CS 142 Final Examination Solutions

Winter Quarter 2019

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
Score									
Max	15	8	16	10	15	12	8	8	10
Problem	#10	#11	#12	#13	#14	#15	#16	Total	
Score									
Max	10	15	15	10	12	8	8	180	

Problem #1 (15 points)

Content distribution networks and Memcache servers are both storage systems commonly used increase performance and scale in large scale web applications. Although it would require changing some things, it would be relatively straightforward to remove one or both of the systems and have web app continue to function.

(a) Show your understanding of Memcache by describing what the problems or disadvantages you would see if you removed Memcache from a large scale web application. Your answer should specifically indicate what functioning would be affected.

Answers that got full credit made some mention to the fact that the app would become slower AND gave or mentioned an example (session state as a canonical example). Some points were given (or taken away) depending on if answers demonstrated a basic understanding of Memcache.

(b) Show your understanding of Content distribution networks by describing what the problems or disadvantages you would see if you removed Content distribution networks from a large scale web application. Your answer should specifically indicate what functioning would be affected.

Answers that got full credit made some mention to the fact that the app would become slower AND gave or mentioned an example (loading static images). Some points were given (or taken away) depending on if answers demonstrated a basic understanding of CDNs.

(c) Both Content distribution networks and Memcache place limits on the kind of data it is appropriate to store in them. Describe the limitations for each of the systems.

Most people had the right idea for CDNs being used for static content. Full points were awarded to those that discussed the fact that Memcache is an in-memory store, and as such, is fast but can be unreliable. Some points were given for mentioning other obvious characteristics of Memcache (key-value store, being used for frequent queries, etc).

Problem #2 (8 points)

In class, it was recommended that a web application validate user input both in the front-end and back-end. A manager of the web app you are working on decided in a rush to market to only validate input in one of the two places. Which place (front-end or back-end) would you suggest doing the input validation if your web application could only have one? Justify your answer and include a discussion of the disadvantage of losing the input validation you choose to omit.

Though subjective, the “right” answer is to validate on the backend. Most points to this question were awarded if the answer included an adequate discussion of the benefits and drawbacks of each (losing front-end validation would result in a poorer user experience, but forgoing backend validation would mean vulnerability to adversarial inputs).

Problem #3 (20 points)

In JavaScript, functions that block (e.g. access the database or wait for the user) need to be treated differently from ones that don't block. For example, if you had non-blocking functions (`op1` and `op2`) and wanted to pass 10 to `op1`, process the result with `op2`, and the log the output you could write:

```
var arg = 10;
var result = op2(op1(arg));
console.log(result);
```

Callbacks

If `op1` and `op2` are blocking functions the programmer needs to change to use callback functions to return the result. Assume now that `op1` and `op2` now use callbacks to return results (e.g., `op1` function signature is now `op1(arg, callback)` where `callback` is a function that is called with the result of apply `op1` to `arg`).

(a) Write code that has the same functionality as above (i.e. console.log result) but uses the callback versions of op1 and op2.

```
op1(10, function(result) {
    op2(result, function(result2){
        console.log(result2);
    });
});
```

A variety of permutations of the above were seen. Some points were taken off for more or less acceptable answers. All points were taken off if no allusion to the asynchronous nature of callbacks was made (e.g., if the result of the functions was stored in a variable and called with console.log, and console.log was not included in a callback function).

(b) Assume that the original op1 was a simple increment by one function (i.e. function op1(arg) { return arg+1; }). Show what the corresponding code would be for the callback version.

```
function op1(arg, callback) {
    callback(arg + 1);


}
```

In this situation, you are a client of callback so you don't need to define it.

problem #3 continued….

Promise

Advocates of using Promises to handle blocking functions in JavaScript point to the less disruptive coding patterns required if all variables are Promises. Assume now that `op1` and `op2` now return promises as results (e.g., `op1` function signature is back to `op1(arg)`). The Promise advocates rejoice in the main part of the code:

```
    var result = op2(op1(arg));
```

being identical when `arg` and `result` are promises.

(a) Write code that has the same functionality as the original (i.e. console.log result) but uses the Promises everywhere. Hint: In order to get the main part described above working you need to create a promise that returns 10 (a promise version of `var arg = 10`) and console.log a promise.

