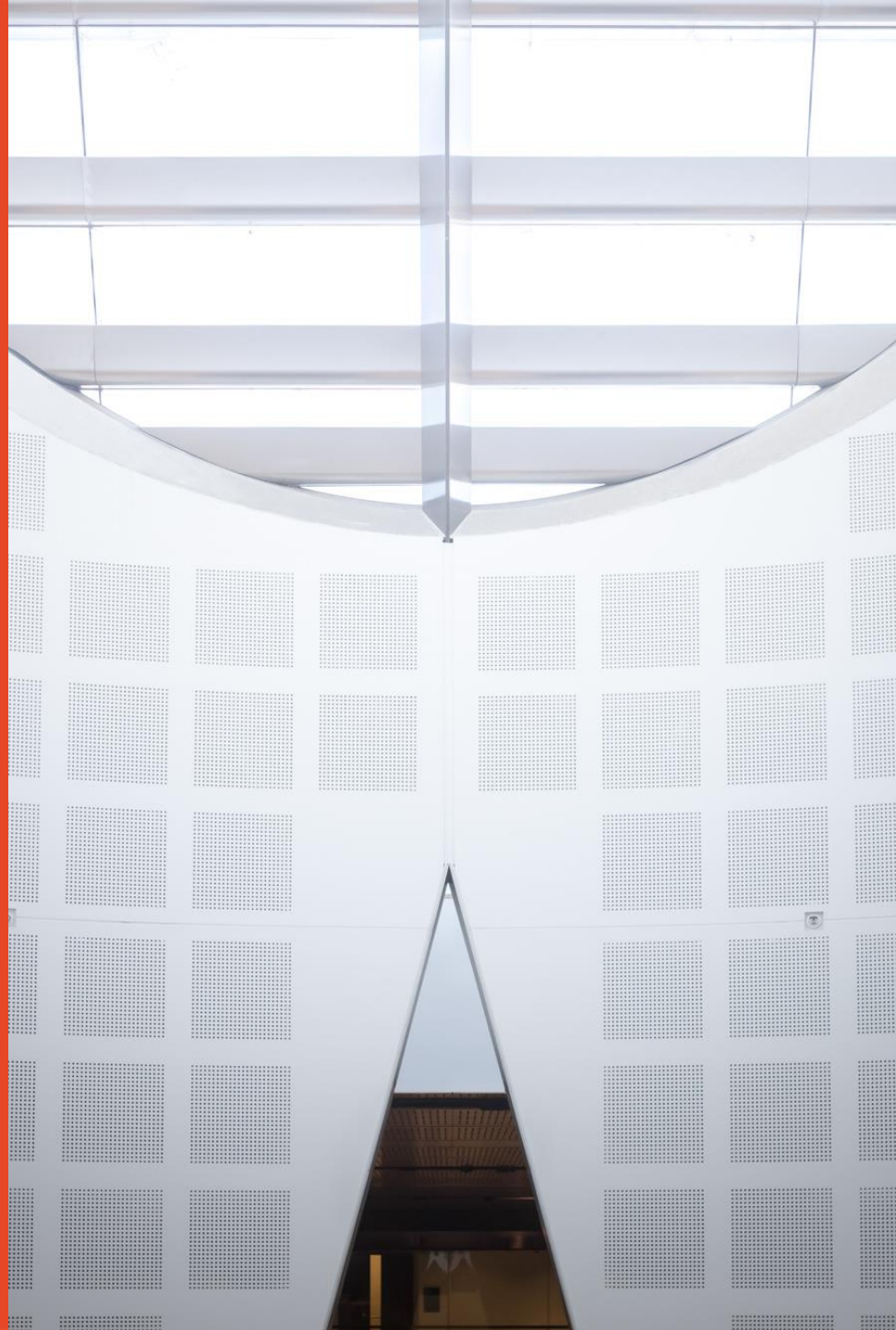
COMP5347: Web Application Development

Web Services

Dr. Basem Suleiman

School of Computer Science



THE UNIVERSITY OF SYDNEY

Outline

- Service Oriented Architecture
- Web Services
  - SOAP Services and RESTful Services
- Creating REST web services in Expressjs Application
- Consuming REST web services in Expressjs application

