Session: 18

HTML5 Web Storage

Objectives

- Explain Web storage in HTML5
- Explain session storage
- Explain local storage
- Explain the Indexed DB API
- Describe a native app
- Explain the difference between native apps and HTML5 apps
- Describe the advantages of native and HTML5 apps
- List the steps to convert HTML5 apps to native apps

Introduction

Traditionally, over the last few decades, Web applications have been using cookies to store small amounts of information on a user's computer.

A cookie is a file that stores user-related information and may either be temporary or permanent.

A cookie can be created for login details which can be saved for a specified period on a user's computer.

- Drawbacks of cookies are as follows:
  - Cookies slow down the performance of Web application, as they are included with every HTTP request
  - Cookies cannot be considered as safe means for transmission of sensitive data
  - Cookies cannot store large amount of information, as they have a limitation of size of 4 KB
- W3C has designed a specification named Web Storage API which offer a solution to store data on the client-side

Web Storage in HTML5

- Is a W3C specification and certain browsers refer to it as ‘DOM Storage’.
- Provides functionality for storage of data on the client-side that is on user’s machine.
- Stores data that can cater for both temporary as well as permanent needs.
- Offers more control than traditional cookies, and is easy to work with.
- Was originally a part of the HTML5 specification, but now it is present in a separate specification and stores a maximum of 5 MB of information per domain.

Web Storage versus Cookies

- Some key differences between cookies and Web storage are as follows:

Cookies are meant to be read on the server-side, whereas Web storage is available only on the client-side.

Cookies are sent along with each HTTP request to the server, whereas Web storage data is not carried over to the server.

Cookies result in bandwidth overhead and thus lead to high costs, as they are sent with each HTTP request. The Web storage is stored on the user's hard drive, so it costs nothing to use.

With cookies, the information data that could be stored is 4 KB, whereas with Web storage, a large amount of data can be stored upto 5 MB.

Browser-specific Web Storage

Web storage is browser-specific and the location where the Web storage data is stored depends on the browser.

Each browser's storage is separate and independent, even if it is present on the same machine.

HTML5 Web storage is implemented natively in most Web browsers, so one can use it even when third-party browser plug-in is not available.

- Following table lists the support of various browsers for HTML5 Web storage:

Browser	Version
IE	8.0+
Firefox	3.6+
Safari	4.0+
Chrome	5.0+
Opera	10.5+

Exploring Web Storage 1-2

Two types of HTML5 Web storage are namely, session storage and local storage.

Both session and local storage enable to store around 5 MB of data per domain.

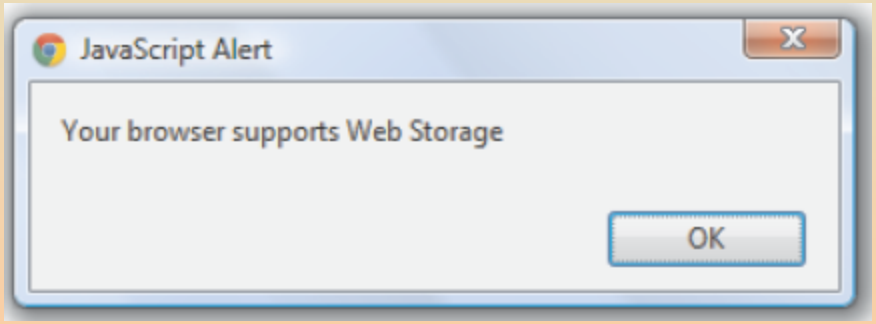
To check for browser support of HTML5 Web storage, a property named **localStorage** or **sessionStorage** is available as a global variable for the window object.

If there is no support, the **localStorage** or **sessionStorage** property will be undefined.