// var arg = 10 using a promise:

```
var arg = new Promise(function (fulfill, reject) {
   fulfill(10);
});

var result = op2(op1(arg));
```

// console.log a promise

```
result.then(r => console.log(r));
```

(c) Using the same functionality assumption as part (b), show what the promise version of op1 would look like. Recall that `arg` is now a promise

```
     function op1(arg) {


          return arg.then((v) => v+1);




     }
```

Problem #4 (10 points)

(a) The word **state** was used several times during the course. We talked about (a) **state** management systems like Redux, (b) session ***state***, and (c) ***state***less web servers. How are the values that make up these states similar and/or different? Be sure to include examples of state value in your answer.

The state of state management is the model data of the components that make up the view of the web app. The session state is shared between the backend and frontend associated with the session. It is certainly possible that there is overlap between information that is in session state and what is displayed (i.e. model data). The state in stateless web servers refers to state (like session state and model data) that would preclude doing load balancing. So state like who is logged in could be found in the (a) state management and (c) session state and not want to be kept in web servers.

(b) In class we talked about (a) server**less** computing, (b) state**less** web server, is there any relationship between these two words with **less** in them? Justify your answer.

The word "less" refer to different things: the abstraction of servers in serverless computing and session state in stateless web servers but a possible connection is the same reason we have stateless web servers (load balancing) is also features in serverless computing where the cloud provider would like to be able to run the user provided functions are any server.

Problem #5 (15 points)

(a) Web app-based discussion forms frequently define their own markup language (like the Markdown example given in class) rather than simply using HTML and having the browser's rendering engine do the markup. Besides syntax complexity disadvantage of HTML over something like Markdown, describe the key security problem with using HTML as a markup language in a web application.
Answers got full credit for discussing how this would leave you open to an xxs attack. Partial credit was given for identifying other security problems or somewhat identifying an xxs attack

(b) While doing a security audit on a web application, you see an HTML tag that looks like:

```
<img src="http://www.yourcompany.com/public/logo.jpg" />
```

The web application itself is served with https but the HTML coder felt that the company logo was public and didn't need encryption protection. Describe the problems with this reasoning.
Answers got full credit for saying how this leaves the website open to a man in the middle attack, and that the cookies are vulnerable in the http request and the attacker can now forge valid looking requests. Partial credit was given for some indication of the above answer.

Problem #5 continued...

(c) Most browsers will happily load HTML, CSS, JavaScript, images, or any other type of file from the local file system of the machine running the browser. This means JavaScript frameworks like Angular, ReactJS, and VueJS can run from the local machine. Explain why this setup doesn't work for programs that load content from a content distribution network.

Answers got full credit for discussing how a CDN works by using special URLs that map to locally placed servers and that it would therefore use something over the “file” protocol for requests, and that running the web app from files means you are using the file protocol making the CDN in a different origin. Therefore this would cause the browser to block requests to the CDN. Answers got partial credit for indicating an understanding of how CDN and file protocol system would clash.

Problem #6 (12 points)

You are evaluating using a REST API or GraphQL for your web app backend. For each of the following metrics, state which of the two would do better on the metric. Be sure to include justification in all of your answers.

(a) Which of the two would result in fewer round-trip communications between the browsers and the web server? Justify your answer.
Answers got full credit for saying something along the lines of how GraphQL can do multiple requests in a single request whereas REST requires multiple for different types of resources. Answers got partial credit for saying GraphQL with another justification.

(b) Which of the options might have unnecessary values communicated from the web server to the browsers? Justify your answer.
Answers got full credit for saying RestAPI because for GraphQL you can specify properties you want to request from a resource. Answers got partial credit for saying RestAPI with another justifcaiton

Problem #6 continued...

(c) Which would be easier to load balance? Justify your answer.