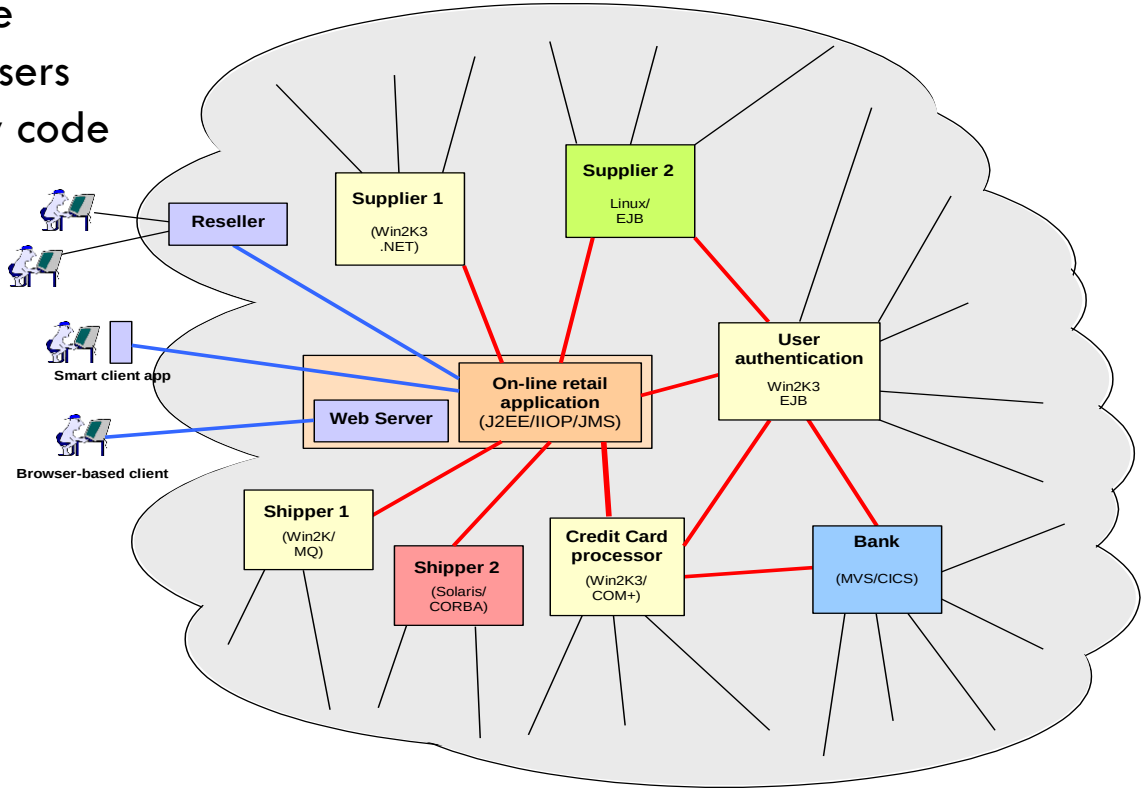
Service-Oriented Architecture (SOA)

*"a **style** of software design where **services** are provided to the other components by application components, through a communication protocol over a network"* – Wikipedia (May 2019)

- Paradigm, approach or style
- Can help to architect applications so that certain quality attributes are satisfied
- Services are the core of this approach

SOA – Web Applications

- Building large-scale integrated applications from services
  - Specific enough to have clear focus
- Integration and communication
- Platform independence
- Abstracted from end users
- Avoid rewriting legacy code



