TECHNISCHE UNIVERSITEIT DELFT
Faculteit Elektrotechniek, Wiskunde en Informatica

Midterm

TI1506 Web and Database Technology

Tuesday, 9 December 2014
13.45-15.45

CORRECT ANSWERS ARE HIGHLIGHTED IN GREEN

INSTRUCTIONS:

- This exam consists of 2 parts (DB and Web), and with a total of 36 multiple-choice questions. All questions are worth an equal number of points.
- The usage of books, notes, old exams, and other written resources is explicitly **FORBIDDEN** during the exam. The use of electronic aids such as smart-phones, laptops, etcetera, is **ALSO NOT** allowed.
- There is only one right answer for each question. If you think there are more, pick the best one.
- You are not allowed to make corrections on the multiple-choice answer form (MAF). You are therefore advised to first mark the answers on this exam and later copy them to the MAF. If you need to make corrections anyway, ask for a new form and copy all your answers to it
- You are not allowed to take the exam sheet with you after the exam. We will publish online the text of the exam together with its solutions
- Note that the order of the answers on your MAF form is not always A-B-C-D.
- Be sure to fill in all header information on the MAF. Enter your *studentnumber* on the form with digits as well as by filling the boxes.
- Sign the MAF. Without your signature, the form is not valid. Since you might forget this at the end, you are advised to do this at the start of the exam.

Good Luck!

Part 1 – Databases

QUESTION 1. Which of the following statements correctly describe the difference between the `HAVING` and the `WHERE` clauses in SQL?

A) The `WHERE` clause tests conditions on **individual rows** resulting from the evaluation of the `FROM` clause; the `HAVING` clause tests conditions on **aggregations** calculated after the application of the `GROUP BY` clause;
B) The `WHERE` clause tests conditions on **aggregated rows** resulting from the evaluation of the `FROM` clause; the `HAVING` clause tests conditions on **individual rows** resulting from the execution of sub-queries;
C) The `WHERE` clause tests conditions on **individual and aggregated rows** resulting from the evaluation of the `FROM` clause; the `HAVING` clause tests conditions **only** on **aggregations** calculated after the application of the `GROUP BY` clause;
D) The `WHERE` clause is part of the SQL standard, and can be used in any RDBMS systems; the `HAVING` clause is not part of the SQL standard, so it is not supported in **all** systems.

QUESTION 2. Suppose you have two relations: `A(col1, col2)` and `B(col3,col4)`. Which of the following SQL queries guarantee that the produced result-set will contain **only** tuples from `A` that have the value of **col1** attribute matching the value of **col3** attribute in `B`?

```
A) SELECT * FROM A LEFT OUTER JOIN B ON A.col1 = B.col3
B) SELECT * FROM A INNER JOIN B WHERE A IN B
C) SELECT * FROM A INNER JOIN B ON A.col1 = B.col3
D) SELECT * FROM A,B ON A.col1 <> A.col2
```

QUESTION 3. What is the role of the server module of typical enterprise database architectures?

A) To provide the user-friendly user interfaces
B) Only to handle the load balancing of third-party applications that use the database
C) To handle data storage, access, and search
D) None of the above

QUESTION 4. How many of the following are types of constraints that can be directly expressed in the schema of a data model?

[1] Domain Constraints
[2] Join Constraints
[3] Primary Key Constraints
[4] Semantic Constraints

A) 1,3
B) 2,3,4
C) 1,3,4
D) 1,2,3,4

QUESTION 5. How many of the following statements are true?

[1] The relational model is value-based
[2] The relational model never contains unspecified information
[3] A key is a set of attributes that uniquely identifies value based relations
[4] The same tuple can appear more than once in a relation.

A) 1
B) 2
C) 3
D) All

QUESTION 6. Which of the following statements are true?

[1] The conceptual schema provides a description of the whole database by means of the logical model adopted by the DBMS.
[2] The external schema provides a description of additional tables in the database in terms of data structures.

A) Only [1] is true
B) Only [2] is true
C) Both are true
D) Both are false

QUESTION 7. Which of the following statements are true?

[1] Data Manipulation Language (DML) specifies the internal schema.
[2] Data Definition Language (DDL) is often used for access authorization specification.
[3] View Definition Language (VDL) specifies user views/mapping to conceptual schema.

