Session 6

Services and Communication in AngularJS

Session Objectives

- ✓ Define and explain Representational State Transfer (REST) Application Programming Interface (API)
- ✓ Describe RESTful services in AngularJS
- ✓ Explain client-server communication
- ✓ List the steps to access server using various server resources
- ✓ Explain error handling in AngularJS client-server communication

Introduction to REST API

What is REST?

- Stands for REST, short for REpresentational State Transfer
- Is a stateless architecture for networked applications and a lightweight alternative to Web services
- Implements a cacheable client-server protocol, Hypertext Transfer Protocol (HTTP), for communication through CRUD operations
- Simplifies network communication by avoiding complex approaches
- Is optimized for the World Wide Web, making it more popular than HTTP
- Is not considered a standard due to no World Wide Web Consortium (W3C) recommendation

Introduction to REST API

In simple terms, REST is a set of principles that define how Web standards, such as HTTP and URIs, are supposed to be used.

Key principles for REST architecture

- Communicate statelessly
- Give every resource an ID
- Link things together
- Use standard methods
- Resources with multiple representations
- Make code available on demand

REST Principles

Principles of REST architecture are formally laid out as:

Client-server Communication	Works on the principle of separation of concerns for making each component independent.
Statelessness	Ensures that no client information remains on the server after communication.
Cacheability	Facilitates caching a server response on the client for using it later while responding to a similar request in future.
Uniform Interface	Simplifies interaction with different services and components and ensures changes happen without affecting the whole interface or interaction.
Layered System	Allows having an intermediary between the client-server communication without letting the client to know about it.
Code on Demand	Allows extending the client's functionality on a temporary basis.

REST versus SOAP

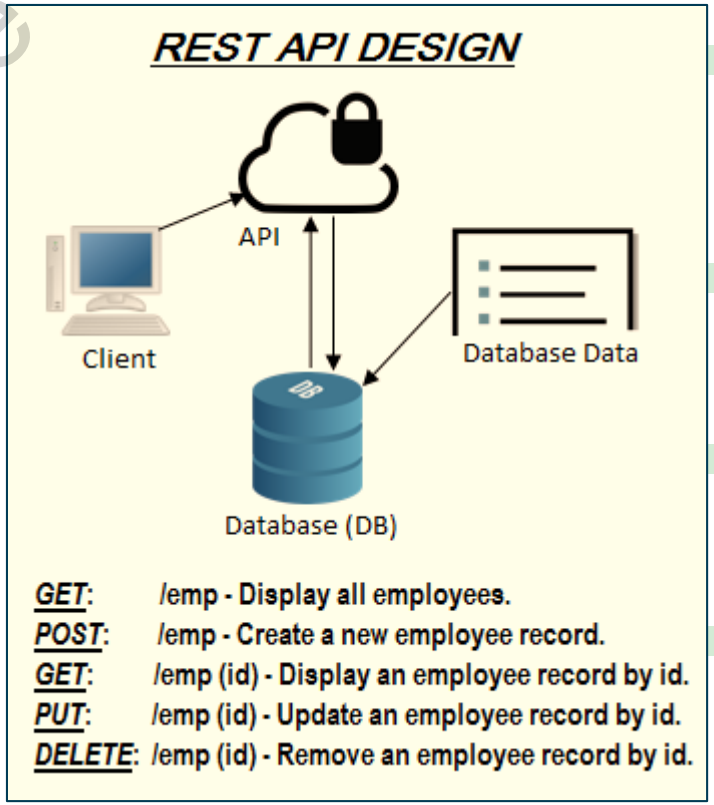
REST and SOAP have several differences:

Point of Distinction	SOAP	REST
Degree of Coupling	Is tightly coupled to the server, just as a custom desktop application.	Is loosely coupled, just as a browser.
Orientation	Is object oriented.	Is resource oriented.
Size	Is heavy due to additional XML.	Is lightweight.
State	Is stateful.	Is stateless.
Standard	Is standard specific.	Is not standard specific.
Speed	Is slower due to strict specifications and requirements for more bandwidth and resources.	Is faster, as there are no strict specifications and that it consumes less resources and bandwidth.
Protocol Dependency	Is independent of transport protocol in use. It can use any transport protocol.	Is dependent on HTTP, although is not coupled to it.
Communication	Uses only eXtensible Markup Language (XML).	Uses self-descriptive messages that control interaction and are represented via media types such as XML, plain text, and JavaScript Object Notation (JSON).

