

THE UNIVERSITY OF HONG KONG

FACULTY OF ENGINEERING  
DEPARTMENT OF COMPUTER SCIENCE

COMP3322 Modern Technologies on World Wide Web

Date: May 18, 2018

Time: 2:30 pm – 5:30 pm

University Number: \_\_\_\_\_

*Only approved calculators as announced by the Examinations Secretary can be used in this examination. It is the candidates' responsibility to ensure that their calculator operates satisfactorily, and candidates must record the name and type of the calculator used on the front page of the examination script.*

*Candidates are allowed to bring into the examination hardcopies of all lecture slides, on which handwritten notes are allowed.*

**Answer ALL questions in the space provided. Use the reverse side of a page if you need extra space.**

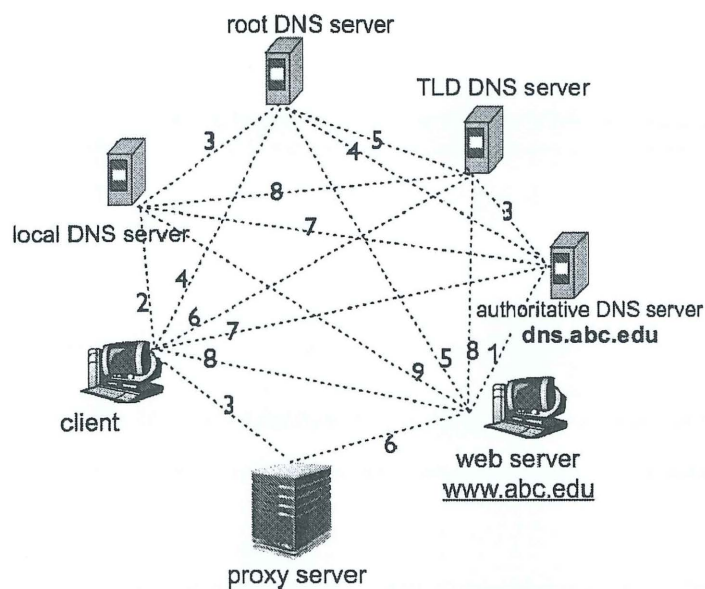
**Total mark is 100.**

| Question     | Score |
|--------------|-------|
| 1 (14%)      |       |
| 2 (16%)      |       |
| 3 (24%)      |       |
| 4 (34%)      |       |
| 5 (12%)      |       |
| Total (100%) |       |

**Question I (14%) Answer the following questions.**

(1) (3%) Suppose you are in Hong Kong and you are sending a WhatsApp message on your smart phone to your friend, who is in Canada and receives your message on her smart phone. Explain what devices are typically involved in the message path from you to your friend, and what layers of protocols in the Internet protocol stack are running on each device.

(2) (6%) Suppose a client clicks a link within her web browser to obtain a web page located at <http://www.abc.edu/index.html>. The IP addresses of the local DNS server and the proxy server are configured within the client. The proxy server handles all HTTP requests/responses from/to the client. Suppose it is the very first time that the client and any client using this proxy server access the website; initially only the authoritative DNS server knows the IP address of the web server hosting the website; initially only the authoritative DNS server knows the IP address of the web server hosting the website; initially only the authoritative DNS server knows the IP address of the web server hosting the website. After the client or a DNS server learns any name-address mapping, it caches the mapping for one hour. The proxy server caches a web resource for one hour after receiving the resource. In the figure below, the number on a dotted line denotes the time taken for one-way message passing between the two hosts/servers connected by the line. For example, "2" on the line connecting the client and the local DNS server indicates that it takes time 2 for the client to send a request to the local DNS server and it also takes time 2 for the local DNS server to send a response to the client. Suppose non-recursive DNS query is used; the HTTP protocol in use is HTTP 1.1; the client finds out the IP address of the web server before sending out respective HTTP requests. We only consider message passing delays but not any other delays in the network.



(a) Describe the steps incurred from when the client clicks on the link until the client receives the web page `index.html`, and compute in total how much time elapses (i.e., total time to complete all those steps).

(b) Suppose the client subsequently clicks a link in the received `index.html` (within 1 minute after receiving `index.html`), which links to <http://www.abc.edu/images/logo.jpg>. Describe the steps incurred from when the client clicks this link until the client receives the image `logo.jpg`, and compute in total how much time elapses for completing these steps.

