Chapter 8

Introducing Geolocation and Offline Support for Data

You are visiting a city for the first time and looking for sight-seeing locations close to your hotel. However, you do not know where to go and which places to look for. Therefore, it will be helpful if you have some application that can help you locate such places and provide feedback for the same. This can be done by implementing Geolocation API.

In addition, you may need that users are able to access a Web page even when they are not connected to the Internet.

This chapter introduces you to the Geolocation API. Further, it discusses how to implement offline support in websites.

Objectives

In this chapter, you will learn to:

- Implement Geolocation
- Implement Offline Support

Implementing Geolocation

ShopHere is a large retail store based in Ohio. The store offers clothes, accessories, and footwear. In addition, it sells the home furnishing goods and electronic items, such as refrigerator, air conditioner, laptops, and mobile devices. As part of its business strategy to attract new customers and build a stable customer base, the store offers discounts and special deals on products. In addition, the official website of the store has a feature that enables users to share their current location, based on which, they are guided to reach the nearest store.

Such a feature is incorporated in websites by using the Geolocation API. The Geolocation API-enabled website can locate users' current geographical location, display points of interest around that location, or guide the users from their current location to a target destination.

Implementing the Geolocation API

The Geolocation API allows a website to retrieve the current geographical location of a user. This API enables you to create applications that guide users how to reach a target location from their current location. An example of such an application is a map application that gives directions to a target location.

The users' location is not retrieved just through code or browser. Instead, the Geolocation API uses certain features, such as Global Positioning System (GPS), Internet Protocol (IP) address of a device, nearest mobile phone towers, and input from a user, in the users' device to retrieve the users' location.

The users' location retrieved by using the Geolocation API is almost accurate depending upon the type of source used to retrieve the location. For example, the IP address gives a location that can be close to the users' location; while, the GPS is able to give more accurate results.

Identifying the users' location may, at times, compromise the users' privacy. Hence, the location of users is not available unless they approve it. A prompt appears that asks the users if they would like to share their current location and also specifies the reason for collecting this data. In addition, it should specify where the data will be shared.

Geolocation is most beneficial with applications used for mobile devices. This is because while the users are travelling, the Geolocation API keeps updating their locations. In case of desktop applications, the users' location remains constant and can be set only once.

The Geolocation API provides the following methods to determine the users' location:

- `getCurrentPosition()`
- `watchPosition()`

getCurrentPosition()

The `getCurrentPosition()` method is used to retrieve the current geographical location of a user. The location is retrieved as a set of coordinates.

The syntax of the `getCurrentPosition()` method is:

```
getCurrentPosition(CallbackFunction, ErrorHandler, Options);
```

In the preceding syntax:

- `CallbackFunction`: This is a function defined by the developer to retrieve the current location.
- `ErrorHandler`: This is the name of a function that is called when an error occurs while retrieving the location of a user. This is an optional parameter.
- `Options`: This optional parameter specifies a set of options, such as timeout for retrieving the location information, used for retrieving the information about the users' geographical location.

The `getCurrentPosition()` method calls callbackFunction, which takes the `position` object as an argument. This object specifies the current geographic location of a user as a set of geographic coordinates. The `position` object returns two properties, `coords` and `timestamp`, on the successful retrieval of the location. The `timestamp` property returns the date and time when the location was retrieved. The `coords` property returns various attributes, such as `latitude` and `longitude`. These attributes are described in the following table.

Attribute	***Description***
coords.latitude	*Specifies the latitude as a decimal number.*
coords.longitude	*Specifies the longitude as a decimal number.*
coords.accuracy	*Specifies the accuracy of position.*
coords.altitude	*Specifies the altitude in meters above the sea level.*
coords.altitudeAccuracy	*Specifies the accuracy of altitude.*

The Attributes of the coords Property

Before retrieving the users' location by using the `getCurrentPosition()` method, you need to check whether the browser supports the geolocation feature. For this, you can use the `geolocation` property of the `navigator` object. The `navigator` object contains the information about a browser. Consider the following code snippet to check whether the browser supports the geolocation feature:

```
if(navigator.geolocation)
var geo=navigator.geolocation
else
{
alert("Your browser does not support geolocation")
}
```

The preceding code snippet checks whether the geolocation feature is supported by the browser. If it is not supported, a relevant message is displayed.