RESTful Web Services

A RESTful Web service:

- Uses HTTP methods.
- Defines a URI that provides HTTP methods and a format for holding a resource.



Resources in RESTful Web Services

A resource:

- Is any content in a Web service with a unique URI, such as document, image, and collection of users.
- Is of a particular type and holds data.
- Possesses a set of HTTP operation methods.
- Has a state.
- Is analogous to an object but has only a few HTTP methods unlike the latter.
- Is accessible by a REST client.
- Is representable in a variety of media types such as XML and JSON.

Resources in RESTful Web Services

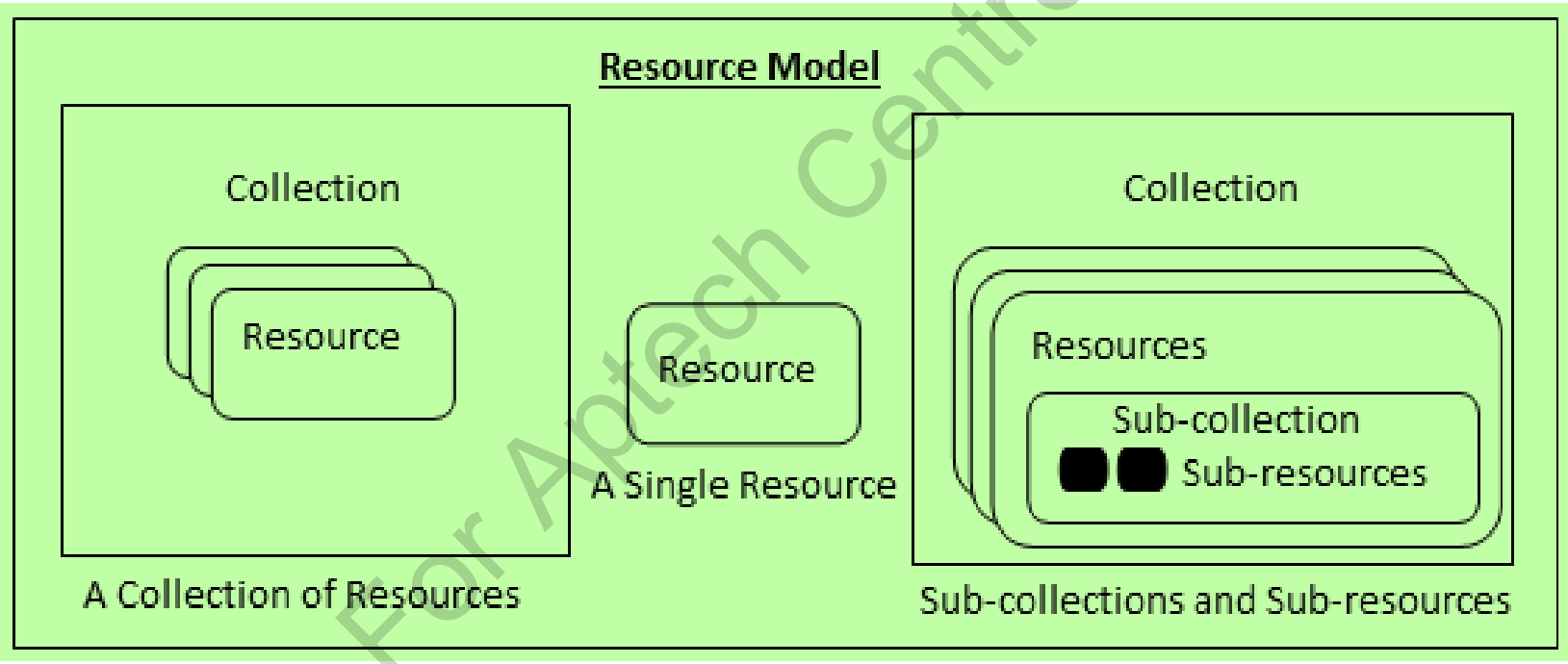
Following are the commonly used HTTP methods in the REST architecture:

POST (C)	Creates a new resource or updates an existing one
GET (R)	Offers a read-only access to a resource (cacheable)
PUT (U)	Creates a new resource when the client is aware of its URI
DELETE (D)	Removes a resource

Resources in RESTful Web Services

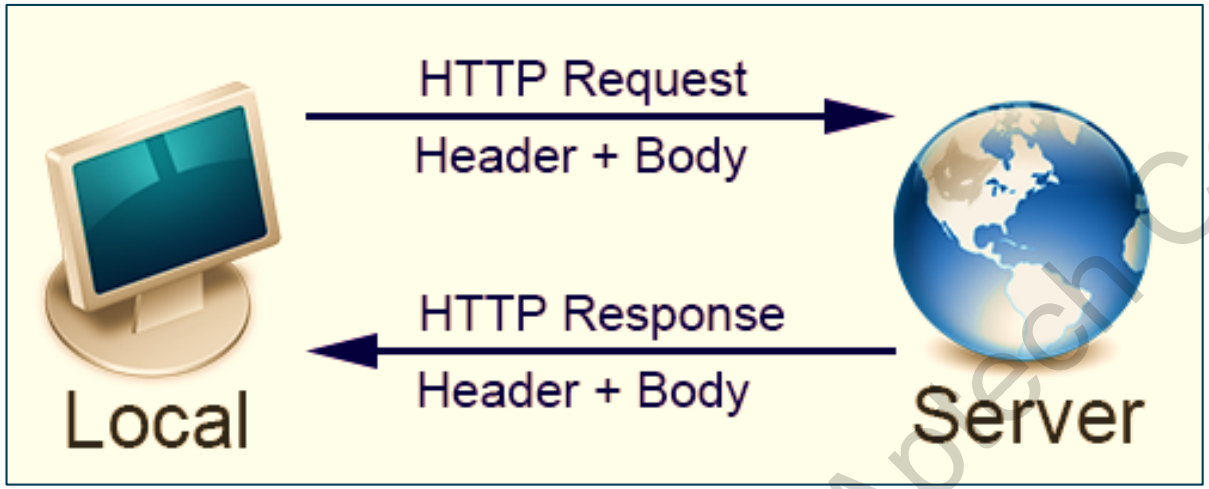
A collection is an unordered group of resources and is a resource in itself.