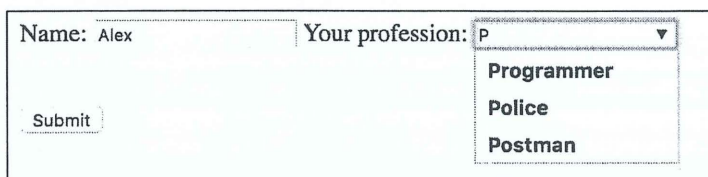
(3) (3%) You are given the following HTTP response. Explain what the "Cache-Control" header values mean, and what the "ETag" header is used for.

```
HTTP/1.1 200 OK
Date: Fri, 27 Oct 2017 14:19:41 GMT
Server: Apache/2.4.27 (Unix)
Cache-Control: max-age=3600, must-revalidate
Expires: Fri, 27 Oct 2017 15:19:41 GMT
Last-Modified: Mon, 23 Oct 2017 02:28:12 GMT
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html

Content...
```

(4) (2%) Give under what circumstances SOAP web service is preferred and under what circumstances REST web service is preferred, when choosing between them for implementing a web service.

**Question 2 (16%)** Suppose you are implementing a simple webpage index.html as shown in the figure below. There is a form containing an input textbox for the client to enter a name, an input field for entering the profession or selecting the profession from a drop-down list of pre-defined options (which show up when the entered letters match starting letters of the pre-defined options), and a submit button. Both input fields should be empty the very first time when a client browser has loaded the page. After entering values into the fields and clicking submit, the client browser should remember the entered values before sending them to the server side. The next time when this browser loads the web page again (e.g., after shutting down and then relaunching the browser), the entered values saved from the previous submission of the form should be automatically loaded into the two input fields.



The image shows a web form with two input fields and a submit button. The first field is labeled 'Name:' and contains the text 'Alex'. The second field is labeled 'Your profession:' and is a dropdown menu with the letter 'P' in the selection box. The dropdown menu is open, showing three options: 'Programmer', 'Police', and 'Postman'. Below the fields is a 'Submit' button.

(1) (2%) To implement the stated functionalities, would you use localStorage or sessionStorage to store user entered data in the client browser? Explain your choice.

- (2) (14%) Based on your choice in (1), implement the web page according to the sketch given below.
- You can add code into a given HTML tag if you deem necessary.
  - Ignore what processForm.php might be achieving, but complete index.html only.
  - You should decide the form elements to use, to achieve the page view as shown in the above figure. When implementing the dropdown list, you can just include three options, namely "Programmer", "Police" and "Postman".
  - Use basic JavaScript to write the code in <script></script>.
  - The JavaScript function for storing the form data in the browser, to be invoked when the form is submitted, is storeFormData() and the JavaScript function to be invoked when the web page is loaded is loadFormData().

Index.html

```
<!DOCTYPE html>
<html>
<body>
<form action="processForm.php" method="GET">
```



```
<input type="submit">
</form>
<script>
  function storeFormData(){
    if(typeof(Storage)!="undefined")
    {

    }
    else {
      alert("Sorry, your browser does not support web storage...");
    }
  }

  function loadFormData(){
    if(typeof(Storage)!="undefined")
    {

    }
    else {
      alert("Sorry, your browser does not support web storage...");
    }
  }
</script>
</body>
</html>
```

**Question 3 (24%)** You are going to implement a simple web app using index.html, script.js and myservice.php. The app is hosted on <http://www.mysite.com>, and can be accessed by <http://www.mysite.com/index.html>. Index.html has been provided as follows.

#### Index.html

```
<html>
  <head>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <fieldset>
      <legend>Contact List</legend>
      <table id="contactlist">
        <tr><th>Name</th><th>Address</th><th>Phone Number</th><tr>
      </table>
    </fieldset>
    <br><br>

    <form id="addcontact">
      <fieldset>
        <legend>Add New Contact</legend>
        Name: <input type="text"> <br>
        Address: <input type="text"> <br>
        Phone: <input type="text"> <br>
        <input type="submit">
      </fieldset>
    </form>
  </body>
</html>
```

Suppose a database "contactDB" has been created on a MySQL database server [sqlldb.services.com](http://sqlldb.services.com), which can be accessed using login ID "user" and password "supersecure". A table has been created in the database and a number of records have been inserted into the table using the follows SQL statements:

```
CREATE TABLE contacts (
  ID int NOT NULL AUTO_INCREMENT,
  name varchar(20) NOT NULL,
  address varchar(20) NOT NULL,
  phone varchar(20) NOT NULL,
  PRIMARY KEY (ID)
)

insert into contacts values(1, 'Max', '123 Major Street', '55555555');
insert into contacts values(2, 'Minnie', '567 Minor Street', '66666666');
...
```

(1) (12%) Implement script.js to achieve the following functionalities: when index.html has been loaded, an HTTP GET AJAX request is sent for <http://www.mysite.com/myservice.php?getallcontacts=true>, to retrieve all the existing contact records from the database table, and display them in the table of id "contactlist". An illustration of the page after loading is given in Fig. 3-1. The client can then enter new contact information as illustrated in Fig. 3-2. Then after clicking the submit button, an HTTP POST AJAX request is sent for <http://www.mysite.com/myservice.php>, carrying the name, address and phone number entered by the client in the three input textboxes. Upon receiving a success response from the server side, the client should add the newly entered contact information as the last row in the contactlist table and clear all three input textboxes, as illustrated in Fig. 3-3.