Answers got full credit for saying RestAPI because GraphQL would group together multiple requests making it harder to load balance and tell how much work an incoming request is. Answers got partial credit for saying RestAPI with another justification

Problem #7 (8 points)

In Project 7, we had you add session management functionality to your photo sharing app. In the starter code we provided, we had you include the line:

```
app.use(session({secret: 'secretKey', resave: false, saveUninitialized: false}));
```

Explain what secret: ‘secretKey’ does, and why it is important we use it. Describe what could go wrong if it was known your web app secret was 'secretKey'.

secret: ‘secretKey’ is used to cryptographically sign the cookie session ID coming from the request, so we can validate a request’s session and retrieve the associated server-side session data. If our secret became known, then our server would be vulnerable to malicious attacks, because an attacker could send requests that would hash to something looking like a valid session ID.

Problem #8 (8 points)

In the ReactJS version of the projects, we had components communicate by explicitly passing callback functions between components. If component A needed to know something happened in component B, A would define a function and pass it to B which called the function when the something happened. Describe how the Listen/Emitter pattern we used in AngularJS handles this kind of communication more cleanly.

Unlike ReactJS, AngularJS has built-in Listen/Emitter functionality that allows us to use functions like $broadcast and $emit to emit/signal event's from one component's controller (B in this case) and $on to listen/receive these signals in other components' controllers (A in this case). This is cleaner than the callback solution in ReactJS because we don't have to pass functions from component-to-component (can get messy quick), and can instead just define the functionality we need within each component.

Problem #9 (10 points)

(a) Give a brief description of the role of Middleware in ExpressJS.

We can use middleware to design functions (or use built-in Express middleware/third-party middleware) that can read and write the request and response objects before reaching our actual API endpoint. Middleware is useful for doing any sort of validation/housekeeping before the request reaches the actual endpoint.

(b) The webServer.js program used in the class projects utilized several ExpressJS middleware packages. The initial version of webServer.js given with Project #4 uses a middleware package and several other ones were added in later assignments. Describe the functionality provided by middleware modules in the class webServer.js including the middleware used in Project #4 and at least one of the other middleware modules added in later projects.

In Project 4 we used Express' built-in express.static middleware. This is what allowed us to serve static content (CSS, images) in our photosharing app. Other middleware used in later projects included express-session (session management), body-parser (parsing POST requests), multer (handle file uploads)

Problem #10 (10 points)

The domain name system (DNS) is used in web applications.

(a) Describe how DNS can be used to do load balancing across a web application's servers.

Instead of returning a single IP for each domain name, the DNS server can choose an IP address from a list of IP addresses. Typically this is done in a round-robin fashion and can be different for each DNS server based on geographical location. In this way, web traffic for a particular domain can be distributed to multiple IP addresses.

(b) Describe the advantage that a load balancing network switch would have over using DNS is this way.

A load balancing network switch can (1) almost instantly change what endpoints are receiving requests (useful in machine failure), (2) balance load based on response time, liveness, or other metrics, (3) direct traffic to a much larger number of machines (compared to a DNS server) and (4) inspect session cookies to keep a user's requests directed to a particular server.

Problem #11 (15 points)

(a) Describe an attack on a web application backend that can be done using the web application session cookie even though the attacker has no way of reading the session cookie.

Cross site forgery (CSRF) leverages the fact that cookies are sent with all HTTP requests to the server that issued the cookie. This means that if an attacker can inject javascript into a page (or get you to visit a bad domain) they can send a request to the website they're interested in stealing your credentials for (a bank website), they can leverage your stored cookies to validate their request without actually reading the session cookie.

(b) Explain why it is much easier for a man-in-the-middle attacker to switch a web app using HTTP to use HTTPS than it is to switch a web app using HTTPS to use HTTP.

The switch from HTTP to HTTPS is a process of *encryption*, meaning the man-in-the-middle just needs to invent a key to encrypt data sent to the server and decrypt data returned from the server. The switch from HTTPS to HTTP is a much more difficult process of *decryption*, meaning the man-in-the-middle would need to figure out the encryption keys the server and browser.

problem #11 continued...

(c) Does HTTPS protect the URL of the HTTP request from the browser to the web server from a man-in-the-middle attack? Explain your answer.

Anser #1
Yes, after the initial handshake, the request headers, body, and URL path and query parameters are encrypted and can't be read by a man-in-the-middle.