Consider the following code snippet for retrieving the users' location by using the `getCurrentPosition()` method:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<P ID="button">Click here to know your location coordinates:</P>
<BUTTON onclick="getLocation()">Get Location</Button>
<SCRIPT>
var geo=document.getElementById("button");
function getLocation()
{
if (navigator.geolocation)
{
navigator.geolocation.getCurrentPosition(getPosition);
}
else{geo.innerHTML="Geolocation is not supported by this browser.";}
}
function getPosition(position)
{
geo.innerHTML="Latitude: " + position.coords.latitude +
"<BR>Longitude: " + position.coords.longitude;
}
</SCRIPT>
</BODY>
</HTML>
```

The preceding code snippet creates the **Get Location** button, which, on clicking, calls the

`getLocation()` method. This method first checks whether the geolocation feature is supported by the browser. If this feature is supported, the `getCurrentPosition()` method is called. This method calls the `getPosition()` method that retrieves the users' location in the form of latitude and longitude values.


The innerHTML property is used to set or retrieve the text of an HTML element.


Due to security reasons and firewall restrictions, you may not always get the latitude and longitude coordinates.

watchPosition()

The `watchPosition()` method returns the current location of a user and continuously updates the location while the user is moving. It is mostly used in devices, such as GPS, which inform the users about the current location while they are travelling. The `watchPosition()` method takes the same parameters as the `getCurrentPosition()` method and returns the same object.

Consider the following code snippet for retrieving the users' location by using the `watchPosition()` method:

```
function getLocation()
  {
  if (navigator.geolocation)
    {
    navigator.geolocation.watchPosition(getPosition);
    }
  else{geo.innerHTML="Geolocation is not supported by this
browser.";}
  }
function getPosition(position)
  {
  geo.innerHTML="Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
  }
```

The preceding code snippet continuously retrieves the user's location by using the `watchPosition()` method while the user is travelling.

You can also stop tracking the users' location by using the `clearWatch()` method. This method stops the call to the `watchPosition()` method.

Handling Errors

At times, the user may not provide the permission to access his location. This may raise an error in your website. The error may also occur when a user checks the current location on a mobile device and the device goes out of coverage area or the network connection is timed out. As a website developer, you need to ensure that your website is able to handle errors effectively. For this, you need to consider all possible causes of errors that can occur in an application and can be dealt with effective error-handling technique.

To handle errors in a geolocation-enabled website, you can use the `getCurrentPosition()` method. The second parameter of this method is an error handler function. This function can handle the following error codes:

- `PERMISSION_DENIED`: Specifies that the user has declined the request to share the location.
- `POSITION_UNAVAILABLE`: Specifies that the users' current location cannot be retrieved.
- `TIMEOUT`: Specifies that the time given to retrieve the users' location has exceeded the maximum limit.
- `UNKNOWN_ERROR`: Specifies that an unknown or undefined error has occurred.

The error handler function accepts the `PositionError` object as an argument. This object has two properties, `code` and `message`. The code specifies the error code, and the message specifies the message that is to be displayed when an error occurs.

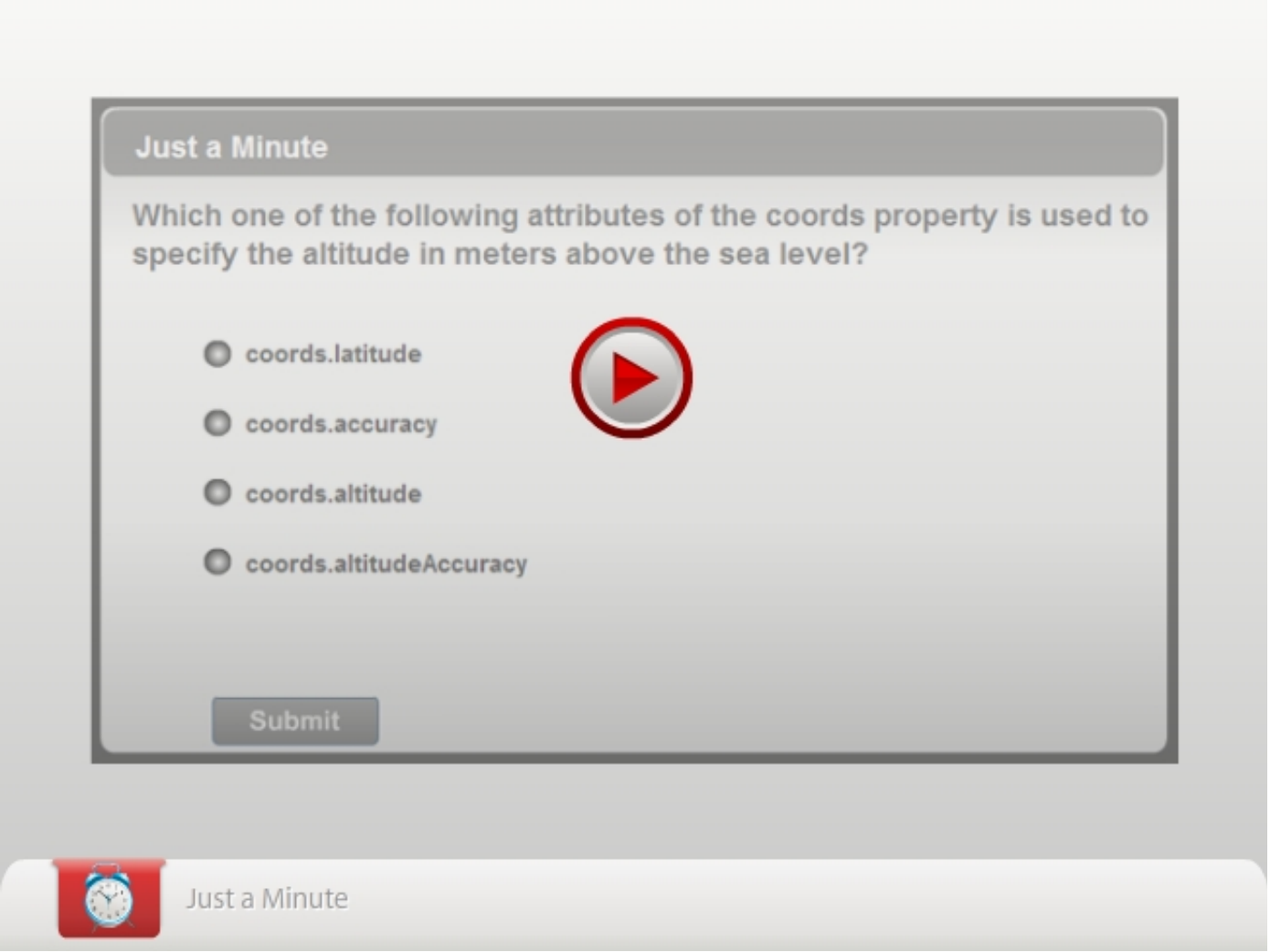
Consider the following code snippet for handling errors while retrieving the users' location:

```
function showError(PositionError)
{
switch(PositionError.code)
   {
   case PositionError.PERMISSION_DENIED:
     x.innerHTML="User denied the request for tracking the location"
     break;
   case PositionError.POSITION_UNAVAILABLE:
     x.innerHTML="User's location is not available"
     break;
   case PositionError.TIMEOUT:
     x.innerHTML="The request to retrieve user's location is timed
out"
     break;
   case PositionError.UNKNOWN_ERROR:
     x.innerHTML="An unknown error occurred"
     break;
   }