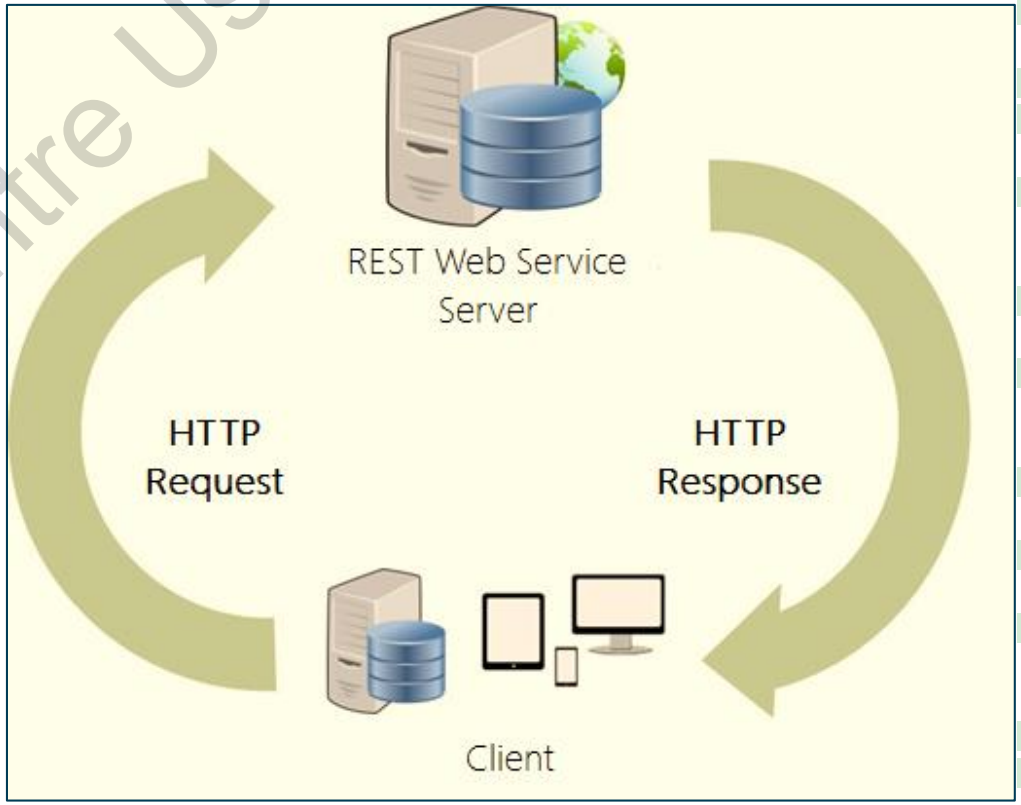
All resources inside are of a single type. It can be present globally.



HTTP Messages

A RESTful Web service uses HTTP for exchanging data between a client and a server.





HTTP Messages - Request

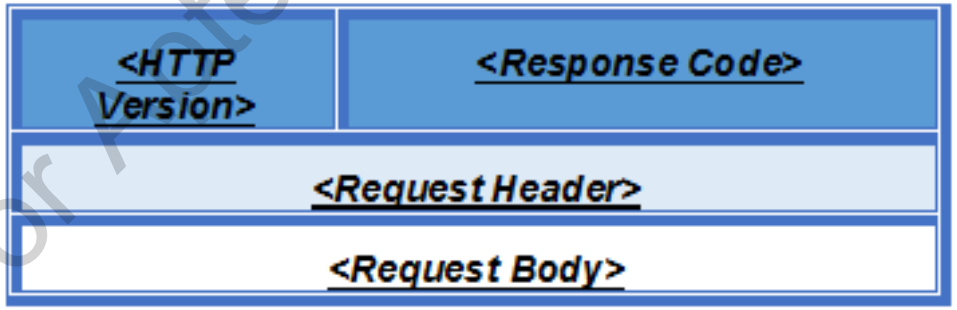
An HTTP Request consists of five parts:

- **Verb**: Indicates an HTTP method.
- **URI**: Represents the identifier for recognizing the desired resource on the target server.
- **HTTP Version**: Signifies the HTTP version, such as HTTP v1.1.
- **Request Header**: Contains key-value pairs to show message's metadata. For instance, it includes the cache settings, type of browser, and message format.
- **Request Body**: Holds the message in the desired format.