Answer #2
No, because the URL must be sent to a DNS server to be resolved and won't be encrypted at this time. Even after the DNS resolving the destination IP, the domain name will continue to remain public so that packets can be correctly routed.

Problem #12 (15 points)

(a) MongoDB objects are stored in BSON (a binary version of JavaScript JSON format). We used Mongoose to enforce a schema on the objects. Explain the advantages of a database with a schema compared to JSON for returning model data.

While JSON blobs are super flexible, a schema assigns property names and their types to records in a collection. This makes it easier to validate the data being entered in the database, provides an interface for making queries and allows for indexes for more efficient database operations.

(b) Explain why databases support multiple secondary indexes but only one primary index.

The primary index ensures that each data record has a unique id in the database and is used to structured/organize the records. Secondary indexes are used for performance optimizations for database queries and usually result in faster read times, but slower writes.

(c) Although Mongoose's syntax and functionality look like old ORM systems, explain why it is incorrect to call our use of Mongoose with MongoDB an ORM.

Mongoose creates a layer on top of MongoDB that makes it feel like a traditional ORM, but under the hood, MongoDB is still a NoSQL database that stores records as documents in collections instead of rows in tables.

Problem #13 (12 points)

In Node.js we could process all the elements of an array using the JavaScript array's `forEach` method like so:

```
      ['A', 'B', 'C'].forEach(processItemFunc);
```

We also introduced the `async.each` interface for processing all the elements of an array like so:

```
async.each(['A', 'B', 'C'], processItemWithCallbackFunc, doneFunc);
function doneFunc() {
      console.log('Done');
}
function processItemWithCallbackFunc(letter, callback) {
      …
      callback();
}
```

(a) What was the advantage of using a callback interface over the simple function call of `forEach`?.

The callback interface allows async.each to be non-blocking and process each item in parallel, whereas forEach is blocking (synchronous) and runs in series. This is particularly efficient when each call of `processItemFunc/processItemWithCallbackFunc` requires waiting.

(b) Describe what is executed when the line `callback()` is executed. Include a description of what the code does.

callback() is a function defined by the async library. `processItemWithCallbackFunc` calls this function to notify async that this item has finished processing. Async waits for every item to call callback() before calling doneFunc() once.

Problem #14 (12 points)

The HTTP protocol allows communication between the browser and the web server and this communication can be used to inform the opposite side to start doing something different or inform the other side of something.

(a) Explain what `Connection: keep-alive` in an HTTP request header tells the web server to do differently?

This tells the server to keep the connection open after sending the response.

(b) Explain what `Cache-Control: max-age=120` in an HTTP response header tells the browser to do differently?

Tells the browser that this response can be cached for 120 seconds before the data needs to be refreshed from the server.

(c) Explain what an `X-XSRF-TOKEN` line in an HTTP request header tells the web server?

X-XSRF-TOKEN holds the cookie for the current session, and protects against CSRF attacks.

Problem #15 (8 points)

The DOM in modern web browsers allows communication to the web application backend using with XMLHttpRequest or the new WebSockets API. Describe the advantage that using WebSockets would have over XMLHttpRequest when being used by a web application to talk to its backend?

Websockets allow for fast, secure, and unstructured two-way communication. The browser and server don't need to adhere to the strict request/response format. Notably, this allows the server to push new data to the browser without the browser needing to request it.

Problem #16 (8 points)

Early web application frameworks like Ruby on Rails did all view rendering on in the web server with the view rendered into HTML that is shipped to and displayed in the browser. JavaScript web app frameworks like AngularJS render the view directly into the browser's DOM from JavaScript running the browsers. The current generation of JavaScript web application frameworks like ReactJS/Angular/VueJS decouples the rendering from the browser's DOM which allows the rendering to be done either in the browser or in the web-server (ie. serverside rendering). Given all the advantages of having JavaScript directly render into the DOM, why would it ever make sense to go back to rendering on the server?

Server-side rendering of views when using a JavaScript framework allows for the initial view of the web app to be displayed to the user before the JavaScript has been downloaded and initialized in the browser. This allows for a fast app startup when it is believed the download and start of the JavaScript environment and the initial view rendering might be slow. Running on a weak network connection or browser environments (e.g. wimpy phones) this could be beneficial.