Fig. 3-1

| Contact List |                           |
|--------------|---------------------------|
| Max          | 123 Major Street 55555555 |
| Minnie       | 567 Minor Street 66666666 |

| Add New Contact                       |                      |
|---------------------------------------|----------------------|
| Name:                                 | <input type="text"/> |
| Address:                              | <input type="text"/> |
| Phone:                                | <input type="text"/> |
| <input type="button" value="Submit"/> |                      |

Fig. 3-2

| Contact List |                           |
|--------------|---------------------------|
| Max          | 123 Major Street 55555555 |
| Minnie       | 567 Minor Street 66666666 |

| Add New Contact                       |  |
|---------------------------------------|--|
| Name:                                 | <input type="text" value="Callie"/>                                    |
| Address:                              | <input type="text" value="789 Median Street"/>                         |
| Phone:                                | <input type="text" value="77777777"/> <input type="button" value="v"/> |
| <input type="button" value="Submit"/> |  |

Fig. 3-3

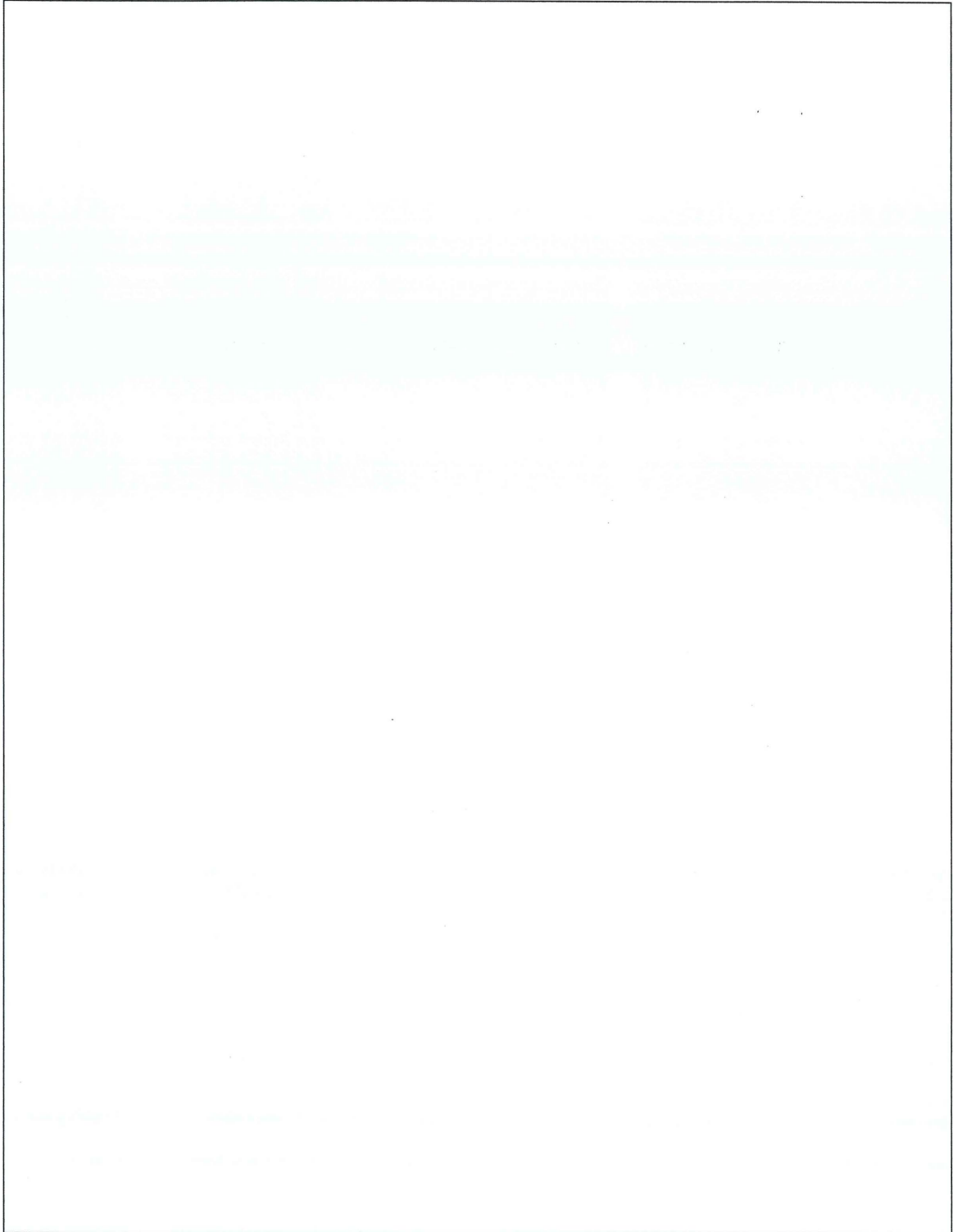
| Contact List |                            |
|--------------|----------------------------|
| Max          | 123 Major Street 55555555  |
| Minnie       | 567 Minor Street 66666666  |
| Callie       | 789 Median Street 77777777 |

| Add New Contact                       |                      |
|---------------------------------------|----------------------|
| Name:                                 | <input type="text"/> |
| Address:                              | <input type="text"/> |
| Phone:                                | <input type="text"/> |
| <input type="button" value="Submit"/> |                      |