}
```

The preceding code snippet creates the function, `showError()`, which is called when an error occurs while retrieving the users' location. This function accepts an error code which is defined in the `switch`

construct. For example, if the user denies the access to his location, the error, `User denied the request for tracking the location`, is displayed on a Web page.



Implementing Offline Support

At times, you must have noticed that while you are surfing a website, a message, such as 'Browser could not open the Web page', is displayed. This is because the connection to the Internet is lost. In such a case, you do not have access to the website that you were surfing. However, if you want to have some level of access to the website even if you are not connected to the Internet, you can use the offline support feature. For example, the executives of ShopHere often need to visit different cities to advertise their products. They need to access the organization's website during their promotional tours. However, the website is not accessible every time, especially in the remote areas where the network is not available. In such a case, the offline support feature of HTML can be used to ensure that the website is accessible to the executives even without the Internet connection. In addition, using the offline support feature avoids the normal network requests needed to load a website.

Before the offline support feature of HTML, the offline storage-enabled websites were created using cookies and plugins. A cookie is a small piece of data, which is sent from a website and stored in a user's browser while the user browses the website. When the user revisits the same website, the data stored in the cookie is retrieved. Cookies are not purely used for offline storage as they store a limited amount of data. In addition, cookies slow down the network activity because they are transferred to and from the server.

These limitations are overcome with the help of the offline support feature of HTML. The offline support feature provides the following benefits:

- Ensures that the website is available even when the user is not connected to the network.
- Reduces network load on the server.

You can make your website work offline by using the following ways:

- Implementing client-side storage
- Implementing application cache

Implementing Client-side Storage

The client-side storage refers to the process of storing data locally within the user's browser. This is also known as Web storage. The data stored by using client-side storage is retrieved only when it is requested, and not with every server request. Moreover, you can store a large amount of data by implementing the client-side storage, where the data is stored in the form of key/value pairs.

The client-side storage can be implemented by using the following objects:

- `localStorage`
- `sessionStorage`

localStorage

The `localStorage` object allows you to store data without any expiration date. This implies that the data stored by using the `localStorage` object is not deleted after the browser is closed, and it will be available when the browser is reopened. The `localStorage` object stores the data only in the form of string. Using this object, the cached data is accessible across all browser windows.

Consider the following code snippet for storing the users' information by using `localStorage`:

```
<DIV ID="str"></DIV>
<SCRIPT>
if(typeof(Storage)!=="undefined")
  {
  localStorage.name="John";
  document.getElementById("str").innerHTML="Name: " +
localStorage.name;
  }
else
  {
  document.getElementById("str").innerHTML="Your browser does not
support local storage";
  }
</SCRIPT>
```

In the preceding code snippet, the `typeof()` method first checks whether the browser supports the Web storage or not. If it does, the `localStorage` object stores the user's name locally in the browser by using the key, `name`, which is assigned the value, `John`.

Later, the information stored in the `localStorage` object is retrieved by using the key and is assigned to the `innerHTML` property of the `<DIV>` tag that has `ID`, `str`.

The text, `Your browser does not support local storage`, is displayed inside the `<DIV>` tag if the browser does not support the Web storage.

sessionStorage

The `sessionStorage` object is used to store the data for only one session. This implies that the data is deleted once the user closes the browser. The data stored by using `sessionStorage` is confined to the browser for which it was created. Consider an example where you need to count the number of times a user has clicked a button in the current session. For this, you can use the `sessionStorage` object, as shown in the following code snippet:

```
<HEAD>
<SCRIPT>
function clickCounter()
{
if(typeof(Storage)!=="undefined")
  {
  if (sessionStorage.clickcount)
    {
    sessionStorage.clickcount=Number(sessionStorage.clickcount)+1;
    }
  else
    {
    sessionStorage.clickcount=1;
    }
  document.getElementById("btn").innerHTML="You have clicked it" +

sessionStorage.clickcount + " time(s) in this session.";
  }
else
  {
  document.getElementById("btn").innerHTML="Your browser does not
support Web storage";
  }
}
</SCRIPT>
</HEAD>
```

