WEB TECHNOLOGIES

Express JS –

Introductions to Web Services and REST APIs

Prof. Vinay Joshi and Dr. Sarasvathi V
Department of Computer Science and Engineering.

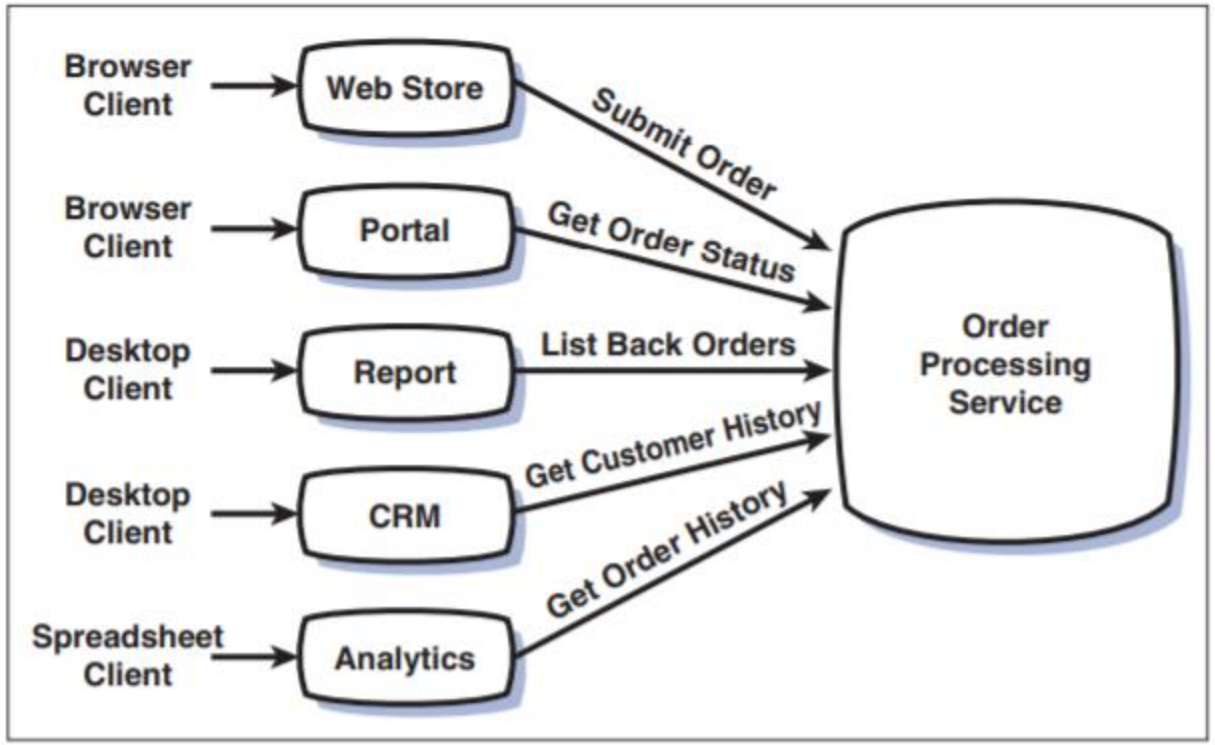
Acknowledgement
The slides are created from various internet resources with valuable contributions from multiple professors

Introductions to Web Services and REST APIs

- A Web service is a service that lives on the Web.
- The ***Web*** is a huge information space filled with interconnected resources.
- A ***service*** is an application that can be consumed by software.
- A **service** is an application that exposes its functionality through an application programming interface (*API*)
- An interface hides the complexities of the internal system.
- One service can support many applications.