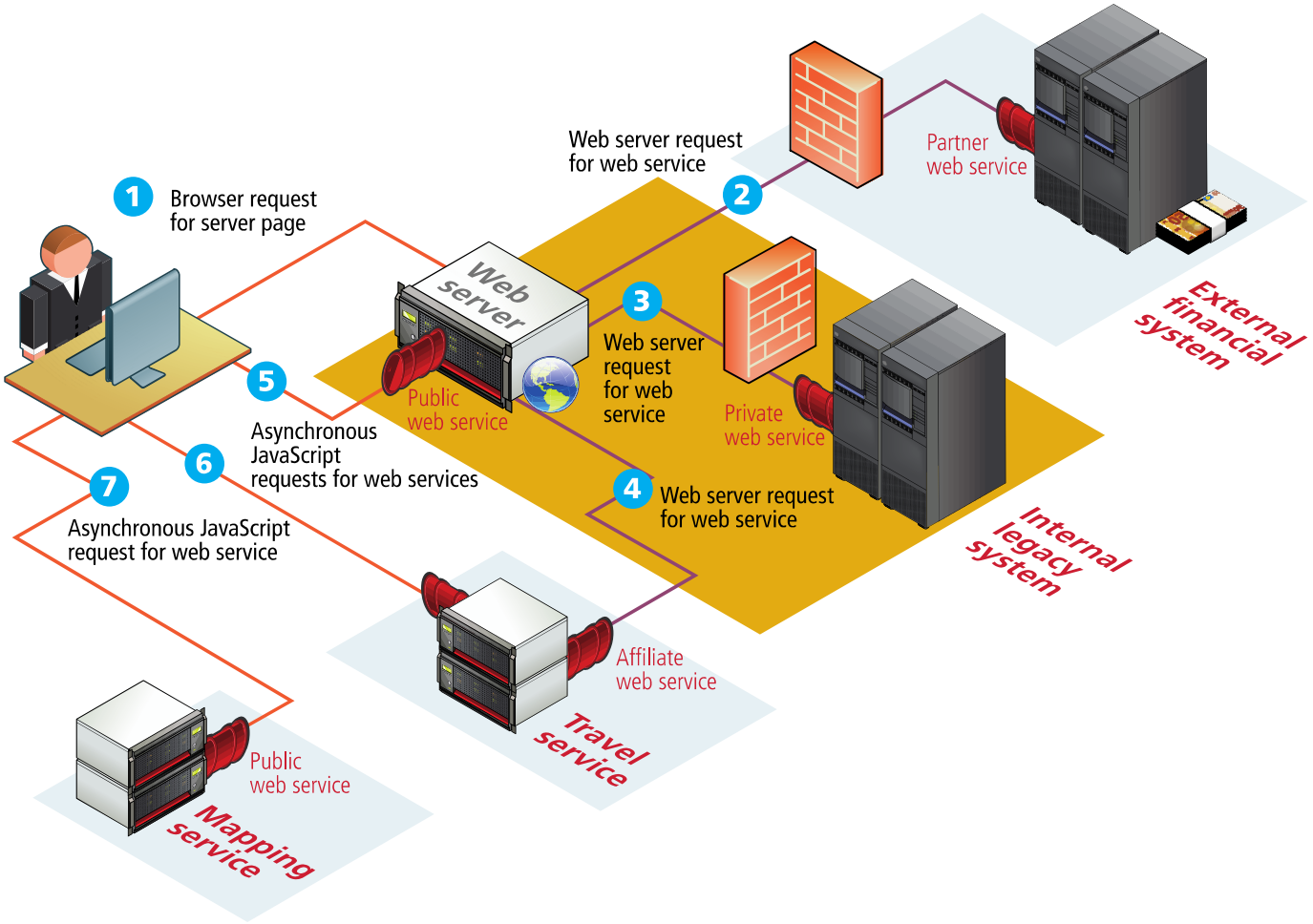
Outline

- Service Oriented Architecture
- Web Services
  - SOAP Services and REST Services
- Creating REST web services in Expressjs Application
- Consuming REST web services in Expressjs application

Web Services

- Relatively standardized mechanism by which one piece of software application can connect and communicate with another using web protocols
- It provides a simple and open way of **integrating** functions or data from various systems
- It can be used **within** an organization and/or **across** the public Internet
- At least two implementations: SOAP-based vs. RESTful services
- Original design of Web Services is very **application-centric** in contrast to the **resource-centric** Web and REST style

A Service-based Application

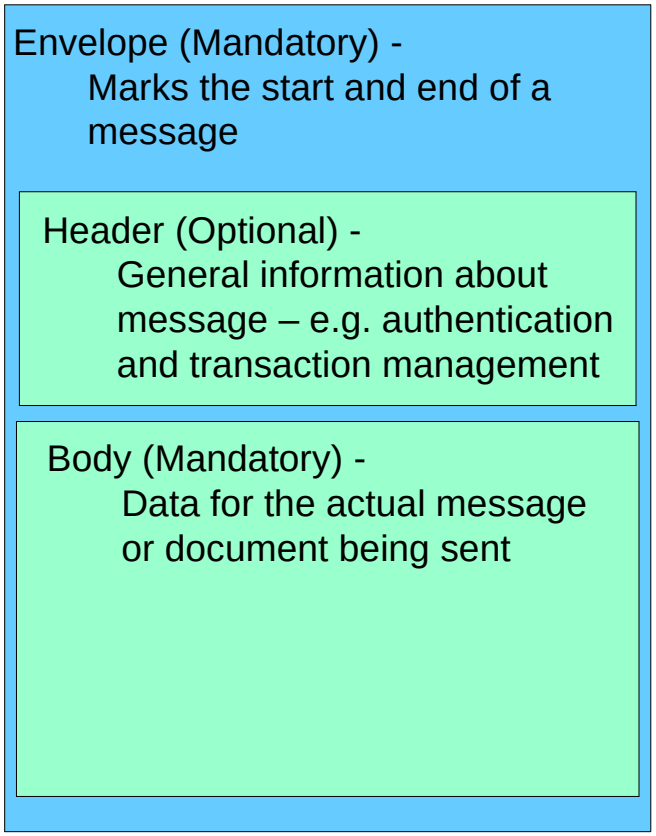


Web Service Standards

- Simple Object Access Protocol (SOAP): a messaging protocol for transferring information (XML format)
- Web Service Description Language (WSDL): A standard for describing Web services (XML format)
  - Interfaces, methods, parameters, request, response
  - Generated and consumed by tools
- Universal Description, Discovery and Integration (UDDI): a registry and protocol for publishing and discovering web services
  - Like white, Yellow and Green 'pages'
  - Not really used
- Also, WS-Policy framework for non-functional properties (e.g., security, reliability)

Calling A Service – SOAP

- SOAP – Simple Object Access Protocol!
  - Originally simple remote procedure calls using XML-formatted protocol
  - Envelope, header, body
  - Extensible protocol
    - Add new header elements for new semantics
  - Very widely accepted standard