A) They are all true
B) Only [1] is true
C) Only [1] and [2] are true
D) Only [2] and [3] are true

QUESTION 8. What is the Cartesian product D_1 x D_2 of the following domains?

D_1 = ('pear', 'apple')
D_2 = ('Yellow', 'Green')

A) {<'pear', 'Green>, <'apple', 'Yellow'>}
B) {<'pear', 'Yellow'>, <'apple', 'Green'>}
C) {<'pear', 'Yellow'>, <'apple', 'Yellow'>, <'pear', 'Green>, <'apple', 'Green'>}
D) {<'Yellow', 'pear'>, <'Yellow', 'apple'>, <'Green', 'pear'>, <'Green', 'apple'>}

QUESTION 9. What is the result of the following logical expression in SQL?

```
NOT (((FALSE OR UNKNOWN) AND TRUE) OR TRUE)
```

A) TRUE
B) FALSE
C) UNKNOWN
D) NULL

The following questions consider the IMDB database, defined by the following SQL schema definition statements.

```
CREATE TABLE `actors` (
  `id` int(11) NOT NULL default '0',
  `first_name` varchar(100) default NULL,
  `last_name` varchar(100) default NULL,
  `gender` char(1) default NULL,
  PRIMARY KEY  (`id`)
)

CREATE TABLE `movies_directors` (
  `director_id` int(11) default NULL,
  `movie_id` int(11) default NULL,
)

CREATE TABLE `roles` (
  `actor_id` int(11) default NULL,
  `movie_id` int(11) default NULL,
  `role` varchar(100) default NULL,
)
```

```
CREATE TABLE `movies` (
  `id` int(11) NOT NULL default '0',
  `name` varchar(100) default NULL,
  `year` int(11) default NULL,
  `rank` float default NULL,
  PRIMARY KEY  (`id`),
)

CREATE TABLE `directors` (
  `id` int(11) NOT NULL default '0',
  `first_name` varchar(100) default NULL,
  `last_name` varchar(100) default NULL,
  PRIMARY KEY  (`id`),
)
```

QUESTION 10. You are required to retrieve from the database the list of actors that participated in at least one movie. The presence/absence of duplicates is not relevant. The result-set must be structured as follows:

first_name	last_name	movie

Which of the following queries is correctly producing the expected results?

A)
```
SELECT first_name, last_name, movie
FROM actors, roles, movies
WHERE actors.id = actor_id AND movie_id = movies.id
```

B)
```
SELECT first_name, last_name, name as movie
FROM actors, movies
WHERE actors.id = actor_id AND movie_id = movies.id
```

C)
```
SELECT first_name, last_name, name as movie
FROM actors, roles, movies
WHERE id = actor_id AND movie_id = id
```

D)
```
SELECT first_name, last_name, name as movie
FROM actors, roles, movies
WHERE actors.id = actor_id AND movie_id = movies.id
```

QUESTION 11. Which of the following queries retrieve the number of female actors in the IMDB database?

[1]
```
SELECT COUNT(gender)
FROM actors
WHERE gender <> "M" AND gender IS NOT NULL
```

[2]
```
SELECT COUNT(gender)
FROM actors
WHERE gender = "F"
```

[3]
```
SELECT COUNT(gender)
FROM actors
GROUP BY gender
HAVING gender = "F"
```

[4]
```
SELECT COUNT(*)
FROM actors as A, actors as B
WHERE A.id = B.id
GROUP BY A.gender
HAVING B.gender = "F"
```

A) Only 1
B) Only 2
C) 2 and 3
D) All

QUESTION 12. Which of the following queries (on the IMDB database) does not return the same result set as other ones?

```
[1] SELECT *
    FROM movies AS m WHERE id IN (
           SELECT id FROM movies WHERE year = 2000 AND rank < 7.3
    )
[2] SELECT *
    FROM movies AS m LEFT JOIN movies AS n ON m.id=n.id
    WHERE m.year=2000 AND n.rank<7.3
[3] SELECT *
    FROM movies WHERE year = 2000 AND rank < 7.3
[4] SELECT n.*
    FROM movies AS m LEFT JOIN movies AS n ON m.id=n.id
    WHERE m.year=2000 AND n.rank<7.3
```

