an unfortunate bug of sometimes forgetting to attach the origin's cookies to HTTP requests sent out. Assume this bug occurred infrequently but did happen at least once an hour.

A) Describe what kind of failures (you would see) if you ran your PhotoApp on this buggy browser.

   The bug would cause requests sent from our frontend to the web server to not have the session cookie infrequently. Model data accesses (mostly GETs) are most of the requests we are likely to use so they would likely hit the bug. The failure of the model data fetch or other operation will be detected by the component and any view rendered will be incomplete or missing updates. Our fetches of files (webpack bundle, images, CSS, etc.) doesn't check the cookie so those would be unaffected by the bug.

B) Is there a simple workaround you could suggest to the user when they hit the bug that would allow them to continue using the application, or would they need to get a browser to get any use of the PhotoApp?

   Given the infrequent nature of the problem, simply trying the model data fetch again would likely correct the problem when the model data fetch was retried with the session cookie. You could suggest to the user a workaround when they hit the bug. They could move away from the view and move back or refresh the browser to trigger model data to be fetched again with the hope this has the cookie attached this time.

Problem #13 (8 points)

The base functionality of the PhotoApp didn't change the session state except during login and logout. Assume you do a Project 8 story that makes most Express handlers update the session state (e.g., keep a display of an estimated number of logged-in users). Would you expect Express session to have to respond with a `set-cookie` header on most responses? Explain your answer.

The set-cookie is used by Express session to store a pointer to the session state stored in the backend. This pointer is returned when the session state is first created (the /admin/login POST). No update is needed to the pointer if the session state is updated. We wouldn't expect to see additional set-cookies if session state changes frequently.

Problem #14 (8 points)

One of the exciting new features of React.js when it was introduced was rather than rendering fully in JavaScript in the browser, the developer has the option of rendering the view into HTML that is downloaded and displayed in the browser. This was called server-side rendering and it became a must-have capability for JavaScript frameworks. Describe what problems might occur if you used server-side rendering for views that take user input.

With server-side rendering the React components are rendered into HTML that is downloaded from the web server to the browser to be displayed. The fast validation and signaling of input problems possible with JavaScript frameworks wouldn't be available.

Problem #15 (8 points)

When using HTTPS to communicate between your browser and your backend web server, who knows the encrypt key being used? Choose from the list:

- Browser,
- Web server,
- Certificate authority, or
- Someone else.

Explain your answer.

The key is generated by the Browser and sent to the web server using the web server's public key. Only the web server should be able to encrypt and read this key. No one other than the browser and web server should know the key.

Problem #16 (8 points)

Explain why cross-site request forgery (CSRF) attacks can happen on certain HTTP verbs that are used in REST APIs (like GET and POST) but not as likely for other verbs (like PUT and DELETE).

It is relatively easy for the attacking page to trigger a GET request to our website using a link or form and it is also easy to trigger a form POST submit. Other HTTP verbs would require using XMLHttpRequest that falls under the same-origin policy. So unless the same origin policy is incorrect for the page, the attacking page can't use these other verbs.

Problem #17 (8 points)

Describe how message authentication codes (MACs) can provide integrity and authentication but not confidentially.

Integrity and authenticity are achieved using cryptography and a shared secret. Only those who know the shared secret can generate a valid MAC. This ensures that trusted recipients can verify that the message was unaltered and was from a trusted and legitimate party. For example, a web server could use this to verify that the session cookie it gave to a browser was not altered and actually authored by the server. Note that there is no confidentiality provided in this scheme as the data is not encrypted. For example, a malicious third-party could intercept the plaintext messages and view them in their entirety.

Problem #18 (10 points)

Answer the following questions about Content Distribution Networks (CDNs).

A) Describe the issue that makes CDNs not suitable for rapidly changing content.

CDNs serve static content from a distributed network of servers meaning that it would take a significant amount of time for rapidly changing content to propagate throughout the network.

B) What would be the problem seen if you put content that was updating frequently?

The frequently updated content may not be seen as quickly as you would like or anticipate compared to a normal web server, so the clients of your application may see stale and outdated content.

Problem #19 (8 points)

Describe the attributes of a scale-out web server architecture that make it easier to handle large variability in load compared to scale-out data storage systems.

In a scale-out web server architecture, each server is independent and ideally stateless. This allows requests to be evenly distributed among multiple servers with ease. In a scale-out data storage system, the database itself is spread across multiple nodes which is known as data sharding. Since the load is balanced across the database, more requests can be handled overall, but individual shards may become overwhelmed which can lead to an overall load imbalance. Additionally, depending on how the data is sharded, it can be more complicated to perform queries and aggregate data from multiple shards.