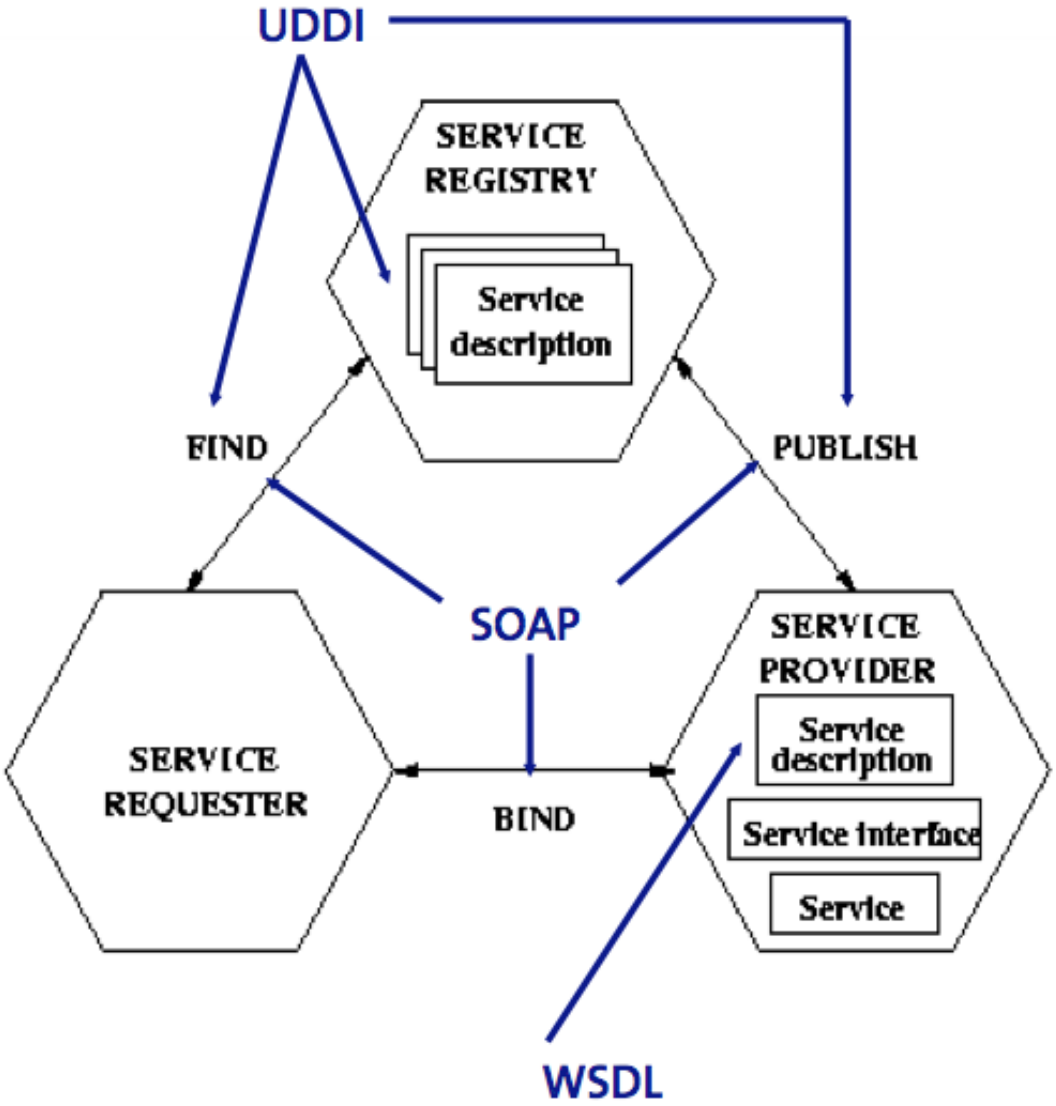


SOAP Example – Stock Quote

```
GET /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml
Content-Length: nnnn
SOAPMethodName: Stock-Namespace-URI#GetLastTradePrice
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
    <SOAP:Body>
        <m:GetLastTradePrice xmlns:m="Stock-Namespace-URI">
                <symbol>DIS</symbol>
        </m:GetLastTradePrice>
    </SOAP:Body>
</SOAP:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
    <SOAP:Body>
        <m:GetLastTradePriceResponse xmlns:m="Stock-NS-URI">
            <return>34.5</return>
        </m:GetLastTradePriceResponse>
    </SOAP:Body>
</SOAP:Envelope>
```