A) 1
B) 2
C) 3
D) 4

QUESTION 13. Which of the following statements better describe the result set produced by following query

```
SELECT DISTINCT m.name
FROM movies AS m INNER JOIN roles AS r ON m.id = r.movie_id
WHERE r.role <> "James Bond"
```

A) The distinct names of all the movies in which there is at most one actor that plays the role of James Bond
B) The distinct names of all the movies in which no actor plays the role of James Bond
C) The distinct names of all the movies in which at least one actor does not play the role of James Bond
D) The distinct names of all the movies in which at most one actor does not play the role of James Bond

QUESTION 14. Consider the following query

```
SELECT movies.name
FROM movies, movies_directors, directors
WHERE movies.id = movies_directors.movie_id
       AND directors.id = movies_directors.director_id
```

Assuming **M** to be the cardinality of the *movies* table and **D** the cardinality of *directors* table. Which of the following formula correctly calculate the maximum amount of tuples that could be returned by the query above?

A) M + D
B) M
C) M x D
D) D

QUESTION 15. Suppose that the `movies_directors` relation in the IMDB database was created as follows:

```
CREATE TABLE movies_directors(
       movie_ID INT REFERENCES Movie(id) ON DELETE SET NULL,
       director_ID INT REFERENCES Director(id) ON DELETE CASCADE
)
```

Which of the following statements are true?

[1] If we try to delete a tuple from `movies_directors`, the tuple is not deleted. Instead, `movie_id` is set to `NULL`.
[2] If we delete a tuple from `movies_directors`, any tuples in `director` referred to by this tuple are also deleted.
[3] If we delete a tuple from `movie`, some attributes of `movies_directors` may have their values set to `NULL`.
[4] If we try to insert a tuple into `director`, with an `id` that is not referred to in `movies_directors`, the operation is rejected.
[5] If we try to insert a tuple into `movies_directors`, with an `id` that does not exist in `director`, the operation is rejected.

A) [3], [4] and [5]
B) [3] and [5]
C) [1], [2], and [3]
D) [1], [2], [3], and [5]

The following questions consider the relational schema described next. Attributes with the same name represent keys and foreign keys

Repair(WorkshopName) → *AutoWorkshop(WorkshopName)*
Repair(AutoLicense) → *Auto(AutoLicense)*

AutoWorkshop (WorkshopName, Address, Director)
Repair (WorkshopName, ReceiptNumber, AutoLicense, Type, Date, Cost)
Auto (AutoLicense, Owner)

QUESTION 16. Which of the following statement(s) about the above relational schema is/are correct?

[1] The same `Auto` can be repaired multiple times by the same `AutoWorkshop` in the same day
[2] Two `AutoWorkshop`s can perform a repair having the same `ReceiptNumber.`
[3] Two identical `Repair`s performed by the same `AutoWorkshop` must have the same cost.
[4] The `Director` of an `AutoWorkshop` cannot be the owner of an `Auto` repaired in his/her own `AutoWorkshop`.

A) [3] and [4] are true
B) Only [2] is true
C) [2] and [3] are true
D) [1] and [2] are true

QUESTION 17. Given the following SQL query

```
( SELECT WorkshopName
  FROM Repair
  WHERE Date BETWEEN 01/02/2014 AND 28/02/2014
  GROUP BY WorkshopName
  HAVING COUNT(*) > 50
)
  EXCEPT
(
  SELECT WorkshopName
  FROM Repair
  WHERE Type LIKE "%WINDSHIELD%"
)
```

Which answer best describes its meaning?

A) Retrieve the name of auto workshops that repaired at least 51 autos in February 2014 but never made a repair on windshields.
B) Retrieve the name of auto workshops that repaired at least 50 autos in February 2014 but never made a repair on windshields.
C) Retrieve the name of auto workshops that repaired at least 51 autos in February 2014 but made some repairs on windshields.
D) Retrieve the name of auto workshops that repaired at least 50 autos in February 2014 but made some repairs on windshields in the same month.