HTTP Messages - Response

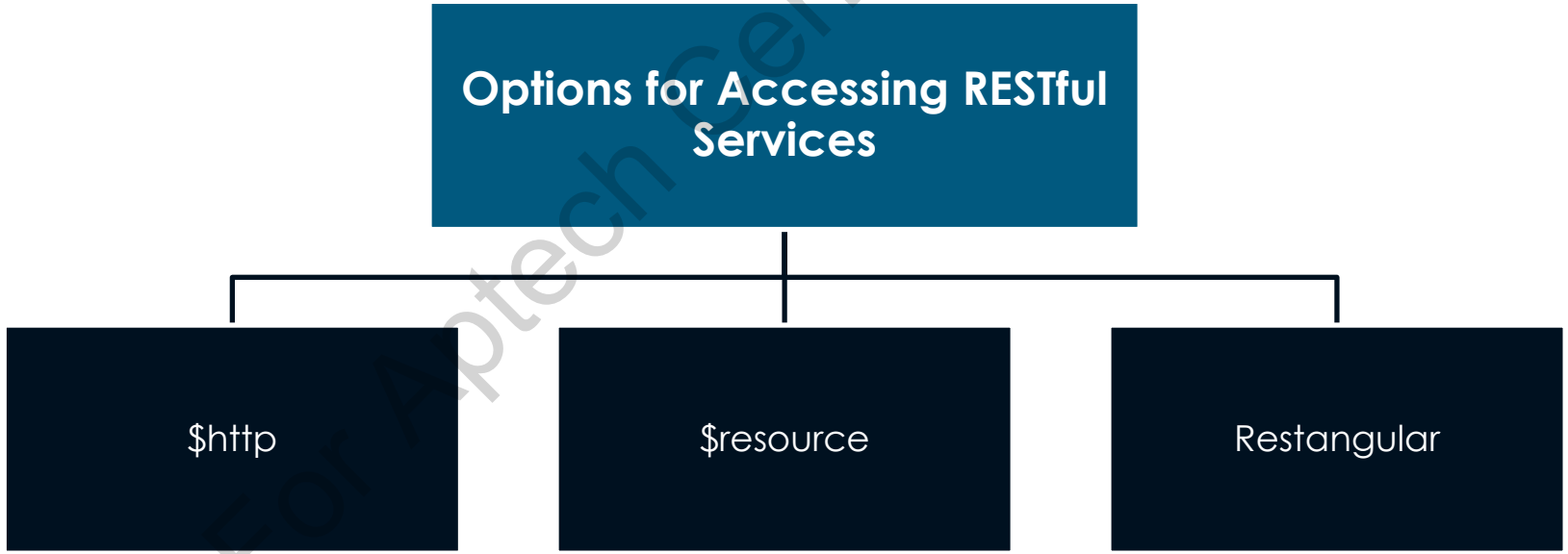
An HTTP Response consists of four parts:

- **HTTP Version**: Signifies HTTP version, such as HTTP v1.1.
- **Response Code**: Is in the form of three digits denoting status for resource requested. For instance, code 404 means resource is not found on the server.
- **Request Header**: Contains key-value pairs to show message's metadata. For instance, it includes the length of content, type of server, response date, and type of content.
- **Request Body**: Holds the message in the desired format.



RESTful Web Services - Approaches

In AngularJS, three options exist for accessing services from a RESTful API:



$http Service in AngularJS

- Sends an Asynchronous JavaScript and XML (AJAX) request to a remote Web service on a server.
- Executes the request without waiting for the server's reply.
- Fetches data from the Web service in the JSON format.
- Implements HTTP methods.

Syntax of $http

The service is a function with a single parameter, which is the request configuration object.

Syntax:

```
$http({ method: 'GET', url: '/user1' })
  success(function (data, status, headers, config) {
    // ...
  })
  error(function (data, status, headers, config) {
    // ...
  });
```

where,

- **`success and error`**: Are callback methods of the `promise` object that `get()` returns.
- **`data`**: Is the server's response as a string or an object.
- **`status`**: Is the HTTP status code.
- **`headers`**: Refers to a function that fetches header information.
- **`config`**: Refers to the configuration object that created the request.