Note that you cannot change anything in index.html but can only add jQuery scripts in script.js to achieve the above functionalities (only jQuery code is allowed in script.js). **Hint:** to prevent default form submission behavior when the submit button is clicked, you can use `e.preventDefault()` where `e` indicates the form submit event.

script.js



(2)(12%) Implement myservice.php to achieve the following functionalities: when it receives an HTTP GET request with query parameter "getallcontacts=true", it retrieves all contact records from the contacts table in the database, and send back to the client the HTML code for creating the new table rows containing the retrieved contact information. If it receives an HTTP POST request which carries name, address and phone number of a new contact, it inserts a new contact record into the contacts table in the database, and returns the HTML code for creating the new table row containing information of the newly added contact. You can assume that the client always enters valid values in all three input textboxes of name, address and phone number, before clicking the submit button.

myservice.php



**Question 4 (34%)** Implement a web app using the express.js framework, which is accessible at <http://localhost:3001/>. The web app achieves the following functionalities:

(i) If it is the very first time that a browser accesses <http://localhost:3001/> or if the last time when this browser accesses the page was at least one hour ago, an HTML page `video.html` is loaded, which displays a video `oceans.mp4`. An illustration is given in Fig. 4-1. When the video is clicked, the browser loads `index.html` (as shown in Fig. 4-2).

(ii) If it is not the first time that the browser accesses <http://localhost:3001/> and the last time when this browser accesses the web app was less than one hour ago, `index.html` is directly loaded, as shown in Fig. 4-2.

Fig. 4-1 Browser display of <http://localhost:3001/> in case (i)

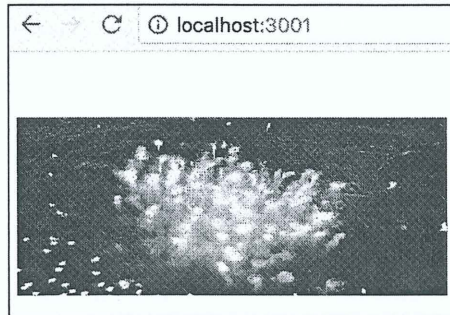
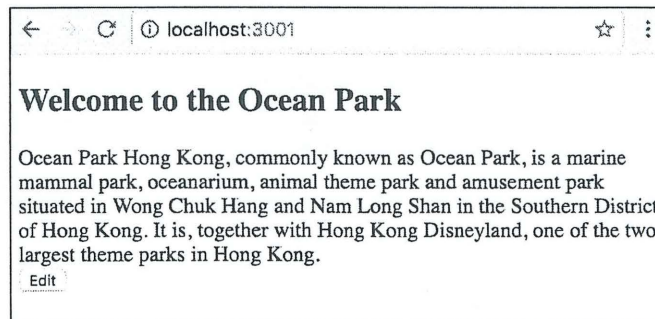
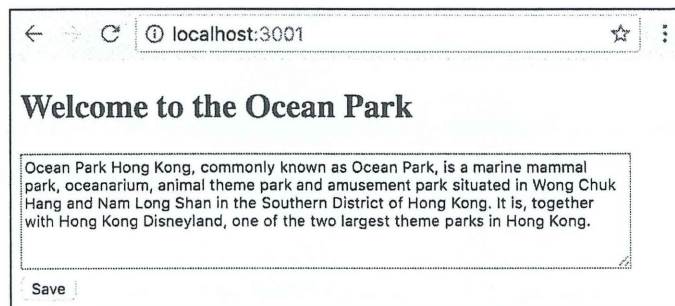


Fig. 4-2 Browser display of <http://localhost:3001/> in case (ii)



On the page as shown in Fig. 4-2, the paragraph of text describing the ocean park is not included in `index.html`, but loaded from `oceanpark.txt` on the server side. When the client clicks the “Edit” button, the page changes to Fig. 4-3: an editable text area appears and contains the paragraph of text as in the page view of Fig. 4-2, and a “Save” button replaces the “Edit” button.

Fig. 4-3 After clicking the “Edit” button



After editing the text in the text area (e.g., adding “My newly added description.” at the end) and clicking the “Save” button, the revised paragraph of text is sent to the server side and stored into oceanpark.txt. The page display becomes Fig. 4-4: the revised text paragraph shows up on the page, followed by the “Edit” button.