QUESTION 18. Consider the following SQL statements

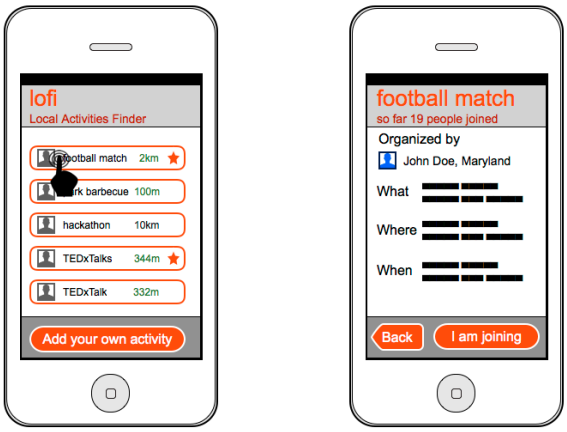
```
[1] INSERT INTO Auto (AutoLicense, Owner) VALUES ('XX-999-XX', NULL)
[2] UPDATE Auto SET Auto.AutoLicense = 'YY-1111-YY'
[3] DROP TABLE AutoWorkshop
[4] INSERT INTO Repair (WorkshopName, ReceiptNumber) VALUES
    ('DelftAutoRepair', '100')
```

How many of the above statements CAN cause a violation of an integrity constraint specified in the relational schema?

A) 1
B) 2
C) 3
D) 4

Part 2 – Web

The following questions will be related to the **LOFI (Local Activities Finder)** application



The application is very simple: once you open the Web page, you immediately see an overview of all activities that are going on in your geographical area.
The application consists of 3 pages:

- OVERVIEW page: this page shows a list of activities in the user's vicinity; the list is automatically updated every few seconds - the server sends all activities to every client; on the client filtering is performed to only show those activities that are within a range of 10km of the user's current location.
- DETAILS page: when the user is interested in a particular activity, he clicks on the activity and is shown a page containing more details; here, the user can also join the activity
- ADD page: finally, the OVERVIEW page also contains a button where the user can add his own activity to the list; the ADD page allows the user to fill in all the details; once the activity is send to the server, it is distributed to all other users using the application

---

QUESTION 19. In the LOFI application, what is a suitable setting of the `EXPIRES` field in the HTTP header?

A) In the HTTP request only, Expires should be set to 5 seconds
B) In the HTTP response only, Expires should be set to 5 seconds
C) In the HTTP request and response Expires should be set to 5 seconds
D) `EXPIRES` is not a field in the HTTP header

QUESTION 20. Which range of HTTP response status codes are resolved automatically by modern Web browsers?

A) 3XX
B) 4XX
C) 5XX
D) HTTP status codes are sent in the HTTP request message, instead of the response

QUESTION 21. For our application we have bought the domain name `lofi.org`.
Our server-side application runs on port 8080; it implements a URL route "retrieveActivities" which accepts two input parameters: the latitude and longitude of the user.

Let's assume a user currently at position (12.343/2.523) wants to retrieve the list of activities in his surroundings. Which of the following URLs does the user have to use?

A) `http://lofi.org?retrieveActivities:8080&latitude=12.343&longitude=2.523`
B) `http://lofi.org:8080?retrieveActivities:8080&latitude=12.343&longitude=2.523`
C) `http://lofi.org:8080/retrieveActivities&latitude=12.343&longitude=2.523`
D) `http://lofi.org:8080/retrieveActivities?latitude=12.343&longitude=2.523`

QUESTION 22. Client and server can use HTTP request/response header fields to "negotiate" about the form of the content (the server aims to send the content in a form the client understands). Which of the following combination of HTTP request and HTTP response fields represents such a "negotiation"?

A) Accept-MD5 & Content-MD5
B) Accept-Location & Content-Location
C) Accept-Encoding & Content-Encoding
D) Accept-Last-Modified & Last-Modified

QUESTION 23. Our application does not ask the user to login before using it. Is it nevertheless possible to reliably track and identify the individual user across the application?

A) Yes, the information in HTTP request headers is sufficient.
B) Yes, client IP address tracking is sufficient.
C) Yes, fat URLs can be used.
D) No, it is not possible.

QUESTION 24. In version 2 of LOFI, basic HTTP authentication will be implemented to provide users with the familiar login screen before they can use the application. What is the biggest issue of basic HTTP authentication?