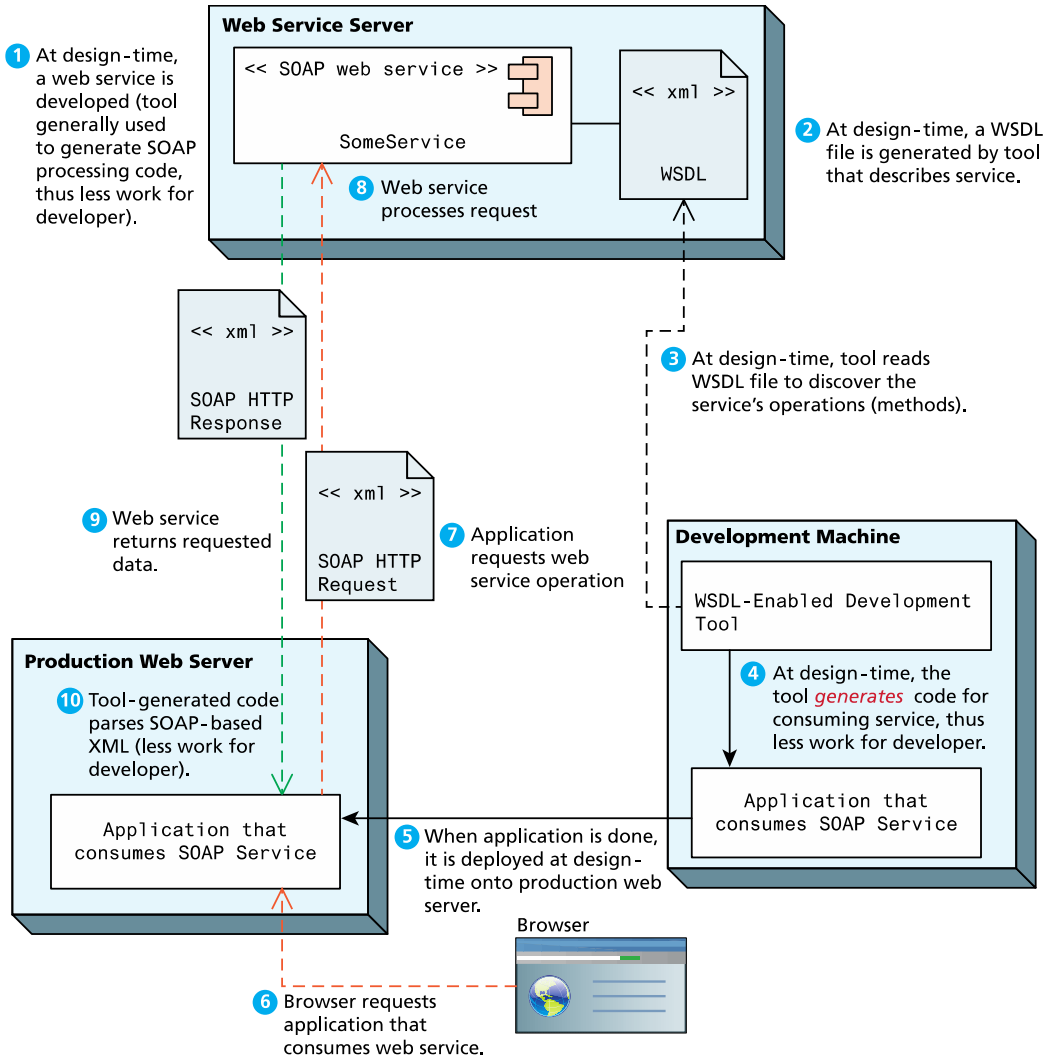
Publishing and Consuming a Service



WSDL Structure

```
<definitions>
 <types>
  definition of types........
 </types>

<message>
  definition of a message....
 </message>

 <portType>
  definition of a port.......
 </portType>

 <binding>
  definition of a binding....
 </binding>

</definitions>
```

SOAP-based Services – Workflow



Web Service APIs – Examples

- Twitter API
  - https://dev.twitter.com/rest/public
- MediaWiki API
  - https://www.mediawiki.org/wiki/API:Main_page
- Flickr API
  - http://www.flickr.com/services/api/
- Google Maps – Directions APIs
  - https://developers.google.com/maps/documentation/directions/intro
- New York Times API
  - http://developer.nytimes.com/docs
- Youtube API
  - https://developers.google.com/youtube/getting_started#data_api

REpresentation State Transfer (REST)

- REST is an architectural style rather than a strict protocol
- REST-style architectures consist of clients and servers
  - Requests and responses are built around the transfer of representations of resources
- A resource can be essentially any coherent and meaningful *concept* that may be addressed
- A representation of a resource is typically a document that captures the current or intended *state* of a resource

Based on Roy Fielding's doctoral dissertation, rephrased by Wikipedia http://en.wikipedia.org/wiki/Representational_State_Transfer

REST – API Format (Early Days)

- Many early day RESTful API's URL has a format consists of:
  - API end point (a concept coming from SOAP)
  - a parameter to specify the action: query, update, etc..
  - and many action specific parameters
  - Most of which are expressed as query strings

- Many API providers provide API sandbox or API explorer to help developer build the request URL

- Example:
  - **https://api.flickr.com/services/rest/?method=flickr.test.echo&name=value**
  - **https://en.wikipedia.org/w/api.php?action=query**&name=value

REST – API Common Format

- The commonly agreed REST API URL format conforms to general web architecture
  - Using URI/URL to specify resources
  - Using HTTP method to indicate action

- A URI as a resource identifier is one of the central concepts of WWW
  - The representation of an available resource, identified by a URI, is fetched using an HTTP GET request without affecting the resource in any way

- The simplicity and scalability of the Web is largely due to the fact that there are a few "generic" methods (GET, POST, PUT, DELETE)

REST – Basic Design Principles

- Use HTTP methods explicitly to interact with any resource on the Web
  - GET: query the state
  - POST: create a resource
  - PUT: modify or create a resource
  - DELETE: delete a resource

