REST & AJAX

FRICTIONLESS FLOW OF INFORMATION

REST Web Services

REST = **RE**presentational **S**tate **T**ransfer

REST is an architectural style consisting of a coordinated set of architectural constraints

First described in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine.

RESTful is typically used to refer to web services implementing a REST architecture.

Unlike SOAP-based web services, there is no "official" standard for RESTful web APIs

such as SOAP.

Simple HTTP client/server mechanism to exchange data

Everything – the UNIVERSE is available through a URI

Utilizes HTTP: GET/POST/PUT/DELETE operations

Architectural Constraints

Client–server

Separation of concerns. A uniform interface separates clients from servers.

Stateless

The client–server communication is further constrained by no client context being stored on the server between requests.

Cacheable

Basic WWW principle: clients can cache responses.

Layered system

A client cannot necessarily tell whether it is connected directly to the end server, or to an intermediary along the way.

Uniform interface

Individual resources are identified in requests, i.e.,using URIs in web-based REST systems.

RESTful API HTTP methods

Resource	GET	PUT	POST	DELETE
Collection URI, such as http://example.com /resources	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	**Delete** the entire collection.
Element URI, such as http://examples .com /resources/item17	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it doesn't exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.	**Delete** the addressed member of the collection

POST means "create new" as in "Here is the input for creating a user".
PUT means "insert, replace if already exists" as in "Here is the data for user 5".
PUT is Idempotent

Idempotent means that multiple calls with the same operation doesn't change the representation

Spring MVC REST-style Controller

Essentially means receive & send the content directly as the message body instead of structuring HTML pages.

We are **NOT** using HTML

We are using well-formed XML OR JSON

Spring support is based on the

@REQUESTBODY & @RESPONSEBODY annotations

Also

@ResponseStatus(value = HttpStatus.***NO_CONTENT)***

For deletes, creates, updates…

RequestBody & ResponseBody

@ResponseBody

Spring framework uses the "Accept" header of the request to decide the media type to send to the client

@RequestBody

Spring framework will use the “*Content-Type*” header to determine the media type of the Request body received.

To get XML, MIME media type = "application/xml"

To get JSON, MIME media type = "application/json “

JSON (JavaScript Object Notation)

```
{
"productId":"P1235",
"name":"Dell Inspiron",
"unitPrice":700,
"description":"Dell Inspiron 14-inch Laptop (Black) with 3rd Generation Intel Core processors",
"manufacturer":"Dell",
"category":"Laptop",
"unitsInStock":1000,
"unitsInOrder":0,
"discontinued":false,
"condition":null
}
```

RESTful Web Service CartRestController

Add Product to cart

```
@RequestMapping(value = "/add/{productId}", method = RequestMethod.PUT)
@ResponseStatus(value = HttpStatus.NO_CONTENT)
 public void addItem(@PathVariable("productId") String productId,
                                     HttpServletRequest request) {
```

- Get Product in cart for display

```
@RequestMapping("/product/{id}", method= RequestMethod.GET)
public @ResponseBody Product
               getRestProduct((@PathVariable(“id") String productId){
```

RESTful input Validation

How does THAT work?

XML - Schema validation is generally not a good idea in a REST service.

A major goal of REST is to decouple client and server so that they can evolve separately.

What about JSON validation/consistency?

API producers have frequently developed their own JSON response formats in the absence of well-defined standards.

ALTERNATIVE OPTION: JSR-303 Bean Validation

Main Point

REST is defined by architectural constraints. For example, it is able to access information through the URI. Everything on the web is available through a URI.

Everything in creation is available through understanding and experience of the Unified Field of Consciousness

Web 2.0

WWW sites that emphasize user-generated content, usability, and interoperability.

Update selected regions of the page area without undergoing a full page reload.

Ajax and JavaScript frameworks:

YUI Library - Dojo Toolkit - Moo Tools - **jQuery**

Ext JS - Prototype JavaScript Framework.