A) Usernames do not have to be unique within an application
B) Username and password are sent unencrypted from client to server
C) Username and password are sent across a TCP connection
D) Users can choose weak passwords

QUESTION 25. To make our application a success we invested heavily in usability testing prior to the application's release. To measure usability we designed the four metrics shown below. Which one does not make sense in the context of our application?

A) The number of clicks necessary to add an activity
B) The amount of time until a user finds an activity of interest
C) The amount of scrolling on an Activity Detail page
D) The number of changes on the Local Activities Overview page

QUESTION 26. To make our application cutting-edge we decided to use features X1, X2 of HTML5.1, that are not available in HTML5. HTML5.1 is set to become a candidate recommendation in late 2016. What does this mean for us?

A) We have to wait until 2016 before our application can be used properly as the most popular browsers (Chrome, Firefox, Internet Explorer, Safari) are scheduled to implement X1 and X2 in late 2016.
B) X1 and X2 are already implemented in all modern browsers.
C) X1 and X2 may already be implemented in some modern browsers.
D) Every browser not implementing X1 and/or X2 will crash when loading our application.

QUESTION 27. To implement the "Add your own Activity" functionality we had the choice between HTML forms (i.e. the use of <form>) or a pure JavaScript-based solution. What is the advantage of the former over the latter?

A) There is no need to implement any client-side JavaScript code.
B) The form input can be validated by the browser (e.g. valid date, validate number).
C) The input data is send within a single HTTP request to the server.
D) HTML forms have a particular look that users are accustomed to - it reduces the cognitive load for the user.

QUESTION 28. To start our application development we built a simple HTML page, shown in **Figure 1**. When you open this Web page in a JavaScript-enabled browser with Internet access, what will be the list of shown activities?

A) Movie night, Park barbecue, hackathon
B) Football match, Park barbecue, hackathon
C) Movie night, Park barbecue, Theather
D) Football match, Park barbecue, Theater

QUESTION 29. Figure 2 contains some basic JavaScript. Which line(s) will lead to an error?

A) None, every line of the code will work
B) Line 23
C) Lines 25-27
D) Lines 26-27

QUESTION 30. In **Figure 3**, we applied the module design pattern to our code. Which line (or lines) will throw an error?

A) Lines 36-38
B) Lines 36-41
C) Line 42
D) Lines 42-43

QUESTION 31. Eventually, we decided for our application to not use Web forms. We want to implement functionality in JavaScript that retrieves activities details from the server when a user clicks on it. **Figure 4** contains a first implementation of this functionality. What is the main issue of this code?

A) A listener for the event "click" cannot be defined for list items
B) The JavaScript code will be executed before the DOM tree is loaded, thus no event listeners will be attached to the list items.
C) When new list items are added without reloading the entire Web page (using Ajax), no listeners will be attached to them.
D) Repeatedly reloading the entire Web page will lead to multiple event listeners being attached to the same list item.

QUESTION 32. Have a look at **Figure 4** again. Line 14 contains a reference to "this". Which object does "this" refer to in this setting?

A) Window object
B) Document object
C) Unordered list object
D) List Item object

QUESTION 33. On the server-side, we first developed a simple TCP-based prototype. The code is shown in **Figure 5**. The server (using node 0.11 functionality) is started on the terminal as usual: `node node1.js`. Two clients connect to the server at the same time via telnet, i.e. `telnet loci.org 8080`. The clients do not disconnect.
What is the output on the server's console after 20 seconds?

A)
```
Waiting for users to be active!
Transmitting list of activities
Transmitting list of activities
A new user listens for activities
A new user listens for activities
```

B)
```
Waiting for users to be active!
A new user listens for activities
Transmitting list of activities
A new user listens for activities
Transmitting list of activities
```

C)
```
Waiting for users to be active!
A new user listens for activities
Retrieving the list of activities
Transmitting list of activities
A new user listens for activities
Retrieving the list of activities
Transmitting list of activities
```

D)
```
Waiting for users to be active!
Retrieving the list of activities
Retrieving the list of activities
Transmitting list of activities
Transmitting list of activities
A new user listens for activities
A new user listens for activities
```

QUESTION 34. Same setup as in question 33. What is the output on one client's console after 20 seconds?