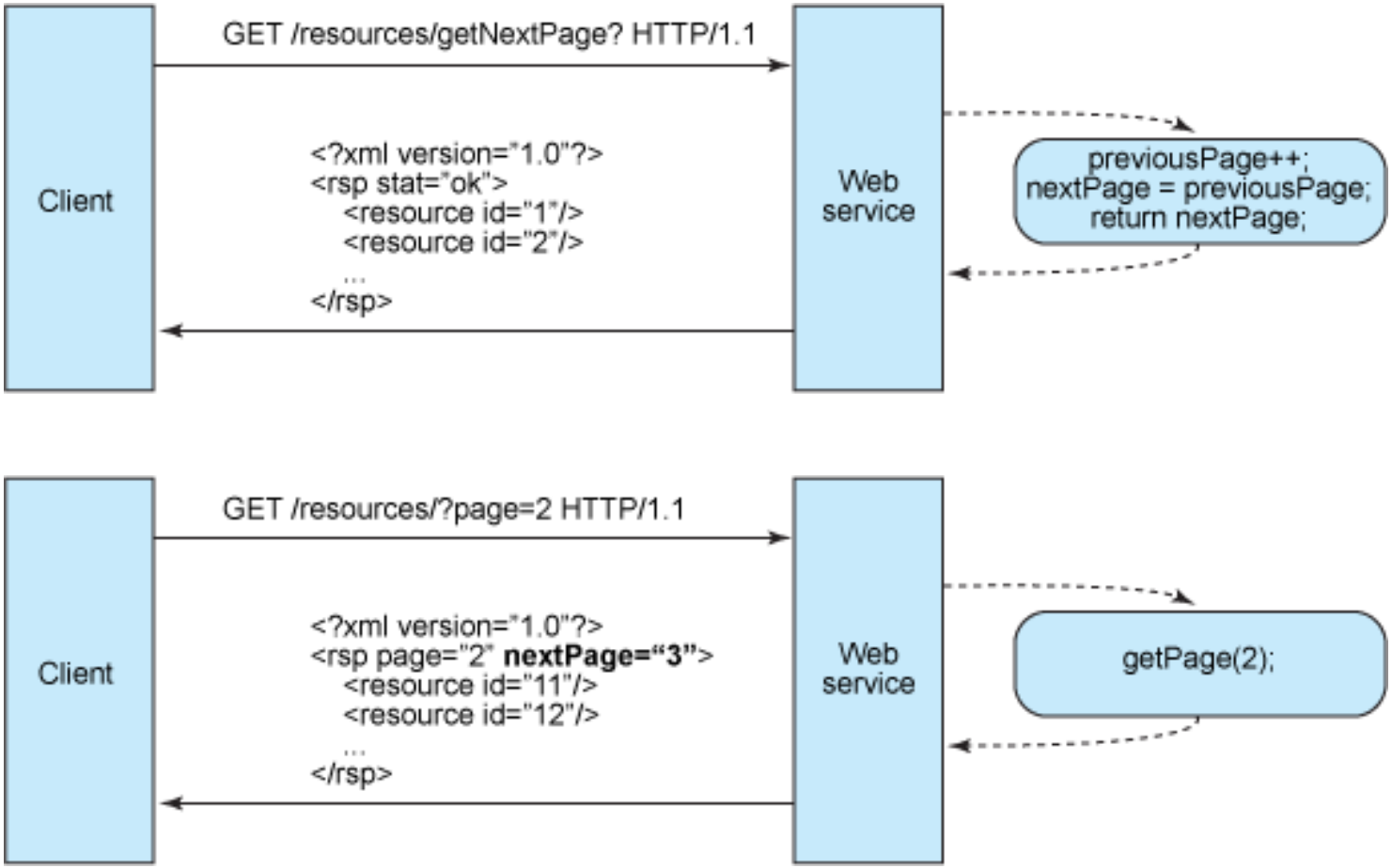
- Be stateless
  - Address the resources explicitly in the request message

- Expose directory structure-like URIs
  - http://www.myservice.org/discussion/topics/{topic}

- Transfer XML, JSON, or both

http://www.ibm.com/developerworks/webservices/library/ws-restful/

REST – Stateless vs. Stateful



http://www.ibm.com/developerworks/webservices/library/ws-restful/

Resource Types

- A resource is something "interesting" in your system
  - E.g., Blog posting, transaction, Printer
- Most of the time we can differentiate between *collection type of resources* and *individual resource*
  - Revisions and revision
  - Articles and article
- The URL's directory structure is based on that
- This can be nested and it is up to developers to decide the nesting direction
  - /movies/ForrestGump/actors/TomHanks
  - /directors/AngLee/movies/LifeOfPi

Web Services – MediaWiki API

https://en.wikipedia.org/w/api.php?action=query&prop=revisions&rvprop=ids|timestamp&rvstart=2016-12-01T00:00:00Z&rvend=2017-01-01T00:00:00Z&rvdir=newer&format=jsonfm&titles=cat&rvlimit=max


```
{
    "batchcomplete": "",
    "query": {
        "normalized": [
            {
                "from": "cat",
                "to": "Cat"
            }
        ],
        "pages": {
            "6678": {
                "pageid": 6678,
                "ns": 0,
                "title": "Cat",
                "revisions": [
                    {
                        "revid": 752709621,
                        "parentid": 752304215,
                        "timestamp": "2016-12-02T20:51:06Z"
                    },
                    {
                        "revid": 752713783,
                        "parentid": 752709621,
                        "timestamp": "2016-12-02T21:17:08Z"
                    },
```

REST – Request URLs and Methods

Action	URL path	Parameters	Example
Create new revision	/revisions		http://localhost:3000/revisions
Get all revisions	/revisions		http://localhost:3000/revisions
Get a revision	/revisions	revision_id	http://localhost:3000/revisions/123
Update a revision	/revisions	revision_id	http://localhost:3000/revisions/123
Delete a revision	/revisions	revision_id	http://localhost:3000/revisions/123

Request Method	Use case	Response
POST	Add new data in a collection	New data created
GET	Read data from data source	Data objects
PUT	Update existing data	Updated object
DELETE	Delete an object	NULL