Express JS

Introductions to Web Services



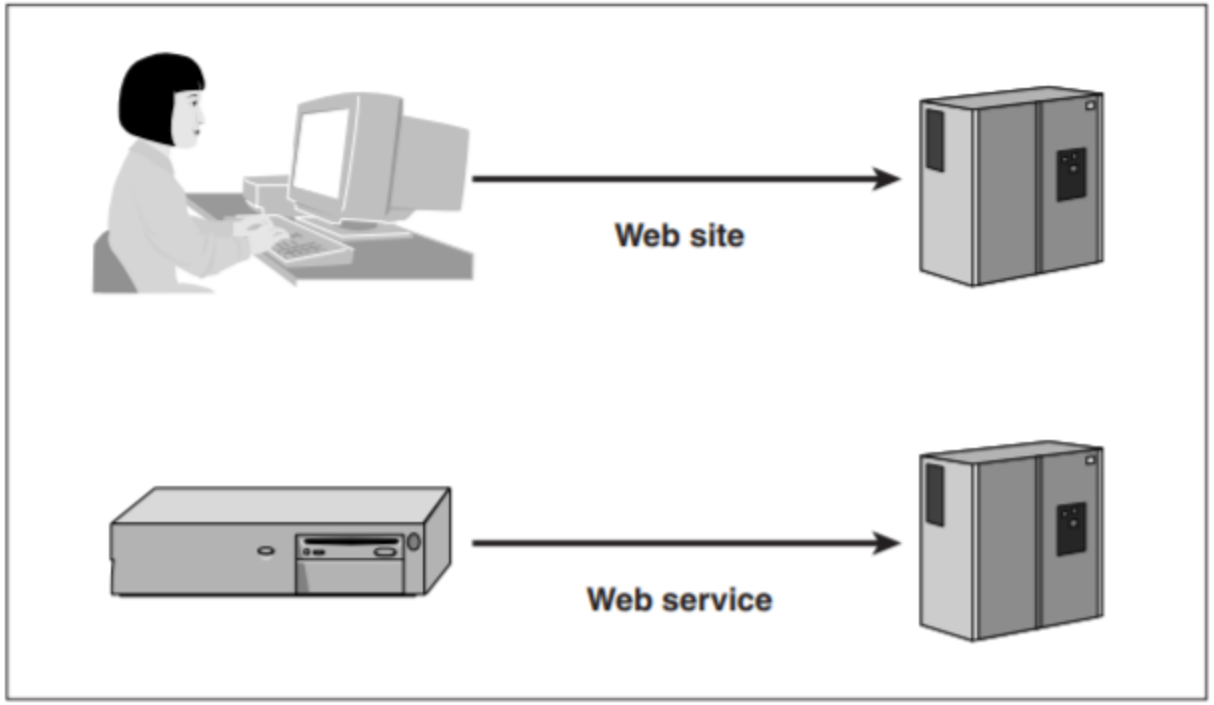
- A service can be shared by many different applications.
- Many different users can share a single service.

Express JS

Introductions to Web Services

•A Web site is designed to be accessed by humans.

•A Web service is designed to be accessed by applications.



Express JS

Definition of Web Services

- A programmable application component that is accessible via standard Internet protocols.
- W3C definition: *Software system designed to support interoperable machine-to-machine interaction over a network.*
- A Web service is an application with a Web API.

Definition of Web Services

The key things to understand about Web services:

- Designed for machine-to-machine (or application-to-application) interaction.
- Should be interoperable - Not platform dependent.
- Should allow communication over a network.