A)
```
Football
Picnic
Retrieving the list of activities
```

B)
```
Retrieving the list of activities
Football
Picnic
```

C)
```
Retrieving the list of activities
Activities updated, now #2
Football
Picnic
```

D)
```
Football
Picnic
Activities updated, now #2
Retrieving the list of activities
```

QUESTION 35. The next step in our server-side coding scheme, is a node.js-based Web server, using the express module. The code is shown in **Figure 6**. The server is started on the terminal as usual: `node node2.js 80`. In a Web browser, the following URLs are entered, one after the other.

```
http://lofi.org/
http://lofi.org/index.html
http://lofi.org/add&activ=chess
http://lofi.org/add?activity=climbing
http://lofi.org/
```

What output is shown in the Web browser in this order?

A)
```
Cannot GET /
Football,Picnic
Activity added!
Activity added!
Cannot GET /
```

B)
```
Cannot GET /
Cannot GET /index.html
Invalid activity!
Cannot GET /add?activity=climbing
Cannot GET /
```

C)
```
Football,Picnic
Football,Picnic
Football,Picnic,chess
Football,Picnic,climbing
Football,Picnic
```

D)
```
Football,Picnic
Cannot GET /index.html
Cannot GET /add&activ=chess
Activity added!
Football,Picnic,climbing
```

QUESTION 36. To avoid having to refresh the entire OVERVIEW page from our LOFI application every few seconds, we want to make use of Ajax. Given the code in Figure 6, what needs to be changed to make it compatible with client-side Ajax technology?

A) The data has to be sent to the client in JSON format.
B) Nothing has to be changed on the server, the client can interpret the data.
C) Another URL route "/activities" needs to be added, such that the client can access the list of activities.
D) The server-side code needs to be rewritten completely with Ajax in mind, changing a few lines of server-side code is not sufficient.

Figure 1

```
<!doctype html>
<html>
<head>
    <title>Local Activities Finder</title>
    <script src="http://code.jquery.com/jquery-1.11.1.min.js">
  </script>
    <script>
                    $(document).ready(function () {
                            var act = document.getElementById("activities");
                            var listLIs = act.getElementsByTagName("li");
                            listLIs[0].innerHTML = "Movie night";
                    });
                    listLIs[2].innerHTML = "Theater";
    </script>
</head>

<body>
    <header>
            <h1>lofi</h1>
            <h2>Local Activities Finder</h2>
    </header>
    <main>
            <div id="activities">
                    <ul>
                            <li data-id="122" data-latitude="12.43434"
                 data-longitude="44.21" data-attendance="1">
                 Football match</li>
                            <li data-id="342" data-latitude="12.12332"
                 data-longitude="44.09" data-attendance="0">
                 Park barbecue</li>
                            <li data-id="424" data-latitude="11.99"
                 data-longitude="43.95" data-attendance="0">
                 hackathon</li>
                    </ul>
            </div>
            <div id="actions">
                    <button text="Add your own activity" id="addActivityButton">
            </div>
    </main>
</body>
</html>
```

Figure 2

```
1  function Activity(title, organizer, what, where, latitude,
2                    longitude, when) {
3    this.title = title;
4    this.organizer = organizer;
5    this.what = what;
6    this.where = where;
7    this.latitude = latitude;
8    this.longitude = longitude;
9    this.when = when;
10   this.attendance = 1;
11 }
12
13 Activity.prototype.addAttendingUser = function(t) {
14                                       this.attendance += 1;};
15
16 var act1 = new Activity("Football match", "John Doe",
17             "A football match in the local park", "Park II",
18             "44.23", "12.34", "9:30am");
19 act1.addAttendingUser(1);
20 var act2 = new Activity("hackathon", "John Smith",
21             "A hackathon at the library", "Library", "44.44",
22             "12.21", "4:00pm");
23 act2.addAttendingUser(10);
24
25 act1.changeLocation("44.40","12.34");
26 act2.changeLocation("dummy1","dummy2");
27 act2.changeLocation(44.40, 12.21);
28
29 /* lets define the method */
30 Activity.prototype.changeLocation = function(latitude, longitude) {
31     this.latitude = latitude;
32     this.longitude = longitude;
33 }
```