Promise Object

The promise object:

- Denotes the final outcome of an action.
- Indicates what to do when an action succeeds or fails.
- Refers to two callback methods to handle the server's response.
- Triggers `success()` when the response is successfully available.
- Triggers `error()` when the response contains an error status code and returns a single object with properties such as data and statusText to convey the reasons of failure.

Following example shows how to make a request asynchronously:

```
$var promise = $http.get("/api/emp/lastname");
promise.success(function(name) {
   console.log("Request succeeded. " + "Name is: " + name);
});
promise.error(function(response, status) {
   console.log("Request failed: " + response + " with status code " + status);
});
```

Using $http in AngularJS - Example

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
 <title>HTTP Get Demo</title>
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
</script>
</head>
<body ng-app="myApp" ng-controller="FormulaOneCtrl">
  {{text}}
  <script>
   var app = angular.module('myApp', []);
   app.controller('FormulaOneCtrl', function($scope, $http) {
     var promise = $http.get("http://ergast.com/api/f1/2019/circuits.json");
     promise.then(function() {
           $scope.text ="Request succeeded.";
     }),
     (function(response, status) {
        $scope.text ="Request failed.";
     });
   });
  </script>
</body>
</html>
```

$resource Service in AngularJS

The `$resource` service:

- Fetches data to manipulate and return it.
- Implements CRUD methods.
- Is built on the top of `$http`.
- Facilitates communication with the backend.

Following example shows how to use the `$resource` service for implementing the GET method in AngularJS:

```
var Tasks = $resource('/task/:taskId', { taskId: '@id' });
var task = Tasks.get({ taskId: 1023 }, function () {
  task.abc = true;
  task.$save();
});
```

Restangular Tool

- Is a third-party, open-source framework.
- Minimizes coding for HTTP communication through CRUD operations.
- Is ideal for Web applications using data from a RESTful API.
- Supports all HTTP methods, facilitates creating custom methods, and removes the necessity to mention the URL.

Server Communication

For communicating with a Web server, AngularJS:

- Offers low-level mechanism and built-in wrappers for interacting with RESTful services.
- Supports AJAX due to which there is no need to refresh the whole page for updating a part of it.
- Fetches layout and data separately.

Accessing Server Resources with $http

For accessing server resources, the `$http` service:

- Implements the standardized way of handling asynchronous calls through promise.
- Triggers generic AJAX calls that can interact with the RESTful or Non-RESTful API on the server.
- Offers two ways, JSONP or `XMLHttpRequest` object of the browser.
- Provides following functionalities:
  - GET
  - POST
  - HEAD
  - PUT
  - DELETE
  - JSONP

Accessing Server Resources with $http

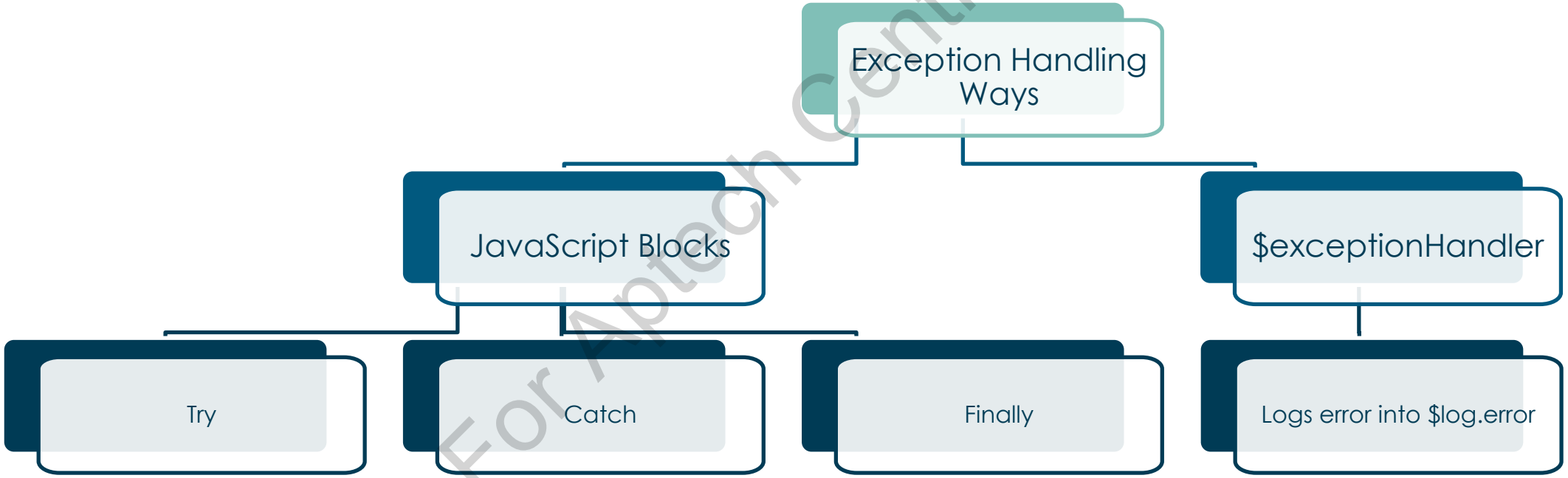
Four steps for accessing server resources via `$http`:

1. Define a custom service for injecting the $http service directly into it.
2. Use the $http service with an appropriate request method.
3. Pass the config object as the parameter to return a promise object.
4. Fetch the response once the request has been processed.

Handling Errors in Client-Server Communication

Error handling is essential to find legible reasons of a failure that occurs.

Ways in which AngularJS facilitates exception handling:



Communication Using $resource

The main AngularJS script files does not include the `$resource` service. Thus, it is essential to download its file and add it to the index.html file.

Following example shows how to include the `$resource` service in index.html:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular-
resource.min.js"></script>
```