Exploring Web Storage 2-2

- Code Snippet demonstrates the script to check the support for HTML5 Web storage in the browser.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Support for Web Storage</title>
    <script>
    function checkSupport() {
                if ( ( 'sessionStorage' in window) &&
  window['sessionStorage'] !== null)
      {
        alert("Your browser supports Web Storage");
        return;
      }
      alert("Your browser does not support Web Storage");
    }
    </script>
  </head>
  <body onload="checkSupport();">
  </body>
</html>
```



Session Storage 1-6

- Keeps track of data specific to one window or tab and discards it as soon the user closes the tab (or window) that he/she was working with.
- Lasts for the entire duration of the session and hence, is not persistent.
- Makes use of named key/value pairs which are enclosed within double quotes.
- Stores the data using the named key, whereas the data is retrieved by referring to that key.
- Key is a string, whereas the value stored in the key can be of any data type such as string, boolean, integer, or float. Regardless of the type of data that is stored, it is actually stored internally as a string.
- Storing and retrieving data of other types requires the use of functions to convert them into the appropriate data types.

Session Storage 2-6

- Following table lists some examples of named key/value pairs belonging to various data types.

Key	Value
Name	Sarah
book	C Programming
Email	info@me.com
car	Toyota Innova
age	28
uservalid	true

Session Storage 3-6

- Several operations that can be performed with the **sessionStorage** object are as follows:

Storing and retrieving data - setItem() and getItem() methods are used to store and retrieve data from session storage respectively.

- Syntax to use `setItem()` and `getItem()` methods is as follows:

- **To assign data**

```
sessionStorage.setItem(key, value);
```

where,

`key`: Is the named key to refer to the data.

`value`: Is the data to be stored.

Session Storage 4-6

- **To retrieve data**

```
var item = sessionStorage.getItem(key);
```

where,

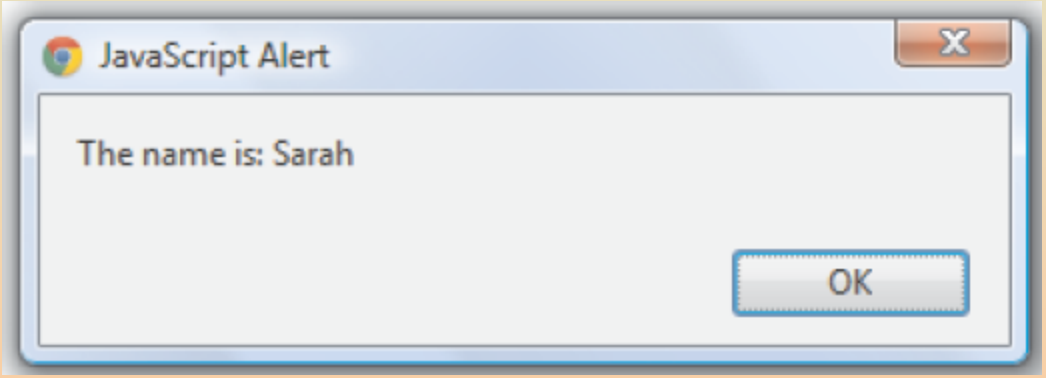
`item`: Is the variable into which the data will be saved.

`key`: Is the named key to refer to the data.

Session Storage 5-6

- Code snippet demonstrates how to set and retrieve a name using `sessionStorage` object.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Working with Session Storage</title>
  <script>
  function testStorage() {
     if (('sessionStorage' in window) && window['sessionStorage'] !==
   null)
    {
      sessionStorage.setItem('name', 'Sarah');
      alert('The name is: ' + sessionStorage.getItem('name'));
    }
  }
  </script>
  </head>
  <body onload=" testStorage()
  </body>
</html>
```



Session Storage 6-6

- **Removing data**

```
sessionStorage.removeItem(key);
```

where,

`key`: Is the named key to refer to the data.

- **Clearing data**

```
sessionStorage.clear();
```

Local Storage 1-4

- Enables to save data for longer periods on the user's computer, through the browser.
- Data is persistent and can be retrieved when a user visits the site again.
- Is used, if data needs to be stored for more than a single session.
- Works in a similar fashion as session storage.
- Uses the same functions, such as setItem(), getItem(), removeItem(), and clear().

