1 Homework 4 Solution

Due Date: CS 580W: Dec 5; CS 480W: Dec 7
No late submissions

To be turned in on paper in class.

Important Reminder: As per the course *Academic Honesty Statement*, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

Please remember to justify all answers.

You are encouraged to use the web or the library but are required to cite any external sources used in your answers.

1. The course discussed two approaches for maintaining state across multiple HTTP requests:

   (a) Include the state as part of the URL for each request, usually as a query parameter.

   (b) Use cookies.

   Usually the state maintained in a client is some kind of id which identifies state stored on the server.

   Discuss the tradeoffs between these two approaches. *10-points*

   - If cookies are used, they need to be allowed by the end-user. This was not true in the early days of the web as cookies had a bad reputation as they could be used for tracking users. Since the web has become much more commercial, users seem to have reconciled themselves to being tracked and this is much less of an issue today.
   - If URLs are used for tracking state, each URL must be modified to include the state. This can be painful to implement manually at multiple places in the server codebase and usually necessitates the use of a framework to ensure this. OTOH, once a cookie has been set by the server, it is the responsibility of the browser to ensure that the cookie is returned for subsequent requests and it is a simple matter for the server code to set up the state from the cookie when each request is received.
   - If URLs are used for tracking state, purely static pages become impossible as any link to another page needs to include the state information.
   - If URL's are used for tracking state, then leaving the web site and going to another web site will loose the state information when the web site is reentered.

- Bookmarks will retain state when URLs are used for tracking state. This need not be the case when cookies are used, depending on whether the lifetime of the cookie extends past the time of revisiting the bookmark. Specifically, using a session cookie and then closing the browser will loose state.
- One advantage of URLs related to the previous point: if state is tracked via URLs, then it is possible to send a URL via email and accessing that email link will also get access to the state; this will never work when cookies are used if the emailed link is accessed in a different browser.

Based on the above tradeoffs, it seems that cookies are the best choice for modern web sites, especially if they are persistent cookies.

2. John is a non-technical proprietor for a small business. He hires a consultant to develop a web site where customers can place orders. The consultant successfully delivers a slick looking web site based on express.js and sets up the express.js server using a web hosting company. The web site works well and orders are pouring in.

    (a) Unfortunately, after a few days of successful operation John gets a call from the web hosting company informing him that the memory footprint of the express.js server has increased tremendously.

    (b) John has no idea what that means and places a call to the consultant requesting assistance.

    (c) Before the consultant can get back to him, John receives another call from the web hosting company informing him that his server crashed and they have restarted it.

    (d) The web site is working well with the server having a small memory footprint. However, John starts receiving complaints from customers saying that they have lost the contents of their shopping carts.

    Give a possible reason for this sequence of events. *5-points*

    One possible reason is that the consultant set up the express.js server to store shopping carts using in-memory sessions. This would result in all of the symptoms listed: the size of the server's memory footprint increases until the server crashes and the shopping carts are lost after the restart.

3. One disadvantage of the PRG Post-Redirect-Get pattern is difficulty sending information from a successful `POST` to the `GET`. Discuss alternative solutions to this problem? *10-points*

    One way to send information from the POST to the GET is to simply store it on the server:

    - It can be stored in the server session. This is usually not a good idea for important information as the server may crash between the `POST` and the `GET` but is definitely an option for unimportant information where loosing the information is acceptable,

- Store the information in persistent storage.

In situations where storing the information on the server is not an option, we could put the information into the `GET` URL as a query parameter. Examples:

- Consider adding an item to a shopping cart. We may want to display the shopping cart with a message saying that the item was successfully added. We do not want to clutter up our server state with the success message. We could simply add the success message to the shopping cart URL as a query parameter.
- A user types sensitive information like a credit card number on a billing page and we want to have the credit card number available on a order review page so that it is available if the user decides to place an order. If we do not store credit card numbers persistently on the server, our web site can avoid embarassing press coverage involving credit cards in case it ever gets cracked. So we would put the credit card number as a query parameter in the URL, but it **must be encrypted**. When the server processes the `GET` request for the order review page, it should add the encrypted credit card number as a hidden widget to the form used to place the order.

In summary, usually pass information from the `POST` to the `GET` via the server. When that is not possible, do it using query parameters, but be sure to encrypt any sensitive information passed via query parameters.

4. You are hired to help an ecommerce web site evaluate two alternatives for allowing users to checkout their shopping carts:

   (a) A checkout flow:

      i. A shipping page where the user provides shipping information for the order.

      ii. A billing page where the user provides a credit card number to pay for the order.

      iii. A page where the user can review all the order information before deciding whether to place the order.

      iv. An invoice page for the order after it has been successfully placed.

   (b) A single checkout page on which the user provides all necessary information like shipping and billing. When an order is placed, the checkout page is replaced by an invoice when an order is sucessfully placed.

   The web site may need to support the following:

- Allow the user to specify promotional offers like coupons during the checkout. These offers may affect the total cost of the order.
- Allow a discounted (possibly free) shipping based on the total cost of the order.