```
<BODY>
<p><BUTTON onclick="clickCounter()" type="button">Click Here</BUTTON>
</P>
<DIV ID="btn"></DIV>
<P>Click the button to see the increase in counter.</P>
<P>Close the browser and try again, the counter will be reset.</P>
</BODY>
```

The preceding code snippet creates the `clickCounter()` function that is called when the button is clicked.

Inside the function body, the `typeof()` method first checks whether the browser supports the Web storage or not. If it does, the `clickCount` key is used with the `sessionStorage` object to store the count or number of times the user has clicked the button during the current session. The value of the key, `clickCount`, is increased by one every time the user clicks the button. Since the value assigned to the key of the `sessionStorage` object is always a string, it is converted into a number before incrementing by one, by using the `Number()` method.

The most recent value stored inside the `sessionStorage` object is retrieved by using the key, `clickCount`, and is assigned to the `innerHTML` property of the `<DIV>` tag that has `ID`, `btn`, which is displayed before the user.

Implementing Application Cache

While browsing a website, you must have faced a situation when the network connection is lost and you click the Back button in the browser to view the previous page. However, you are not able to view that page as you are not connected to the Internet, and the browser did not cache the page properly. To view the previous page in such a situation, you need to reconnect to the Internet. To address such issues, while developing a website, you can specify the files the browser should cache so that even if you refresh the page when you are offline, you are able to view the page. This process of caching a website is known as application cache.