Fig. 4-4 After clicking the “Save” button



The main folder structure of your app is as follows:



video.html and index.html are given as follows:

#### video.html

```
<!DOCTYPE html>
<html>
<body>
  <a href="/">
    <video width="300" height="200" autoplay>
      <source src="videos/oceans.mp4" type="video/mp4">
    </video>
  </a>
</body>
</html>
```



## index.html

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  <script src="/javascripts/script.js"></script>
</head>
<body>
  <h2>Welcome to the Ocean Park</h2>
  <div id="mydiv">
    <div id="description"></div>
    <button id="editbutton">Edit</button>
  </div>
</body>
</html>
```

(1) (21%) Based on the given video.html and index.html, implement app.js, wservice.js and script.js to complete the app. Further clarifications/requirements are as follows:

- Part of the code in app.js is given. Add code to use the router instance created in wservice.js to handle all HTTP requests. Besides, you should add any other necessary code in app.js in order to achieve the app functionalities.
- In wservice.js, implement the server-side code for handling HTTP GET requests for <http://localhost:3031/>, <http://localhost:3031/getdescription>, and HTTP PUT requests for <http://localhost:3031/savedescription>. In the middleware handling HTTP GET requests for <http://localhost:3031/>, use the cookie technology to decide whether to send video.html or index.html back to the client. In the middleware handling HTTP GET requests for <http://localhost:3031/getdescription>, send oceanpark.txt back to the client. In the middleware handling HTTP PUT requests for <http://localhost:3031/savedescription>, save the revised text carried in the body of a PUT request into oceanpark.txt. The following API is given for your reference on how to store *data* into a file with the *filename* (data will replace the file's original content). In the callback function, send an empty string back to the client if file writing is successful and the error message otherwise.