Provide a technical evaluation for the two alternatives. Your evaluation should include the following:

- For the checkout flow, suggest possible URLs for the different pages as well as the HTTP methods used.
- The information flow between the user and the application.
- Requests sent between the browser and server.

You may assume that the server stores all details of the shopping cart and current state of the checkout flow with cookies being used to link browser requests to the server state. However, for security reasons, credit card numbers are not permitted to ever be stored on the server. *15-points*

We assume that the shopping cart is identified using some id *cart-id*. We assume that the shopping cart not only contains the order items, but will also be updated with the shipping and billing information as the customer advances through the checkout flow.

For the checkout flow:

- If we allow both promotions and discounted shipping, then it is possible that entering in discount information like coupons can affect the shipping cost. There are two alternatives:
  (a) We allow the customer to enter in the discount information on a separate page before we display the shipping page. This adds another page to the checkout flow.
  (b) We make the shipping page dynamic using AJAX so that the shipping cost updates dynamically as the user enters in the discount information.
- When a credit card number is entered into the billing page we would pass it on to the order review page as an encrypted query parameter (see the answer to the previous question).
- We would use the PRG pattern to advance through the checkout flow using URLs like the following:
  (a) If we have a discount page, then `GET` to /*cart-id*/`discounts` would display the discount page. Submitting a form on that page would `POST` to the same URL with success updating the server state and doing a redirect to /*cart-id*/`shipping`

(b) A `GET` to /*cart-id*/`shipping` would display the shipping page. Submitting a form on that page would `POST` to the same URL with success updating the server state and doing a redirect to /*cart-id*/`billing`

(c) A `GET` to /*cart-id*/`billing` would display the billing page. Submitting a form on that page would `POST` to the same URL with success updating the server state and doing a redirect to /*cart-id*/`order-review?cc=`*encrypted-cc*.

(d) A `GET` to /*cart-id*/`order-review` would ensure that an encrypted credit card number was available in the `cc` query parameter and display the order-review page with the encrypted `cc` put into the page as a hidden input to a form. Submitting that form would `POST` to the same URL with success updating the server state and doing a redirect to /*order-id*/`invoice`, where *order-id* is some generated ID for the place order (which may be different from the *cart-id*).

A single checkout page could use AJAX to update the server as the user fills in different portions of the page:

- It would obviate any clumsiness in the checkout flow caused by the inherent sequencing between promotion information and computation of shipping costs.
- It would allow immediate feedback to customers on errors without requiring customers to submit forms.
- It would allow the *Place Order* button to be made inactive until all necessary information has been entered in and validated.
- Depending on the commercial agreements entered into by the ecommerce company with its payment processors, it may be possible to preauthorize the credit card with the payment processor as soon as the customer has entered in the credit card number and show any credit card errors even before the customer places the order.
- One disadvantage of a single checkout page over the multi-page checkout flow is that the page can get quite heavy, both visually and in size. Some care in designing the page and its state can reduce this problem.

The checkout flow seems to be a throwback to the days of the pre-AJAX web. The single checkout page seems to be a better choice for a modern web site.

5. A code base uses `XMLHttpRequest` to provide an `insertContent()` function which asynchronously inserts content from a specified `url` into the DOM element specified by an `id`:

```
    function insertContent(id, url) {
```

```
    const req = new XMLHttpRequest();
    req.onreadystatechange = () => {
      if (this.readyState == 4 &&
          this.status == 200) {
        //insert into page
        document.getElementById(id).
          innerHTML = req.responseText;
      }
    };
    req.open("GET", url);
    req.send();
  }
```

This function is used to insert messages into a page:

```
  const messages = [
    { id: 'intro', url: 'http://ex.com/intro.txt', },
    { id: 'descr', url: 'http://ex.com/desc.txt', },
    ...
  ];

  messages.forEach(m => insertContent(m.id, m.url));
```

Since the messages are received asynchronously, they will be inserted into the DOM in an arbitrary order. Describe how you would modify the above code so as to insert the messages into the DOM in the same order as they are specified in the `messages[]` array. *10-points*

The key idea is to buffer each received message until all preceeding messages have been written into the DOM. When a message is received, we will write it and all subsequent messages which have already been received into the DOM.

Example pseudo-code which provides the details of this idea:

```
  const messages = [
    { id: 'intro', url: 'http://ex.com/intro.txt', },
    { id: 'descr', url: 'http://ex.com/desc.txt', },
    ...
  ];

 const messagesBuffer = new Array(messages.length).fill(null);
  let waitingIndex = 0; //waiting for message at this index

  function receiveMessage(index, id, message) {
    messagesBuffer[index] = { id, message };
    if (index === waitingIndex) {
      //insert all following messages into DOM
      let i;
      for (i = index;
           i < messagesBuffer.length && messagesBuffer[i];
           i++) {
        const msg = messagesBuffer[i];
        const [msgId, msgText] = [msg.id, msg.message];
       document.getElementById(msgId).innerHtml = msgText;
      };
      waitingIndex = i;
    }
  }

  function fetchMessage(id, url, index) {
    const req = new XMLHttpRequest();
    req.onreadystatechange = () => {
      if (this.readyState == 4 &&
          this.status == 200) {
          receiveMessage(index, id, req.responseText);
      }
    };
    req.open("GET", url);
    req.send();
  }

 messages.forEach((m, i) => fetchMessage(m.id, m.url, i));
```

It is not necessary that your answer show sample code like the above; it is sufficient that you explain the basic idea of buffering each message outside the DOM until all preceeding messages have been received.