The application cache provides the following advantages:

- **Offline browsing**: Specifies that a website can be viewed even if the user is not connected to the Internet.
- **Speed**: Specifies that if the user requests the Web page, which is already there in the cache, it is retrieved from the cache and not from the server. Therefore, the loading of the Web page is faster as the network is not accessed as the connection to the server is not needed.
- **Reduced server load**: Specifies that the Web page, if cached, will always be made available from the cache unless the browser detects that the cache manifest has been updated on the server or the user has cleared the browser cache. Then, the browser downloads a new version of the manifest and the resources listed in the manifest. Therefore, the number of requests sent to the server are less; thereby, reducing the load on the server.

To implement application cache, you need to create a text file called manifest. This file contains a list of resources that needs to be cached for use when there is no network connectivity. The manifest file also contains the list of files or pages that should never be cached. You need to save the manifest file with the .appcache extension. The manifest file is divided into the following sections:

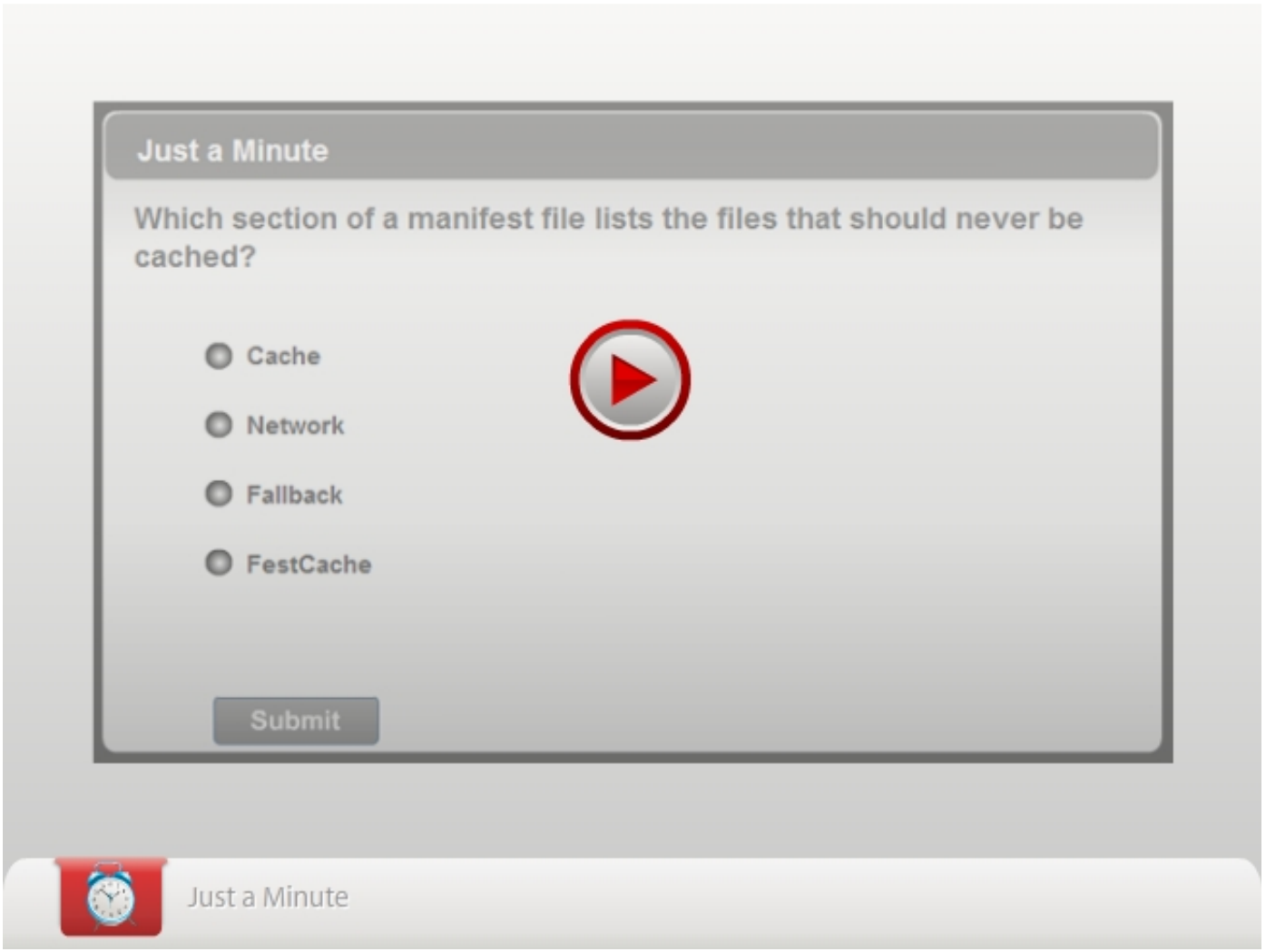
- **Cache**: Lists the files that need to be cached after they are downloaded for the first time.
- **Network**: Lists the files that should never be cached.
- **Fallback**: Specifies the task to be performed when a user tries to fetch the uncached files.

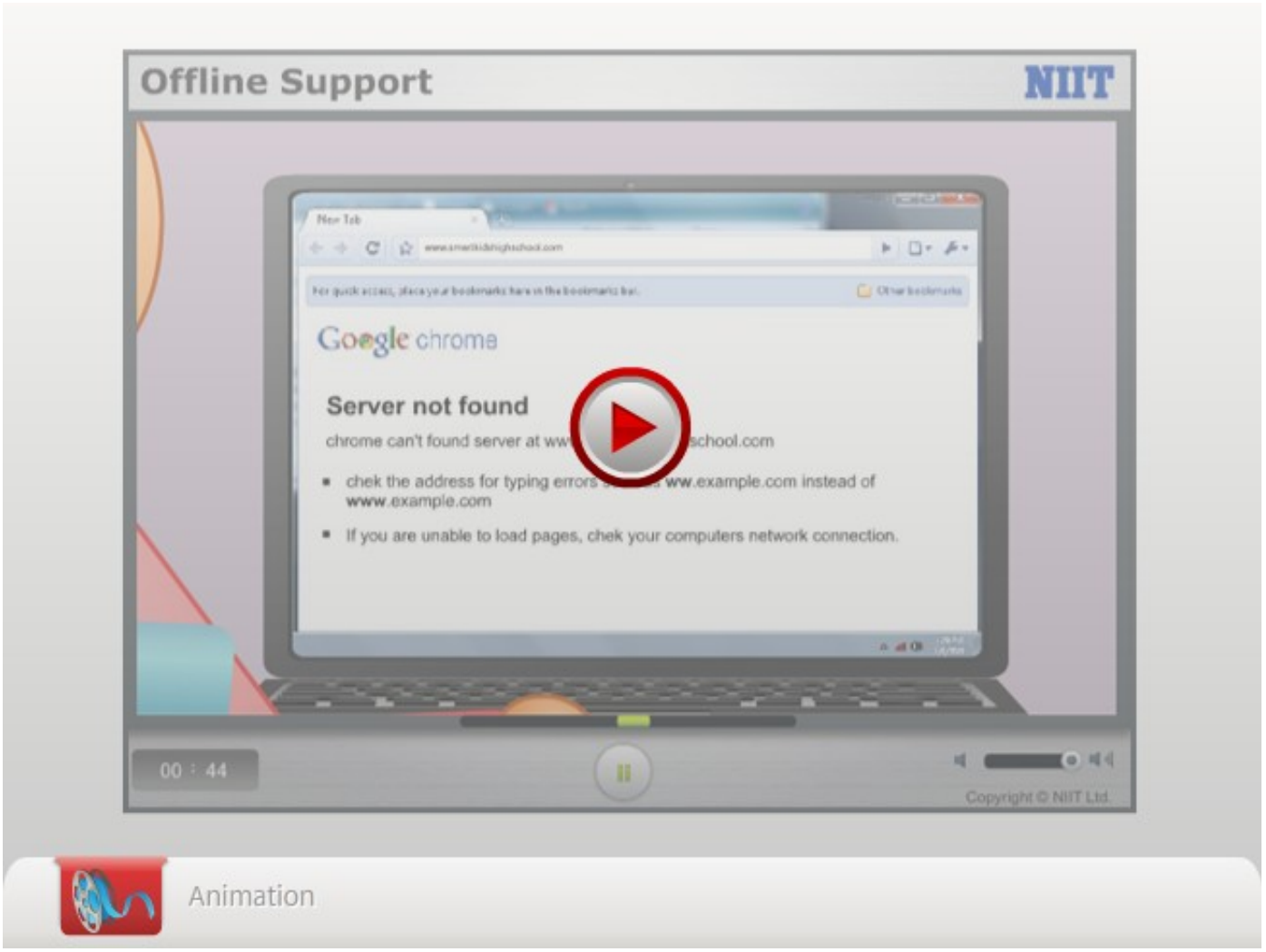
To refer to a manifest file in an HTML document, you need to add the manifest attribute in the opening `<HTML>` tag, as shown in the following code snippet:

```
<HTML manifest="HotelFacilities.appcache">
```

The preceding code snippet enables the application cache for the `HotelFacilities.appcache` manifest file.

Once you have cached an application on the local machine, the browser shows the cached file even if it has been changed or updated on the server. Therefore, to ensure that the browser updates the cache, you need to modify the manifest file.





Activity 8.1: Implementing Offline Support

Summary

In this chapter, you learned that:

- The Geolocation API allows a website to retrieve the current geographical location of a user.
- The Geolocation API provides the following methods to determine the users' location:
  - `getCurrentPosition()`
  - `watchPosition()`
- To handle errors in a geolocation-enabled website, you can use the `getCurrentPosition()` method. The second parameter of this method is an error handler function.
- The offline support feature provides the following benefits:
  - Ensures that the website is available even when the user is not connected to the network
  - Reduces network load on the server
- The client-side storage refers to the process of storing data locally within the user's browser.
- The client-side storage can be implemented by using the following objects:

- `localStorage`
- `sessionStorage`

- The `localStorage` object allows you to store data without any expiration date.
- The `sessionStorage` object is used to store the data for only one session. This implies that the data is deleted once the user closes the browser.
- To implement application cache, you need to create a text file called manifest.
- You need to save the manifest file with the .appcache extension.

Reference Reading

Implementing Geolocation

Reference Reading: Books	***Reference Reading: URLs***
The Definitive Guide to HTML5 By Adam Freeman	*http://diveintohtml5.info/geolocation.html* *http://www.w3schools.com/html/html5_geolocation.asp*

Implementing Offline Support

Reference Reading: Books	***Reference Reading: URLs***
The Definitive Guide to HTML5 By Adam Freeman	*http://www.pageresource.com/html5/local-session-storage-api/*