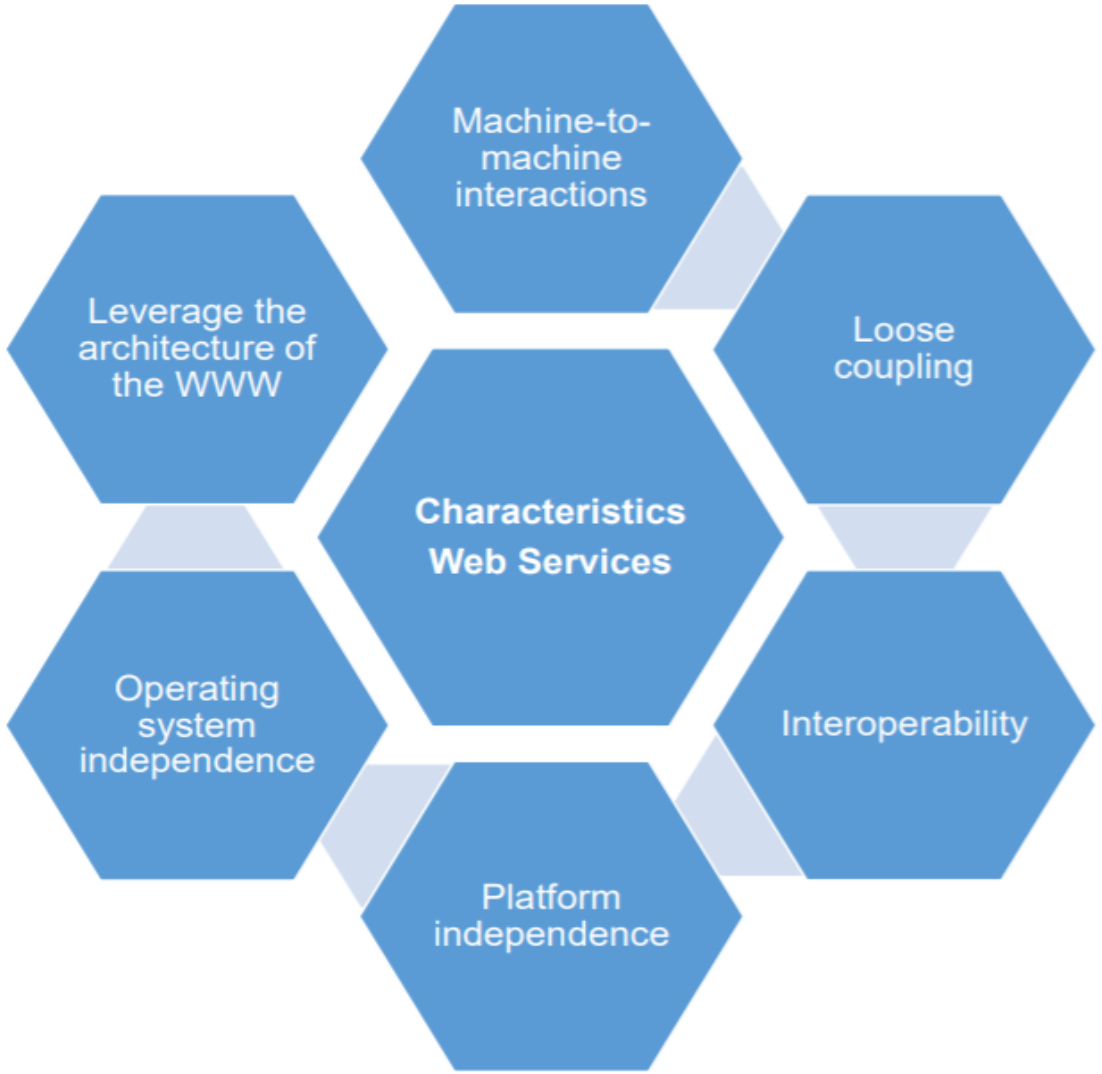
Express JS

Why Web Services?

- Web services help you integrate applications.
- Web services support heterogeneous interoperability.
- Web services are inexpensive.
- Web services are flexible and adaptable.

Express JS

Characteristics of Web Services



Express JS

REST APIs(Representational State Transfer)

- REST is an architecture for developing web services.
- builds upon existing systems and features of the internet's Hypertext Transfer Protocol (HTTP) in order to achieve its objectives.
- Guided by REST constraints (design rules).
- Based on Resource Oriented Architecture.
- A network of web pages where the client progresses through an application by selecting links.
- Requests/responses relate to representations of states of a resource.
- When client traverses link, accesses new resource (i.e., transfers state).
- Uses simple HTTP protocol.

Express JS

REST APIs

- REST is about resources and how to represent resources in different ways.
- REST is about client-server communication.
- REST is about how to manipulate resources.
- REST offers a simple, interoperable and flexible way of writing web services that can be very different from other techniques.

Express JS

REST APIs

- Client **request**s a specific **resource** from the server.
- The server **respond**s to that request by delivering the requested resource.
- Server does not have any information about any client.
- So, there is no difference between the two requests of the same client.
- A model which the representations of the resources are transferred between the client and the server.
- The Web as we know is already in this form!