6. Jane, who is a programmer working at a university lab, is learning about web services and builds a simple web service running on her PC at `http://jane.lab.university.edu/fortune` which returns a random fortune. Jane embeds this service into her personal home page on her PC by using the `insertContent()` function from the previous question with the following code snippet:

```
    insertContent('fortune', '
                  'http://jane.lab.university.edu/fortune')
```

The lab director is impressed by this web service and instructs the lab's intranet programmer to embed the fortune into the lab's intranet home page at `https://lab.university.edu` by:

(a) Adding a suitable `<div id="fortune"></div>` to its HTML.

(b) Modify its JavaScript to include the `insertContent()` function as well Jane's code snippet which calls `insertContent()`.

This does not work. Give possible reasons why. *10-points*

Some possible reasons:

- Since `http://jane.lab.university.edu/fortune` is at a different origin from `https://lab.university.edu`, the same-origin policy will not allow JavaScript from the lab intranet domain to make a call to Jane's domain. A fix for that would be to have Jane set up CORS exceptions to whitelist the lab intranet domain.

- It is possible that the lab is set up with restrictive firewalls. Since most employees use their workstations purely as web clients, the lab's firewall policies probably do not allow use of a workstation as a server by the intranet server. So it may be impossible for the intranet homepage to access Jane's web service running on her PC without some modification of the lab's firewall rules.

7. Given a reactjs component `Component`:

```
class Component extends React.Component {
  constructor(props) {
    super(props);
    this.id = props.id;
    //no redefinition of this.handleChange
  }

  handleChange() {
    //references this.id
    ...
  }
}
```

(a) What is wrong with the following `render()` method for the above `Component`?

```
render() {
  return (
 <input type="text" onChange={this.handleChange()}/>
  );
}
```

(b) What is wrong with the following `render()` method for the above `Component`?

```
render() {
  return (
            <input type="text"
              onChange={function() {this.handleChange()}}/>
          );
        }
```

(c) Why will the following closely related `render()` method for the above `Component` work?

```
        render() {
          return (
            <input type="text"
                onChange={() => this.handleChange()}/>
          );
        }
```

15-points

(a) The problem is that the `onChange` handler is being run when the `onChange` handler is being registered; not when the input is changed.

(b) The problem is that `this` is setup incorrectly; it will refer to the global `window` object or `undefined` if in `strict` mode. The `handleChange()` method requires that `this` reference the component. Hence the handler will not work.

(c) Unlike a function defined using `function`, a fat-arrow function will inherit `this` from its surrounding context. So the onChange handler will inherit it's this from the `render()` function; since `render()` is a method of the component, this will correctly refer to the component within the `handleChange()` method.

8. You need to call two independent remote web services fronted using two `async` functions `async function service1()`, and `async function service2()`. Since these are remote web services, it is likely that the calls take appreciable time (milliseconds) which is not predicable beforehand. Your code needs to block until both web service calls have completed. It is trivial to use `await` to write code which calls both web services sequentially but that has the disadvantage of taking time which is the sum of the times taken by each service. How would you set things up so as to call both services but to block only for a time corresponding to the slowest service? *10-points*

   Make use of `Promise.all()`. Calling `service1()` and `service2()` should return a `Promise`, so something like

   ```
   await Promise.all([service1(), service2()]);
   ```

should block until both web services have completed. `Promise.all()` will run the web services in parallel, so the block will only be for the time corresponding to the slowest web servicce, rather than the sum of the times taken by both services.

9. You have a working promise chain:

```
promiseGen().
  then(function(v1) { ... return v2; }).
  then(function(v2) { ... return v3; }).
  then(function(v3) { ... return v4; });
```

Due to changes in your code, you need to change the `function()` in the third `then()` so that its body has access to the value `v1` returned to the first `then()` by the promise generator `promiseGen()`. However, `v1` is out-of-scope for the third `then()`. How can you make this happen? *15-points*

One solution is to use a contextual (global) variable:

```
let gV1 = null;
promiseGen().
  then(function(v1) { gV1 = v1; ... return v2; }).
  then(function(v2) { ... return v3; }).
  then(function(v3) { ... gV1 ... return v4; });
```

This would work because the third promise function would not run until after the first two have completed. It has the usual problems with the use of contextual variables.

Another solution is to pass the value down the promise chain:

```
promiseGen().
  then(function(v1) { ... return [v1, v2]; }).
  then(function([v1, v2]) { ... return [v1, v3]; }).
  then(function([v1, v3]) { ... v1 ... return v4; });
```

Note the use of destructuring in the function parameter for the last two promise functions.