CS 142 Final Examination (Solutions)

Spring 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)

_____

(Print your name, legibly!)

_____@stanford.edu

(SUID - Stanford email account for grading database key)

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

Problem #1 (12 points)

(a) Assume you have joined a new international standards body for the Internet that has decided to take on the problem of misleading names. The standards body has taken up a complaint against the names *HyperText Transport Protocol* (HTTP) and the DOM's *XMLHttpRequest.* Using our class's Photo App as an example, describe what is misleading about the name and why it is misleading for our Photo App:

A. The word "HyperText" in HyperText Transport Protocol.

HyperText refers historically to text containing hyperlinks. However, as we saw in our photo app, we can transmit many other data types. For example, we used JSON encoding to transfer model data objects, script tags fetched JavaScript programs, link tags fetched CSS stylesheets, img tags fetched jpeg/png images, etc.

B. The acronym XML in XMLHttpRequest.

XMLHttpRequests actually more often use JSON for their request bodies, as we saw in our implementation of fetchModel. XML is just the historic name used by the first developers.

Problem #2 (12 points)

(a) If you display a web page in the Chrome browser that is entirely fetched using the HTTPS protocol, Chrome will display  next to the URL. If you change the web page to include a small image of something public (e.g. a logo) and use HTTP to fetch the image Chome will display 

next to the URL as if you didn't use HTTPS for anything. Explain why the Chrome developers could justify slapping "Not Secure" on something seemingly benign as an HTTP fetch of a small image. Describe the attack they are worried about.

This is referred to as a "Mixed Context" that can lead to many problems. Since we're just fetching an image (i.e. `<img src="http://...`) most of the attacks like injecting scripts or striping HTTPS are not relavent. We do expose the session cookies to an "man-in-the-middle" attacker so they can do a session hijacking attack. Since man-in-the-middle is the kind of attacks HTTPS is trying to address, it makes sense the Chrome developers would flag this as insecure.

(b) Your friend has started a company with a photo sharing web app. In order to associate each request to the server with a logged in user, she includes the user ID in the cookie. On each incoming request, the server checks that field in the cookie to figure out which user the request is coming from. (1) What is problematic about this? (2) What is the correct way to achieve the same goal? (3) Would switching to use HTTPS solve the problem?

(1) This is problematic if the user IDs are known. An attacker anywhere on the Internet can easily generate such as cookie and impersonate any user. This would be considered an example of a predictable session ID problem.
(2) To prevent such an easy way of forging a session, we need to generate the identity information on the web server and use a message authentication code (MAC) to make it hard for an attacker to generate.
(3) Since this is a problem with forging cookies, encrypting the transport between the browser and the web server doesn't do anything to prevent it. It would make it hard for a man-in-the-middle attacker to steal a session cookie but that doesn't matter if they can just generate one.

Problem #3 (10 points)

In all web applications, it is important to keep the state consistent between the front-end and back-end. We attempted to do so in our photo app by utilizing **state** on the front-end and **session state** on the back-end. Unfortunately, there were still several ways the two states could become out of sync.

a. With the base functionality of our photo app (without considering any extra-credit features), describe how a user could end up being logged in on the back-end but not on the front-end.

1. Log-in on the front-end, which stores some sort of loggedIn flag in React's state and the user information in the backend session.
2. Refresh the application. This clears the front-end state but does not log out the user on the back-end.

Alternatively, as in part (b), you could just call the login backend endpoint directly without utilizing the frontend interface.

b. In our implementation, is it possible for a user to be logged in on the front-end but not on the back-end? Explain why or why not.

Yes. A malicious user can log in via the frontend, which yields a consistent state across the front-end and back-end. Then, the user can bypass the front-end by issuing a request to logout directly via the backend API, which will log out the user on the backend but keep the logged in state on the frontend.

Problem #4 (10 points)

While early web applications were delivered to the browser like an old style web page written in HTML, we used ReactJS to directly draw our Photo App using the DOM interface of the browser. It is possible to run ReactJS in the web server and go back to sending HTML to the browser.