Ember.js, React.js, AngularJS Backbone.js

Payload - typically formatted in XML or JSON

NOTE: we will use Spring MVC REST technology to create the Web 2.0 payload

AJAX

Asynchronous **J**avascript **A**nd **X**ML

Web applications are able to make quick, incremental updates to the client without reloading the entire browser page

The use of XML is not required; JSON is often used instead (AJAJ)

Examples :

* Validate values of form fields before saving

* Dynamically load dropdown values from database

* Load table data and paginations

* Live Search

***Ajax* -** a broad group of Web technologies that communicates with a server in the background, without interfering with the current state of the page.

Jquery

SLOGAN:

The Write Less, Do More, JavaScript Library.

Fast, small, and feature-rich JavaScript library.

HTML document traversal and manipulation, event handling, animation, and Ajax

Cart.js

RESTful services:

addToCart	- Add Item
removeFromCart	- Remove Item
showProduct	- Show details of product in Cart

Cart.js addToCart Function

```
 var contextRoot = "/" + window.location.pathname.split('/')[1];

addToCart = function(productId){
$.ajax({
      url: contextRoot + '/rest/cart/add/' + productId,
      type: 'PUT',
      dataType: "json", // Accept  header
      success: function(response){
             alert("Product Successfully added to the Cart!");
     },
      error: function(){
             alert('Error while request..');
         }
     });
 }
```

Jquery AJAJ View example

```
<script type="text/javascript" src="http://code.jquery.com/
                                          jquery-1.10.1.min.js"></script>
<script type="text/javascript" src="<spring:url
                         value="/resource/js/cart.js"/>"></script>
```

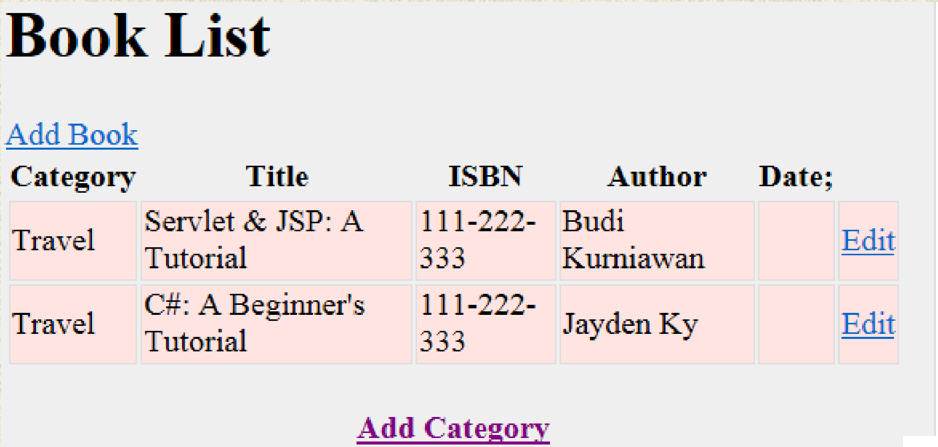
Invoke the Javascript add item function

```
 <a href="#" class="btn btn-warning btn-large"
                         onclick="addToCart('${product.productId}')">
```

Invoke the Javascript remove item function

```
<a href="#" class="label label-danger"
        onclick="removeFromCart('${item.value.product.productId}')">
```