Outline

- Service Oriented Architecture
- Web Services
  - SOAP Services and REST Services
- Creating REST web services in Expressjs Application
- Consuming REST web services in Expressjs application

Create REST API in Express.js

- Most Web frameworks provide feature to support developing Web services

- Express.js
- Additional route feature: *route parameters*
  - Route parameters are named URL segments that are used to capture the values specified at their position in the URL
  - The values are populated in **`req.params`** object

Create REST API in ExpressJs

- Example
  - Route path: /users/:userId/books/:bookId
  - Request URL: http://localhost:3000/users/34/books/8989
  - req.params: { "userId": "34", "bookId": "8989" }

```
app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params)
})
```

Specifying Client Data

- Sending data from client to server
  - **Route parameter**
    - Route path: `/users/:userId/books/:bookId`
    - Url: `http://localhost:3000/users/34/books/8989`
    - `req.params.userId`
    - `req.params.bookId`
  - **Query String**
    - url: http://localhost:3000/usersbooks?userId=34&bookId=8989
    - `req.query.userId`
    - `req.query.bookId`
  - **Request body**
    - data {userId:34, bookId:8989} is sent as part of request body
    - if using body-parser middleware
      - `req.body.userId`
      - `req.body.bookId`

Create REST API Using Express.js

model

```
RevisionSchema.statics.getByTitle = function(title, callback){
    return this.find({'title':title}).exec(callback)
}
```

controller

```
module.exports.getByTitle=function(req,res){
    title = req.params.title
    Revision.getByTitle(title,function(err,result){
        if (err){
            console.log("Cannot find revisions of title: " + title)
        }else{
            res.json(result)
        }
    })
}
```

route

```
router.get('/revisions/:title', controller.getByTitle)
```

RESP API – Response

http://localhost:3000/revision/revisions/BBC

app.use('/revision',revroutes)

Outline

- Service Oriented Architecture
- Web Services
  - SOAP Services and REST Services
- Creating REST web services in Expressjs Application
- Consuming REST web services in Expressjs application

Consume REST API in Express.js

- Complex API calls may benefit from using a package that wraps up the API

- Simple GET type of queries can always be implemented using general modules designed for handling http requests
  - Core **`node.js`** modules: **`http`**, **`https`**
  - **`request`** module

- The **`request`** module (https://github.com/request/request)
  - To install
    - npm install request –save
  - To make a request: **`request(options, callback)`**

Consume REST API in Express.js

- The **`request`** module (https://github.com/request/request)
  - To install
    - npm install request –save
  - To make a request: **`request(options, callback)`**

```
var request = require('request');
request('http://www.google.com', function (error, response, body) {
  console.log('error:', error); // Print the error if one occurred
  console.log('statusCode:', response && response.statusCode); // Print the response status code if a response was re
  console.log('body:', body); // Print the HTML for the Google homepage.
});
```

Call Wikipedia API – Code Example

```
var request = require('request');

var wikiEndpoint = "https://en.wikipedia.org/w/api.php",
    parameters = ["action=query",
                  "format=json",
                  "prop=revisions",
                  "titles=australia",
                  "rvstart=2016-11-01T11:56:22Z",
                  "rvdir=newer",
                  "rvlimit=max",
                  "rvprop=timestamp|userid|user|ids"]
var url = wikiEndpoint + "?" + parameters.join("&")
console.log("url: " + url)
var options = {
    url: url,
    Accept: 'application/json',
    'Accept-Charset': 'utf-8'
}
```

Wikipedia PAI documentation: https://www.mediawiki.org/wiki/API:Main_page

Wikipedia API Call – Making Request

```
request(options, function (err, res, data){
    if (err) {
        console.log('Error:', err);
    } else if (res.statusCode !== 200) {
        console.log('Status:', res.statusCode);
    } else {
       json = JSON.parse(data);
       pages = json.query.pages
       revisions = pages[Object.keys(pages)[0]].revisions
       console.log("There are " + revisions.length + " revisions.");
       var users=[]
       for (revid in revisions){
            users.push(revisions[revid].user);
       }
       uniqueUsers = new Set(users);
       console.log("The revisions are made by " + uniqueUsers.size + "
   unique users");
    }
});
```

MediaWiki API

- https://en.wikipedia.org/w/api.php?action=query&prop=revisions&rvprop=ids|timestamp&rvstart=2016-12-01T00:00:00Z&rvend=2017-01-01T00:00:00Z&rvdir=newer&format=jsonfm&titles=cat&rvlimit=max


```
{
    "batchcomplete": "",
    "query": {
        "normalized": [
            {
                "from": "cat",
                "to": "Cat"
            }
        ],
        "pages": {
            "6678": {
                "pageid": 6678,
                "ns": 0,
                "title": "Cat",
                "revisions": [
                    {
                        "revid": 752709621,
                        "parentid": 752304215,
                        "timestamp": "2016-12-02T20:51:06Z"
                    },
                    {
                        "revid": 752713783,
                        "parentid": 752709621,
                        "timestamp": "2016-12-02T21:17:08Z"
                    },
```