A. Explain the main advantage of this sending HTML approach.

The initial page load will be faster. The browser can display the page before the Javascript has been downloaded and initialized in the browser. This allows for a fast app startup, which should help in areas with weak connectivity or smaller devices such as phones or tablets. Accepted explanations that mentioned something about the JavaScript potentially loading later/slowly, less data sent on the initial request, benefits in low connectivity areas.

B. Explain what the disadvantages would be for our Photo App to do this sending of HTML rather than using the DOM interface.

Every page now requires an HTTP request (which can be slow). Navigating through the app can become laggy and give a poor user experience. Additionally, this adds more computational load to our servers. Issues of load balancing to handle extra requests. Partial credit given for answers that mentioned UI latency or extra work being done server-side to create the HTML files.

Problem #5 (8 points)

A student asked the following question on Piazza:

> In the projects, how is the photo-share.html file loaded? In the projects 5-8, how is the photo-share.html loaded from the server to the browser. It seems that all the functions in the webServer.js (app.get() or app.post()) are only loading the model data. Does loading photo-share.html even use webServer.js? Thank you.

Write a good answer to the question. In the spirit of the course theme of how web apps work, your answer should include a description of the mechanism that gets the photo-share.html file into the browser.

Express middleware allows us to export static files (`app.use(express.static(__dirname))`). Partial credit given if student mentioned that Express played a role, less partial credit given for mentioning the server was accessing a file from the project directory.

Problem #6 (8 points)

Web app developers use the term "Server Push" to describe having a Web Server with the ability to update information in Browsers by "pushing" it from the Web Server to Browser. Describe something fundamental about the HTTP protocol that makes it less than ideal for implementing "Server Push".

HTTP is a request/response based protocol. The client can only receive data from the server by requesting it. If we were to try to do server push built off HTTP, we'd have to break this pattern somehow. We could try to leave TCP connections open, but that creates way too much traffic/overhead. The server cannot initiate HTTP requests since there is no way to verify the identify of the browser before sending the request.

Problem #7 (8 points)

In class we learned that Memcache is a storage system that is used to store session state and cache results of database queries. It is used because it is significantly faster than accessing data stored in database. Explain why we don't use Memcache for all our web app's storage needs.

Memcache tries to be very fast (low latency) at the expense of the traditional reliability of storage in database system. One copy (no replication) is kept and that copy is kept in main memory which is lost if the power fails. We would want to storage our web app data in some more reliable than memcache.

Problem #8 (8 points)

Explain why it is easier to apply a scale-out approach to web servers than database servers.

The scale-out approach for web servers is much easier to manage because we can build stateless servers such any server can handle any particular request, and we can balance the load appropriately.

This is a more complicated for database servers, because we need to figure out how to partition the data across multiple instances (data sharding), and hope our partition scheme doesn't have imbalance, especially as our application scales.

Problem #9 (8 points)

Describe the key property of cloud computing services (e.g. Amazon Web Services) that make it a game changer for web applications that face uncertain but possibly explosive growth when compared to owning your own servers.

Cloud computing services allow developers to pay for only their necessary resource uses. The possibly explosive growth means you don't know how many resources you will need, but cloud services can easily handle this case by scaling up the amount of resources.

Problem #10 (8 points)

Assume a new version of the Chrome came out that contained a bug that broke enforcement of the same origin policy. Accesses that were previously prevented by the same origin policy are now permitted. Explain what an attacker running in a different browser tab could do to our Photo App. Describe the attack.

There were a few variations of correct answers for this question. One example was the following. An attacker could hijack our session and use it then run malicious javascript. The javascript would now not be flagged as coming from a different origin. This means they could make requests as though they were us. They could add pictures, comment on other people's photos and impersonate us. Some students lost points for not mentioning anything specifically related to our photo app.

Problem #11 (10 points)

(a) Describe a Denial of Service(DoS) attack that would work on your Project #7.
Most students got full credit on this question by mentioning some way an adversary could deplete a resource, such as by uploading too many photos.