Next, declare a dependency on `$resource` so that a controller can use it. Finally, call the $resource() function with the targeted REST endpoint address.

The function returns `$resource` object, which communicates with a REST backend server. By default, the `$resource` object possesses five methods, which are `get()`, `save()`, `query()`, `delete()`, and `remove()`.

```
angular.module('resourceApp.services').factory('Domain', function($resource)
{
  return $resource('/api/domains/:empid');
});
```

Communication Using $resource


```
<!DOCTYPE html>
<html lang="en" ng-app="resourceApp">
<head>
<meta charset="UTF-8">
<title>Resource Demo</title>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
</script>
<script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular-resource.min.js">
</script>
</head>
<body ng-controller="MainCtrl">
Customer Names:
    <select ng-options="Customer.Name for Customer in Customers" ng-model =
"CurrentCustomer"></select>
        <br>
        <br>
        New Name: <input type = "text" name = "CustName" ng-model="CustName"><br>
         <input type = "submit" value = "Submit" ng-click="Add()" >
         {{text}}
</body>
<script>
var app = angular.module('resourceApp', ['ngResource']);
```

Communication Using $resource


```
app.controller('MainCtrl', function($scope, $resource) {
    var Customers = $resource('Customer.json').query(function(){
        $scope.Customers = Customers;
        $scope.CurrentCustomer = Customers[0];
           $scope.Add = function () {
                 $scope.CurrentCustomer.Name = $scope.CustName;
              Customers.save($scope.CurrentCustomer, function() {
    $scope.text="Added";
  });
  };
  });
});
```

Summary

- REST is a stateless architecture that aids in making networking applications by implementing HTTP.
- RESTful applications work on six principles or constraints namely client-server communication, statelessness, cacheability, uniform interface, layered system, and code on demand.
- An HTTP response returns a status code that indicates the status of the requested resource.
- The $http service communicates asynchronously with a remote Web service and returns a Promise object.
- The $resource service is easier than $http to access a RESTful Web service and a RESTful backend.
- The $exceptionHandler service catches an uncaught exception and passes it to $log.error for displaying the exception in the browser's console.