```
var fs = require('fs');
fs.writeFile(filename, data, function (error) {
  // callback function
});
```

- Implement the client-side script.js using jQuery. When index.html has been loaded, send an AJAX HTTP GET request for <http://localhost:3031/getdescription>, and then load the received data into the correct element in index.html. When the "Edit" button is clicked, change the display of the page to that in Fig. 4-3. You can set the rows and cols of the <textarea> element to 6 and 70, respectively. When the "Save" button is clicked, send an AJAX HTTP PUT request for <http://localhost:3031/savedescription>, carrying the revised text paragraph in the text area in the body of the PUT request. If a success response is received from the server side, change the page display to that in Fig. 4-4; otherwise, prompt the error message. **Hint:** you may find the following API useful for registering a handler to an *event* on dynamically created HTML element(s) (as decided by *selector*):

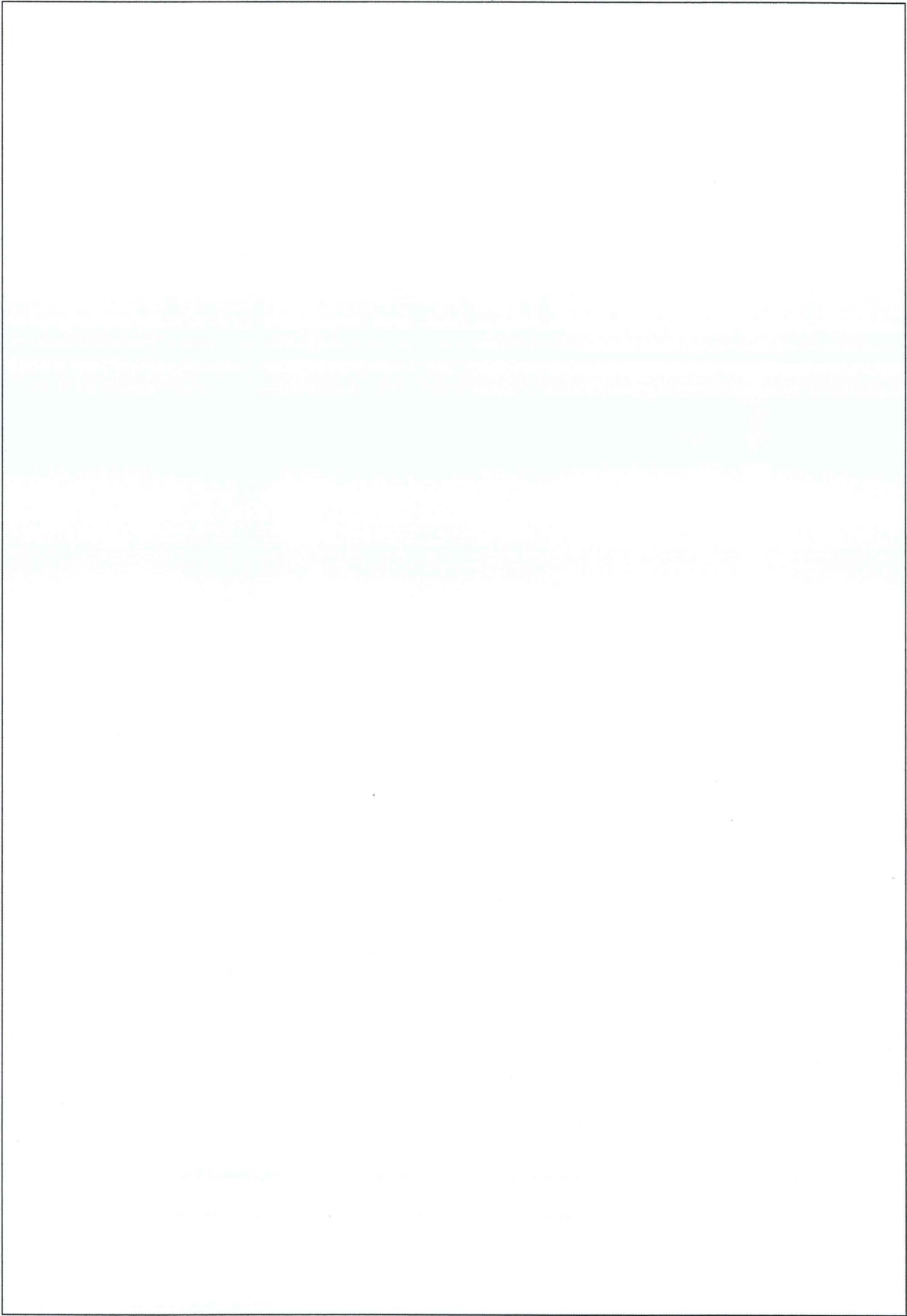


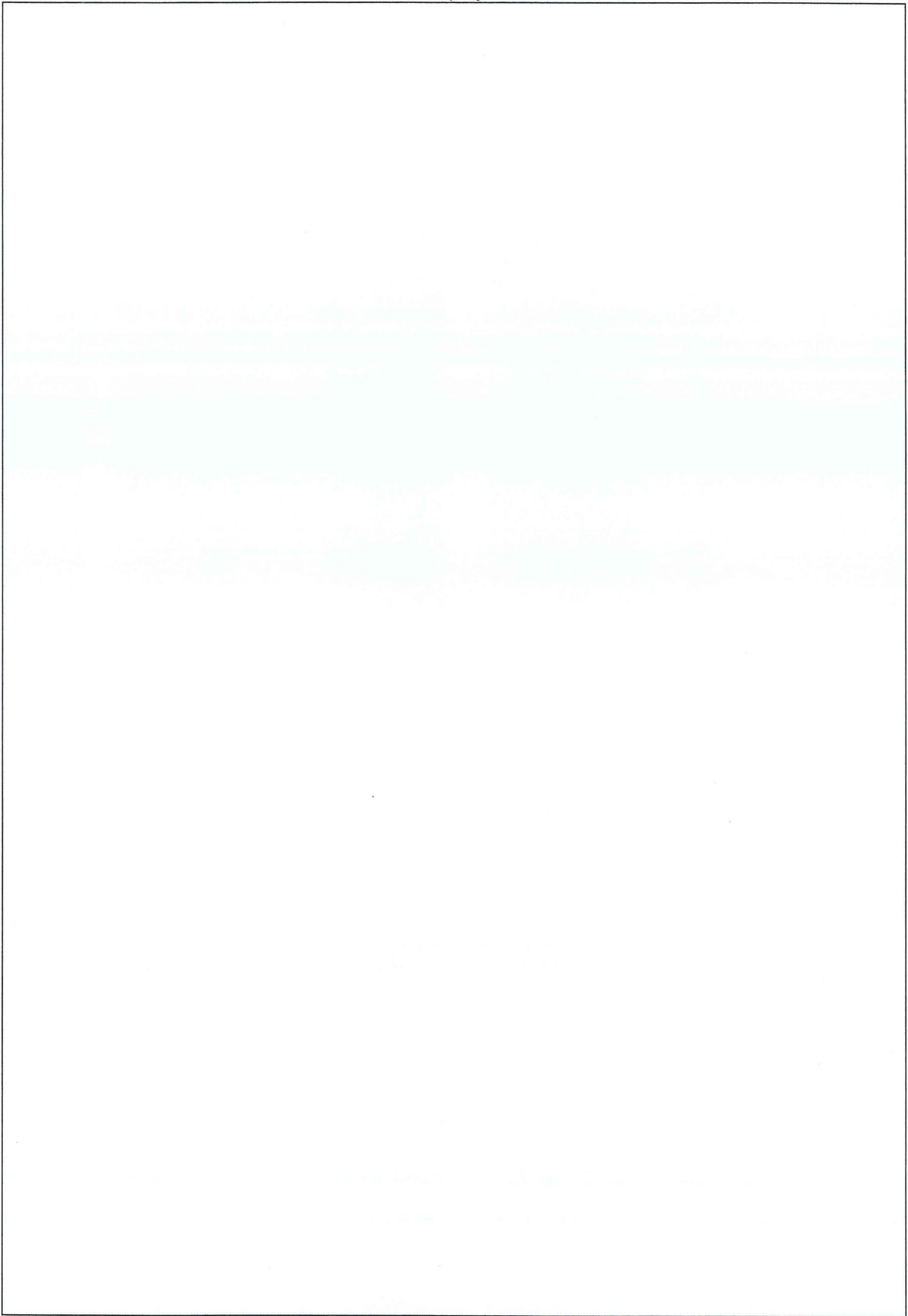
```
$('#body').on(event, selector, function() {  
    //event handler  
});
```

- You should not change anything in index.html and video.html in this sub question.

app.js