(b) Describe how you could change your Project #7 to mitigate the attack.
Most people again got full credit for this question by suggesting something that would protect against the attack they outlined in a. For instance, limiting the number of photos a user can upload.

Problem #12 (12 points)

Consider the following Node.js program:

```
var async = require('async');

var arr = ["A", "B"];
var printOrder = [];

async.each(arr, function (elem1, doneCallback) {
  async.each(arr, function (elem2, doneCallback2) {
     console.log(elem1 + elem2);
     printOrder.push(elem1 + elem2);
     doneCallback2();
  }, function () {
     console.log('innerDone', elem1);
     doneCallback();
   });
}, function() {
   console.log('outerDone');
   console.log("PO1", printOrder);
})
console.log("PO2", printOrder);
```

(a) Execute the code by hand and show a possible output from the program. (Hint: `(elem1 + elem2)` concatenates the two characters)

AA
AB
innerDone A
BA
BB
innerDone B
outerDone
PO1 ['AA', 'AB', 'BA', 'BB']
PO2 ['AA', 'AB', 'BA', 'BB']

(b) Is it possible that the console.log lines that do not contain the string "Done" can occur in a different order on different runs? Justify your answer

No, async.each will actually execute synchronously since there are no events to wait for/no blocking calls (just console.logs which are run immediately). Partial credit was given to answers that answered YES and explained that the array elements could be handled in random order (which would be true if our console.log statements were instead events that async needed to wait on). No credit given to answers that relied on time or some callbacks “finishing first”.

Problem #13 (10 points)

(a) Explain how Cross-Site Request Forgery (CSRF) attack can allow a website running in a different browser tab to perform operations on a web app like it is logged into the web app even though the attacker doesn't know the user name or password of the user in the web app.

The browser sends all cookies associated with a website when making a request, even if the request is initiated by a third party website. CSRF leverages this. By forcing you to make requests, a malicious site can send your session cookies with the requests.

(b) Explain why the browser can not just turn off the mechanism used by CSRF. Give an example of something useful that breaks with the mechanism turned off.

Cookies allow for many fundamental features of web-apps, such as remembering sessions/login info. These are used for features like logging into other sites (think “Login with Facebook/Google”) or for maintaining sessions across tabs/windows/deep linking.

Problem #14 (12 points)

(a) Database systems that support primary and secondary indexes will allow a user to have multiple secondary indexes but only one primary index on a table. Explain the reason for this.

In a database, a primary key means that the database uses the key to organize the storage of the relation/object. Although MongoDB uses it uniquely generated object as the primary key, it is the storage organization not the uniqueness that defines primary keys. Since you can only organized storage around one key, one can only have one primary key.

(b) When an index is added to a collection in MongoDB, it could result in a performance increase or a performance decrease, describe how you could look at a system running and predict if adding an index would help or hurt performance. Give examples that show both cases of helping and hurting.

Indexes can help when you have a database that is spending much time scanning through data looking from something. It is in this case an index can greatly help performance by allowing the database to directly fetch the data rather than having to scan for it. Indexes hurt performance when they require extract computing resources to update and/or store. In terms of predicting what an index would do, we can first look for much scanning through (without using) relations/objects and not a lot of mutant would indicate indexes could be a win. A database that doesn't do much scanning is not going to get much of a win from indexes and can get hurt if there is much changes going on.

Problem #15 (10 points)

You have been appointed a CA for CS142 and are approached by a student during office hours for a bug in Project #6. Instead of using `axios` to talk to the web server, they used another JavaScript library called `getIt` that had similar interface to `axios.get`. The student describe the following.

- The student tried doing:

  ```
  let userList = getIt("/usr/list");
  ```

  and discovered userList was an object but didn't contain the model data properties from /user/list.
- The student then discovered that if they did:

  ```
  let userList = getIt("/usr/list");
  setTimeout(function(){ processModel(userList.fulfillmentValue);}, 10);
  ```

  the system worked. Although the userList never got set to the model data some property named `fullfillmentValue` ended up having the model data even though it wasn't set immediately after the call to `getIt`.