Local Storage 2-4

- Code snippet demonstrates the use of local storage to store the value of **username** field and later, retrieve the value in another Web page.

```
<!DOCTYPE html>
<html>
  <title> Local Storage </title>
  <script>
  function store() {
    if (('localStorage' in window) && window['localStorage'] !== null) {
      var username = document.getElementById('username').value;
      localStorage.setItem('username', username);
    } else {
      alert ('your browser does not support storage');
    }
  }
  function cancel_store() {
    if (('localStorage' in window) && window['localStorage'] !== null) {
      localStorage.removeItem('username');
    } else {
      alert ('your browser does not support storage');
    }
  }
```

Local Storage 3-4

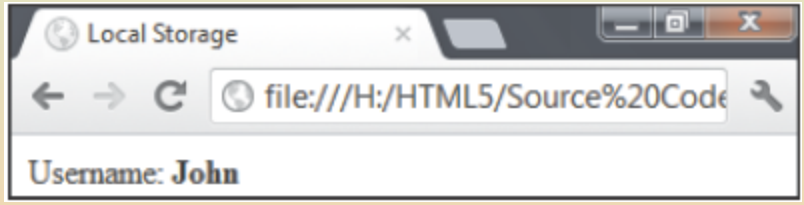
- Code snippet demonstrates the use of local storage to store the value of **username** field and later, retrieve the value in another Web page.

```
    </script>
  </head>
  <body>
    <form method="get" action="success.html">
      Username: <input type="text" id="username" value="" size="20"
  onblur="store()"/>
      <input type="submit" value="Submit"/>
      <input type="reset" Value="Cancel" onclick="cancel_store()"/>
  </body>
</html>
```

Local Storage 4-4

- Code snippet shows the success.html page that retrieves value from the local storage and displays it in the browser.

```
<!DOCTYPE html>
<head>
  <title> Local Storage </title>
  <script>
    function print() {
      var username = localStorage.getItem('username');
      document.getElementById('lblMsg').innerHTML = 'Username: is <b>'+
username+'</b>';
    }
  </script>
</head>
<body onload="print()">
  <label id="lblMsg"></label><br>
</body>
</html>
```



Indexed Database API 1-3

A database is an organized collection of data.

Databases, such as relational database stores the data in the form of tables.

A table comprises rows and columns that are used to store data.

The representation of data from a table is in the form of records.

HTML5 has introduced a new Web Storage API which can host Web databases locally within the user browser.

Web databases are not like relational databases in terms of functionality.

Indexed Database API 2-3

Indexed Database API is a specification also known as IndexedDB.

It is basically an object store that can be used to store and manipulate data on the client-side.

The object store is the primary storage mechanism that stores the object in the database managed locally within the browser.

It enables to create an object store of a particular type in which objects can be persisted using JavaScript.

IndexedDB enables to create Web applications with rich query abilities and which can work both online and offline.

IndexedDB supports two types of API namely, synchronous and asynchronous.

The synchronous API can be used with WebWorkers, whereas asynchronous API can be used for Web applications.

Indexed Database API 3-3

IndexedDB API is implemented using **window.indexedDB** object.

Browsers implement the IndexedDB object with their own prefixes. For example, Chrome browser uses the **webkit** prefix, whereas Mozilla supports **–moz** prefix.

- Following table lists the browser support for the IndexedDB API.

IE	Firefox	Chrome	Safari	Opera	iOS Safari
6.0	-	-	-	-	3.2
7.0	8.0moz	-	-	-	4.0-4.1
8.0	9.0moz	16.0webkit	5.0	-	4.2-4.3
9.0	10.0moz	17.0webkit	5.1	11.6	5.0
10.0	11.0moz	18.0webkit	6.0	12.0	-
-	12.0moz	19.0webkit	-	-	-

Indexed DB API 1-2

- Some of the basic constructs of IndexedDB API are as follows:

Database - The IndexedDB database comprises more than one object store. Each database contains a name that identifies the origin of the database and a version number which identifies the lifetime of the database.