```
var express = require('express');  
var app = express();  
  
var path = require('path');  
var bodyParser = require('body-parser');  
  
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded({extended: false}));
```



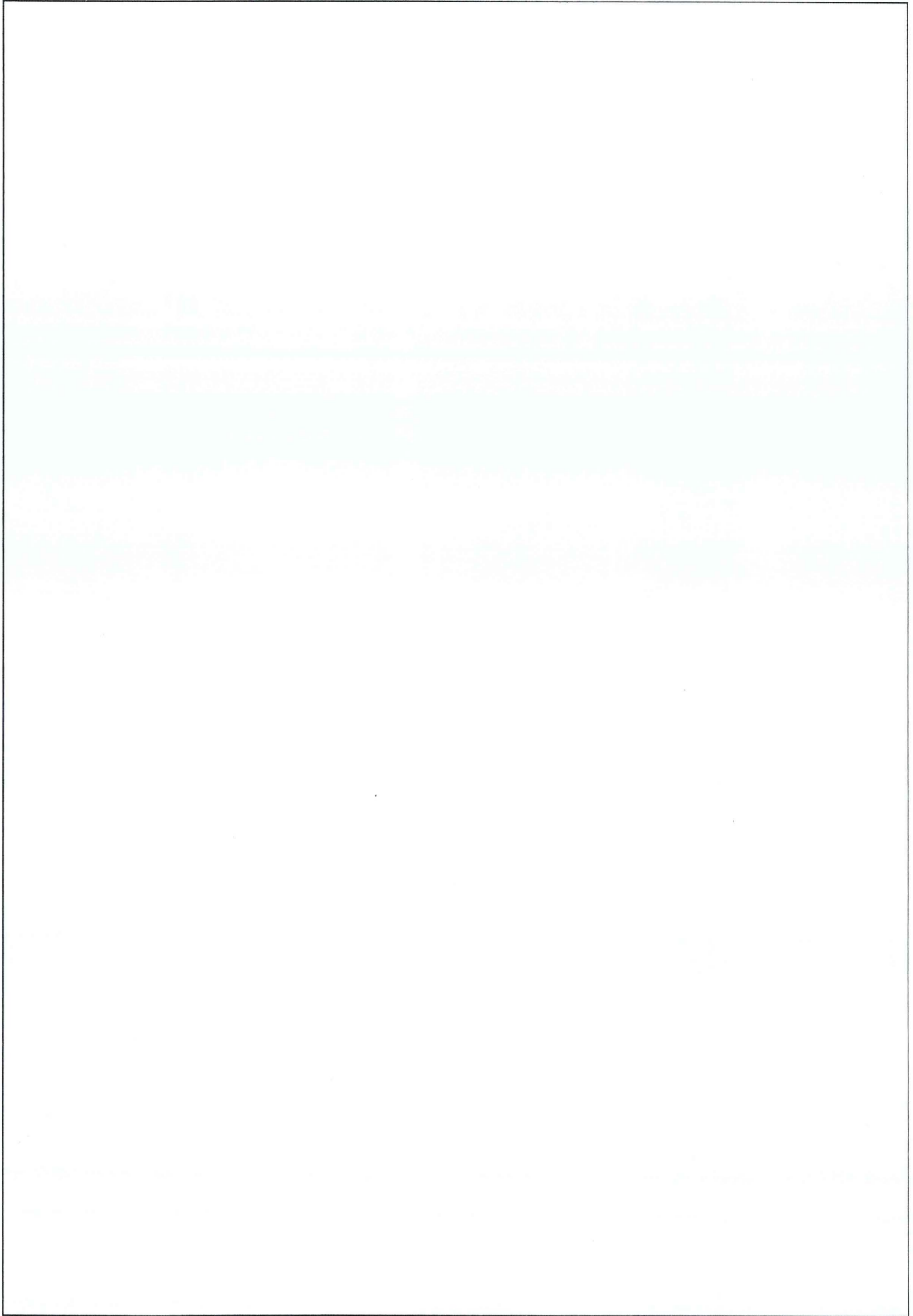


(2) (13%) Implement the save web app functionalities as you implemented in sub question (1), using AngularJS for implementing the client side. You can rewrite index.html and script.js using AngularJS (wherever necessary), while other files in the web app should remain the same.

index.html

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></script>
  <script src="/javascripts/script.js"></script>
</head>

</html>
```