(a) Write an explanation to the student what is going on with their code.
getIt() is returning a promise. The user isn't handling the promise correctly, but is getting around this by waiting 10 seconds for it to resolve, and accessing the value at resolution as `userList.fulfillmentValue`

(b) Show the code you would suggest to properly use `getIt`. Your solution should pass the response data to the routine `processModel` as was done in the code above.

```
let userList = getIt("/usr/list");

userList.then(function(result) {
      processModel(result);
});
```

Problem #16 (10 points)

(a) The relational data model stores data in tables consisting of columns and rows. Assume you have been given the task of moving data in an object database like MongoDB. Describe how each of the following concepts from a relational database would map to something in the object model:

(i) A Table

collection of objects of a specific collection

(ii) A Row in a Table

Instance of an object

(iii) A Column in a Table

A property in the object schema

(iv) A value in a particular Row and Column in a Table

The actual value of the property for the instance of an object

(b) It is possible to map a relational data model into an object model. Explain why the reverse, mapping an object model (i.e. JSON-like MongoDB) to a relational database, doesn't work.

A majority of points were given if students mentioned that json's ability to allow for more types such as nested jsons could cause issues. However, the main issue we were looking for was that every object in a collection could have a different set of properties. This would make it very difficult to have any sort of mapping to a table with fixed columns.

Problem #17 (12 points)

(a) When doing routing of URLs in both React Router and in ExpressJS in our photo app, we ended up having routes that contained a colon character (e.g. `"/foo/:bar"`) yet we never included a colon in the hierarchical part of a URL we used.

(i) Explain the purpose of this colon character?

The colon character indicates to React Router that 'bar' is a route parameter, ie that whatever the user places in that position in the URL will be a route parameter named ‘bar’. React Router will use attach this parameter to our component's 'props' object so we can display the data associated with this parameter in the component. (similar interpretation for ExpressJS)

(ii) Describe what would happen if we just deleted the colon from the routes.

If we delete the colon, React Router will only route /foo/bar to our component, instead of /foo/anything to our component. Since our component also probably has some dependency on the ‘bar’ component on the ‘props’ object, our app will likely break. (similar interpretation for ExpressJS)

(b) Our use of ExpressJS in the Photo App took advantage of much functionality offer as Express Middleware. If we didn't use the Middleware mechanism, could we have implemented the same functionality in our webServer.js?
If your answer is no, describe the functionality we couldn't achieve without Middleware.
If your answer is yes, explain the disadvantage of not using Middleware.

Yes, we could have implemented the same functionality. The disadvantage is we would have to copy paste the middleware functionality into all of our API endpoints, which is redundant.
Note: Many people answered ‘no’ because we wouldn’t have access to third-party middleware like express-session. The question isn’t asking about third-party software (and middleware doesn’t have to be third-party), but about whether the *mechanism* of middleware itself is indispensable to achieve some *functionality* in our app, which it is not. We only accepted the ‘no’ answer if students made some mention of the need to add code to every endpoint, any answer discussing the ‘difficulty of coding third-party software ourselves’ received only partial credit since that is not what the question is asking.

Problem #18 (12 points)

(a) In your photo app assignment, you were encouraged to validate user input in both your back-end server and your front-end React application. However, some errors can only be handled on the back-end; give an example of such an error and explain why it cannot be validated on the front-end.

An example from the photo app would be verifying that a username is valid during the registration process before allowing a successful registration. This cannot be validated on the front-end because it requires making a database query to check if the username is taken, and the front-end only has access to the database through the back-end.

(b) Our photo app assignment accessed the MongoDB database using the Mongoose object definition language rather than talking directly to MongoDB. Describe the advantages this setup of going through Mongoose gave us when compared to accessing MongoDB directly.

Mongoose masks the lower level interface to MongoDB with something that is friendlier. It provides some level of abstraction in the form of the Schema and Model objects. It also performs built-in object validation based on our schemas for us (which requires writing substantially more code in MongoDB). Note: answers with a sentence or two copied from the slides without any further interpretation/explanation received partial credit.