Resources

- Resources are consistent mappings from an identifier [such as a URL path] to some set of views on server-side state.
- Every resource must be uniquely addressable via a URI.
- “If one view doesn’t suit your needs, then feel free to create a different resource that provides a better view. ”
- “These views need not have anything to do with how the information is stored on the server. They just need to be understandable (and actionable) by the recipient.” *Roy T. Fielding*

PES UNIVERSITY

RESTful Design Specifications (Constraints)

Client-Server
- Separation of concerns - user interface vs data storage
- Client and server are independent from eachother

Uniform Interface
- All resources are accessed with a generic interface (HTTP-based) which remains same for all clients.

Stateless
- Each request from client to server must contain all of the information
- No client session data or any context stored on the server

Layered System
- Allows an architecture to be composed of hierarchical layers
- Each component cannot “see” beyond the immediate layer.

Cacheable
- Specify data as cacheable or non cacheable
- HTTP responses must be cacheable by the clients

Code On-Demand
- REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts.

HTTP Methods

- GET – *safe, idempotent, cacheable*
- PUT – *idempotent*
- POST
- DELETE – *idempotent*
- HEAD
- OPTIONS

Express JS

REST APIs

Table 5-1. *CRUD Mapping for Collections*

Operation	HTTP Method	Resource	Example	Remarks
Read – List	GET	Collection	GET /customers	Lists objects (additional query string can be used to filter)
Read	GET	Object	GET /customers/1234	Returns a single object (query string may be used to filter fields)
Create	POST	Collection	POST /customers	Creates an object, and the object is supplied in the body.
Update	PUT	Object	PUT /customers/1234	Replaces the object with the object supplied in the body.
Update	PATCH	Object	PATCH /customers/1234	Modifies some attributes of the object, specification in the body.
Delete	DELETE	Object	DELETE /customers/1234	Deletes the object

RESTful Design Considerations : Steps for designing RESTful Web Service

- Identifying resources the service will expose over the network.
- Designing the URI Templates – map URIs to resources
- Applying the Uniform HTTP Interface – options available on each resource for different user groups.
- Security Considerations – Authentication and authorization

RESTful Design Considerations : Steps for designing RESTful Web Service

- Designing the Resource Representations – XML/JSON.
- Supporting alternate Representations – XML or JSON based on filters
- Providing Resource Metadata – Ability to discover resources and options

RESTful Design Considerations : RESTful Service Implementation Considerations

- Parse the incoming request to
  - Use URI to identify the resource.
  - Identify URI variables (and map them to resource variables)
  - HTTP method used in the request (and whether it's allowed for the resource).
  - Read the resource representation

- Authenticate and authorize the user.

RESTful Design Considerations : RESTful Service Implementation Considerations

- Use all of this information to perform the underlying service logic.

- Generate an appropriate HTTP response, including
  - Proper status code
  - Description
  - Outgoing resource representation in the response entity body (if any)

Express JS

REST APIs - Requests & Responses

REQUEST

GET /news/ HTTP/1.1

Host: example.org

Accept-Encoding: compress,

gzip

User-Agent: Python- httplib2

Here is a **GET** request to «http://example.org/news/»

Method = **GET**

REST APIs - Requests & Responses

RESPONSE

HTTP/1.1 200 Ok

Date: Thu, 07 Aug 2008 15:06:24 GMT

Server: Apache

ETag: "85a1b765e8c01dbf872651d7a5"

Content-Type: text/html

Cache-Control: max-age=3600

<!DOCTYPE HTML>

Express JS

REST APIs - Requests & Responses

- The request is to a resource identified by a **URI**
- (**URI** = **U**nified **R**esource **I**dentifier).
- In this case, the resource is «http://example.org/news/»
- Resources, or addressability is very important.
- Every resource is URL-addressable.
- To change system state, simply change a resource.

THANK YOU

Vinay Joshi and Dr.Sarasvathi V

Department of Computer Science and Engineering

vinayj@pes.edu

sarsvathiv@pes.edu

Acknowledgement

The slides are created from various internet resources with valuable contributions from multiple professors