Object Store - Is the main mechanism to store data in a database. They hold the data stored in the database in the form of records.

Keys - Each record stored in the database is identified by a unique key.

Values - Are the data stored in the records.

Key Path - Is a string that defines how the browser should extract key from a value. The key from a value can be extracted either in the object store or index.

Index - Is used when the data from the object store is retrieved based on some other values other than a key.

Transaction - Any addition or retrieval of the data in a database is performed by using transaction. Each transaction has a mode, scope, and a request list

Indexed DB API 2-2

- Some of the other basic constructs of IndexedDB API are as follows:

Requests - Operations, such as reading or writing on the database is performed using a request. Each request contain attributes, such as flag, source object, result, and error.

Cursor - Is a mechanism used to retrieve multiple records from a database.

Key Range - Records from the object stores and indexes are retrieved using keys or key ranges. A key range refers to retrieval of data between specified bounds based on the keys.

Implementing IndexedDB API 1-6

- Steps to implement the IndexedDB API in a Web application are as follows:

1. Open a database
2. Create an object store
3. Start a transaction
4. Perform some database operations, such as add and retrieve
5. Work with the retrieved results

Implementing IndexedDB API 2-6

- **Opening a Database**

- Code snippet shows the code to open a database

```
var indexedDB = window.indexedDB || window.webkitIndexedDB ||
window.mozIndexedDB || window.msIndexedDB;
var request = indexedDB.open("CompanyDB", 1);
request.onsuccess = function (event) {
. . .
};
request.onerror = function (event) {
console.log("IndexedDB error: " + event.target.errorCode);
};
```

Implementing IndexedDB API 3-6

- **Updating Version of a Database**

After the database is opened, it can be structured by providing a version number which helps to set up the database.

- Code snippet shows the code that specifies the version number to the database

```
var setVrequest = db.setVersion("1.99");
setVrequest.onsuccess = function(event) {
. . .
}
```

Implementing IndexedDB API 4-6

- **Creating the Object Store**

Structure of IndexedDB database facilitates the storage of multiple object stores.

Object store is created using createObjectStore() method which accepts two arguments namely, the store name and a parameter object.

- Code snippet demonstrates the code to create an object store named employee in the CompanyDB database.

```
var employeeData = [
{ name: "John Smith", email: "john@company.com" },
{ name: "Jill Patrick", email: "jill@company.com" },
{ name: "Rock Ethan", email: "rock@company.com" },
{ name: "Daniel Andrew", email: "daniel@company.com" }
];
var objectStore = db.createObjectStore("employee", {
keyPath: "id", autoIncrement: true });
for (i in employeeData) {
  objectStore.put(employeeData[i]);
  alert("Record added");
}
```

Implementing IndexedDB API 5-6

- **Creating a Transaction**

To perform database operation, such as retrieving data from the object store, IndexedDB provides a IDBTransaction object.

This object can be created in three mode namely, read-only, read-write, and snapshot.

- Code snippet demonstrates the code to retrieve data from the employee object store using the get() function of the transaction object.

```
var trans = db.transaction(["employee"],
IDBTransaction.READ_WRITE).objectStore("employee");
var request = trans.get(2);
request.onerror = function(event) {
  // Handle errors!
};
request.onsuccess = function(event) {
  // Do something with the request.result!
  alert("Name: " + request.result.name);
};
```

Implementing IndexedDB API 6-6

- **Opening a Cursor**

Cursor is used to retrieve multiple records from an object store.

They can be used when the value of key path is not known. They are part of a transaction and are opened for a particular object store.

- Code snippet demonstrates the code to retrieve multiple records from the employee object store.

```
store = db.transaction("employee").objectStore("employee");
store.openCursor().onsuccess = function(event) {
  var cursor = event.target.result;
  if (cursor) {
        alert("Name for id " + cursor.key + " is " +
cursor.value.name);
    cursor.continue();
  }
};
```

Limitations of IndexedDB API

- Design limitations for IndexedDB API used for client-side storage of data are as follows:

Internationalized sorting deals with sorting of string data. As the database does not follow any international order for storing data, internationalized sorting is not supported by the API.