**Question 5 (12%)** Implement a simple web page as shown in the figures below using React. Some products are displayed in a table underneath a checkbox, and the name of an out-of-stock product is shown in bold font. When the checkbox is not checked, the page display is as in Fig. 5-1; when the checkbox is checked, the page display is as in Fig. 5-2.

Fig. 5-1 when the checkbox is not checked

Only show products in stock

| Category    | Name              | Price    |
|-------------|-------------------|----------|
| Sports      | Football          | \$49.99  |
| Sports      | Baseball          | \$9.99   |
| Sports      | <b>Basketball</b> | \$29.99  |
| Electronics | Google Pixel 2    | \$849.00 |
| Electronics | <b>iPhone X</b>   | \$999.00 |
| Electronics | Samsung Note 8    | \$930.00 |

Fig. 5-2 when the checkbox is checked

Only show products in stock

| Category    | Name           | Price    |
|-------------|----------------|----------|
| Sports      | Football       | \$49.99  |
| Sports      | Baseball       | \$9.99   |
| Electronics | Google Pixel 2 | \$849.00 |
| Electronics | Samsung Note 8 | \$930.00 |

The app is mainly implemented by React code in index.js and app.js. index.js is given as follows. Suppose there is an index.html containing a <div> element with id "root", in which React elements will be rendered.

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import FilterableProductTable from './App';
const PRODUCTS = [
  {category: 'Sports', price: '$49.99', stocked: true, name: 'Football'},
  {category: 'Sports', price: '$9.99', stocked: true, name: 'Baseball'},
  {category: 'Sports', price: '$29.99', stocked: false, name: 'Basketball'},
  {category: 'Electronics', price: '$849.00', stocked: true, name: 'Google Pixel 2'},
  {category: 'Electronics', price: '$999.00', stocked: false, name: 'iPhone X'},
  {category: 'Electronics', price: '$930.00', stocked: true, name: 'Samsung Note 8'}
];
ReactDOM.render(
  <FilterableProductTable products={PRODUCTS} />,
  document.getElementById('root')
);
```

Complete the code in app.js below, to achieve the above web page. Especially, there are 12 lines of (partially) missing code, marked by "//TO COMPLETE", and you are only expected to complete those 12 lines of code.

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';

class ProductRow extends React.Component {
  render() {
    const product = ; //TO COMPLETE
```

```

const name = product.stocked ?
  product.name :
  <span style={{fontWeight: 'bold'}}>
    {product.name}
  </span>;

return (
  <tr>
    <td>{product.category}</td>
    <td>{name}</td>
    <td>{product.price}</td>
  </tr>
);
}
}

class ProductTable extends React.Component {
  render() {

    const inStockOnly = ; //TO COMPLETE

    const rows = [];
    this.props.products.map((product) => {
      if (inStockOnly && !product.stocked) {
        return;
      }
      rows.push(
        <ProductRow
          product={product}
        />
      );
    });

    return (
      <table>
        <thead>
          <tr>
            <th>Category</th>
            <th>Name</th>
            <th>Price</th>
          </tr>
        </thead>
        <tbody>{rows}</tbody>
      </table>
    );
  }
}

```

```

class CheckForm extends React.Component {
  constructor(props) {
    super(props);

                                                                    //TO COMPLETE
  }

  handleInStockChange(e) {

                                                                    //TO COMPLETE
  }

  render() {
    return (
      <form>
        <input
          type="checkbox"

          checked={                                                                    } //TO COMPLETE

          onChange={                                                                    } //TO COMPLETE

        />
        Only show products in stock
      </form>
    );
  }
}

class FilterableProductTable extends React.Component {
  constructor(props) {
    super(props);
    this.state = {

                                                                    //TO COMPLETE

    };
    this.handleInStockChange = this.handleInStockChange.bind(this);
  }

  handleInStockChange(                                                                    ) { //TO COMPLETE

    this.setState({

                                                                    //TO COMPLETE

    })
  }
}

```

```
}
render() {
  return (
    <div>
      <CheckForm
                                                                    //TO COMPLETE
                                                                    //TO COMPLETE
      />
      <ProductTable
        products={this.props.products}
                                                                    //TO COMPLETE
      />
    </div>
  );
}
}

export default FilterableProductTable;
```

— END OF PAPER —