Calling Wikipedia API – https Version

```
var https = require('https')

var wikiEndpointHost = "en.wikipedia.org",
    path = "/w/api.php"
    parameters = ["action=query",
        "format=json",
        "prop=revisions",
        "titles=australia",
        "rvstart=2016-11-01T11:56:22Z",
        "rvdir=newer",
        "rvlimit=max",
        "rvprop=timestamp|userid|user|ids"],
   headers = {
        Accept: 'application/json',
        'Accept-Charset': 'utf-8'
   }

var full_path = path + "?" + parameters.join("&")

var options = {
     host: wikiEndpointHost,
     path: full_path,
     headers: headers}
```

https://nodejs.org/api/https.html

https version (cont'd)

```
https.get(options,function(res){
    var data ='';
    res.on('data',function(chunk){
        data += chunk
    })
    res.on('end',function(){
        json = JSON.parse(data);
        pages = json.query.pages
        revisions = pages[Object.keys(pages)[0]].revisions
        console.log("There are " + revisions.length + " revisions.");
        var users=[]
        for (revid in revisions){
            users.push(revisions[revid].user);
        }
        uniqueUsers = new Set(users);
        console.log("The revisions are made by " + uniqueUsers.size + " unique users");
  })
}).on('error',function(e){
    console.log(e)
})
```

References

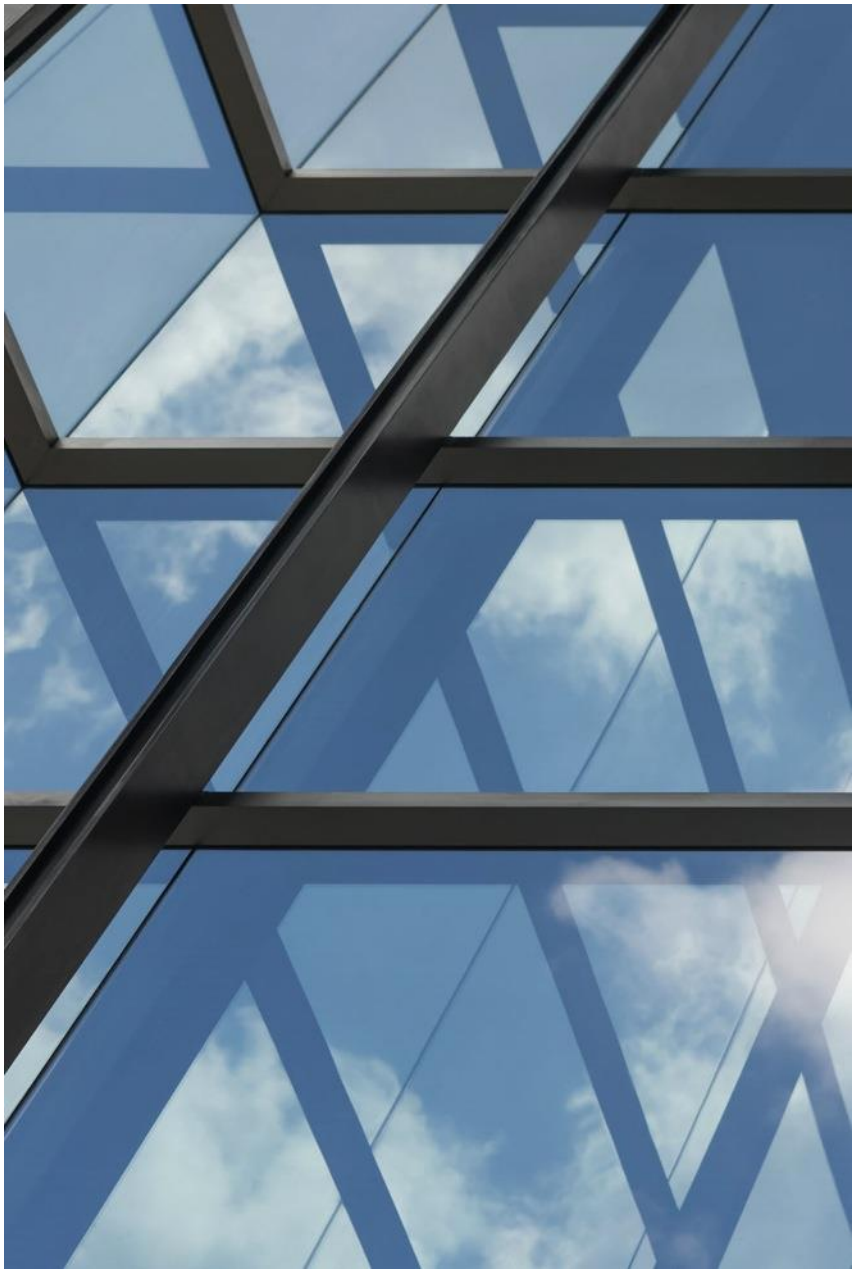
- Randy Connolly, Ricardo Hoar, Fundamentals of Web Development, Global Edition, Pearson

- Wikipedia PAI documentation: https://www.mediawiki.org/wiki/API:Main_page

- Node.js https API. https://nodejs.org/api/https.html

W10 Tutorial: Web Services (Wikipedia API)

W11 Lecture: Web Application Security



THE UNIVERSITY OF SYDNEY