IndexedDB API does not synchronize client-side database with the server-side databases.

IndexedDB API supports querying the client-side database, but does not support the use of operators, such as LIKE that is used by Structured Query Language (SQL).

Converting HTML5 apps to Native apps

A native application also known as native app is an application program that is built for using it on a particular device or platform.

A native app, when compared with Web app is installed on a device and has a faster response, because it has a direct user interface.

BlackBerry Messenger (BBM) is a native app available on blackberry mobile devices.

Difference between Native Apps and HTML5 Apps

HTML5 Web apps are accessible and used on any devices through Web browser similar to the mobile Web site.

Web apps have the ability of offline access which means that the user need not have a network connection.

- Following table lists differences between the native apps and HTML5 apps.

Native Apps	HTML5 Apps
Native Apps runs on iOS and Android devices that can be downloaded or purchased from the online app stores.	HTML5 Apps runs on a Web server, usually in a Web browser.
Native Apps use programming language, such as Java for Android devices and Objective C for iOS devices.	Web developers use HTML, JavaScript, and CSS. They need to acquire the skills of Java and objective C for writing native applications.

Advantages of HTML5 Apps

- Main advantage of using HTML5 is to create applications that executes on a wide range of devices easily. Some of the reasons to develop HTML5 applications are as follows:

Users cannot identify the differences - Cannot identify whether they are working on a hybrid HTML5-native application or a fully native application or an HTML5 application.

Users adjust styles for devices - HTML5 apps can be viewed on any devices that contains Web browser.

Upcoming functionalities - HTML5 does not support all features on a device, but it is coming up with new functionalities.

Improving Performance - Many developers learn new methods to improve the performance of Web.

Independent device - If the developers want that their application to be used by a large number of users, then they should design and develop applications for both mobile users as well as desktop users.

Developers are not locked in app stores - HTML5 developers are not restricted to an app store. Instead, they can create applications and sell them like any other Web page.

Advantages of Native Apps

- Major advantage of native apps over HTML5 apps is that they are faster than HTML5 apps. Native apps provide more benefits over HTML5 apps. These are as follows :

Providing access to device hardware - There are no APIs available for accelerometers, cameras, or any other device hardware for HTML5 apps.

Uploading Files - Native apps can access the file system in Android and some files in iOS. However, the HTML5 file API does not work on Android or iOS.

Push Notifications - The push notifications are sent always with an open IP connection to applications on the iOS device.

Accessing device files - Native apps communicate with files on devices, such as contacts and photos. However, these files cannot be seen from HTML5 app.

Superior graphics than HTML5 - HTML5 has a canvas element, but it will not create a full 3D experience.

Offline access - HTML5 provides access to offline Web applications. However, a native app is stored on local machine, so the users does not require access to the Web to work with the application.

Converting HTML5 Apps to Native Apps

- Users have a choice of developing their application in HTML5 and convert them into a native app
- Users can use some tools to convert an HTML5 app to Native app and they are as follows:

PhoneGap - Is an HTML5 app that allows the user to create native apps with Web technologies and is accessible to app stores and APIs.

Appcelerator - Is a cross-platform mobile application development support and allows the users to create Android, iOS, and mobile Web apps.

Summary

- Web Storage is a W3C specification that provides functionality for storing data on the client-side for both temporary as well as permanent needs.
- HTML5 Web applications make use of Web storage to implement client-side persistent storage and they are: session storage and local storage.
- Session storage keeps track of data specific to one window or tab.
- The setItem() and getItem() methods are used to store and retrieve the data from session storage.
- Local storage enables to save data for longer periods on the user's computer, through the browser.
- IndexedDB API is basically an object store that can be used to store and manipulate data on the client-side.
- A native application also called as native app is an application program that is built for a particular device or platform.