Figure 3

```
1  var activitiesModule = (function () {
2
3    var numActivities = 0;
4
5    /* 'public' members; return accessible object */
6    return {
7
8      Activity : function(title, organizer, what, where,
9                          latitude, longitude, when) {
10           this.title = title;
11           this.organizer = organizer;
12           this.what = what;
13           this.where = where;
14           this.latitude = latitude;
15           this.longitude = longitude;
16           this.when = when;
17           this.attendance = 1;
18       },
19
20     incrActivityCounter : function () {
21       numActivities++;
22     },
23
24     decrActivityCounter : function () {
25       if(numActivities>0) {
26         numActivities--;
27       }
28     },
29
30     getNumActivities : function() {
31       return numActivities;
32     }
33   };
34 })();
35
36 var act1 = activitiesModule.Activity("Football match",
37             "John Doe", "A football match in the local
38             park", "Park II", "44.23", "12.34", "9:30am");
39 var act2 = new activitiesModule.Activity("hackathon",
40             "John Smith", "A hackathon at the library",
41             "Library", "44.44","12.21", "4:00pm");
42 act1.when;
43 act2.when;
44 activitiesModule.incrActivityCounter(2);
45 activitiesModule.getNumActivities();
```

Figure 4

```
<!doctype html>
<html>
<head>
    <title>Local Activities Finder</title>
    <script src="http://code.jquery.com/jquery-1.11.1.min.js">
  </script>
    <script>
            var main = function() {
                    var list = document.getElementsByTagName("li");
                    for(var i=0; i<list.length; i++) {
                            document.getElementsByTagName("li")[i].onclick =
                                    showDetails;
                    }
                    console.log(this.toString());
            }
            $(document).ready(main);
            function showDetails() {
              /* make a request to the server and return details
              page */
            }
    </script>
</head>

<body>
    <header>
            <h1>lofi</h1>
            <h2>Local Activities Finder</h2>
    </header>
    <main>
            <div id="activities">
                    <ul>
                            <li data-id="122" data-latitude="12.43434"
                 data-longitude="44.21" data-attendance="1">
                 Football match</li>
                            <li data-id="342" data-latitude="12.12332"
                 data-longitude="44.09" data-attendance="0">
                 Park barbecue</li>
                            <li data-id="424" data-latitude="11.99"
                 data-longitude="43.95" data-attendance="0">
                 hackathon</li>
                    </ul>
            </div>
            <div id="actions">
                    <button text="Add your own activity" id="addActivityButton">
            </div>
    </main>
</body>
</html>
```

Figure 5

```
const
    net = require('net'),
    observer = require('observed');

var activities = ["Football", "Picnic"];

server = net.createServer(function(connection) {

    /* setTimeout calls the implemented function after 5000
    milliseconds (5 seconds) */
    setTimeout(function() {
            console.log("A new user listens for activities");
            connection.write("Retrieving the list of activities\n");
    }, 5000);

    console.log("Transmitting list of activities");
    for(var i=0; i<activities.length; i++)
            connection.write(activities[i]+"\n");

    /* send the upated activities to the client when the
    object 'activities' changes */
    var watcher = observer(activities).on("change", function() {

            connection.write("Activities updated, now #"+
                    activities.length+":\n");
            for(var i=0; i<activities.length; i++)
                    connection.write(activities[i]+"\n");
    });

    /* when a client sends an activity to the server, add it
    to the activities array */
    connection.on('data', function(data) {
            /* trim whitespaces from string */
            var trimmed = data.toString().replace(/(\r\n|\n|\r)/gm,"");
        console.log("Data received from the client: "+trimmed);
        activities[activities.length]=trimmed;
    });

    /* a client disconnects */
    connection.on('close', function() {
            console.log("A user is abandoning us");
    });
});

server.listen(8080, function() {
    console.log("Waiting for users to be active!");
});
```

Figure 6

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4
5  var app;
6
7  var port = process.argv[2];
8  app = express();
9  http.createServer(app).listen(port);
10
11 var activities = ["Football", "Picnic"];
12
13 app.get("/", function(req,res) {
14     res.send(activities.toString());
15 });
16
17 app.get("/add", function(req,res) {
18     var query = url.parse(req.url, true).query;
19     var activ = (query["activity"]!=undefined) ? query["activity"] : "";
20
21     if(activ.length>0) {
22             activities[activities.length]=activ;
23             res.send("Activity added!");
24     }
25     else {
26             res.send("Invalid activity!");
27     }
28 });
```