Rest Service Form Validation

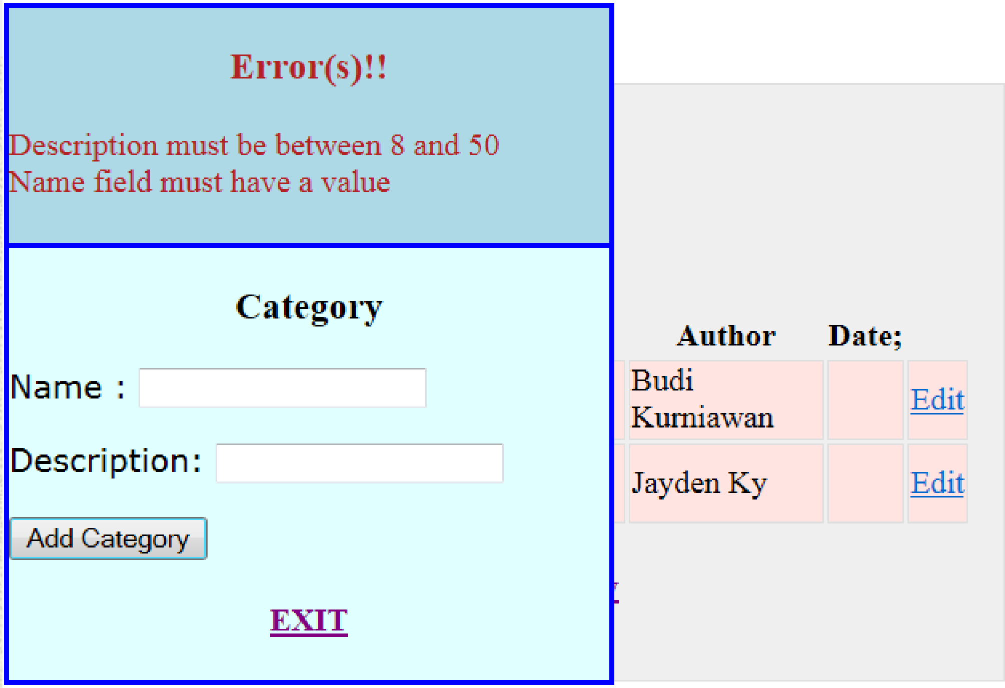


Book List

Add Book

Category	Title	ISBN	Author	Date;	
Travel	Servlet & JSP: A Tutorial	111-222-333	Budi Kurniawan		Edit
Travel	C#: A Beginner's Tutorial	111-222-333	Jayden Ky		Edit

Add Category



Error(s)!!

Description must be between 8 and 50
Name field must have a value

Category

Name :

Description:

EXIT

Author	Date;	
Budi Kurniawan		Edit
Jayden Ky		Edit

RequestBody & ResponseBody

bookList.jsp

```
<form id= "categoryForm" method="post">
                                   .....
 <input type="button" value="Add Category"
       onclick="categorySubmit();return false;">
```

AJAJ CALL:

```
function categorySubmit(){
    var send =
      JSON.stringify(serializeObject($('#categoryForm')));
$.ajax({
      url: '/Book5Rest/addCategory',
      type: 'POST',
      dataType: "json",            // Accepts
      data:send,
      contentType: 'application/json', // Sends
      success: function(){
```

AJAJ Call Continued…

```
error: function(errorObject ){
//error: function(jqXHR,  textStatus,  errorThrown ){
//example- function(jQuery XMLHttpRequest, "error","Bad Request" )

if (errorObject.responseJSON.errorType == "ValidationError") {
       $('#success').html("");
       $("#errors").append( '<H3 align="center"> Error(s)!! <H3>');
      $("#result").append( '<p>');

      var errorList = errorObject.responseJSON.errors;
      $.each(errorList,  function(i,error) {
               $("#errors").append( error.message + '<br>');
       });
       $("#errors").append( '</p>');
       make_visible('result');
}
else {
alert(errorObject.responseJSON.errors(0));  // "non" Validation Error
```

RequestBody & ResponseBody

Form Rountrip done with RESTful Web Service

```
@RequestMapping(value =
                    "/addCategory",method=RequestMethod.POST)
public @ResponseBody Category add(@Valid @RequestBody Category category)  {
               categoryService.save(category);
               return category;
              }
```

-

if NO BindingResult bindingResult in Signature,

MethodArgumentNotValidException will be thrown if Category fails validation...

Form Validation Exception Handling

```
@ExceptionHandler(MethodArgumentNotValidException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ResponseBody
    public DomainErrors handleException (
               MethodArgumentNotValidException exception) {
        List<FieldError> fieldErrors =
               exception.getBindingResult().getFieldErrors();

        DomainErrors errors = new DomainErrors();
        errors.setErrorType("ValidationError");
        for (FieldError fieldError : fieldErrors) {
         DomainError error = new DomainError(
                      messageAccessor.getMessage(fieldError));
         errors.addError(error);
        }
         return errors;
    }
```

Welcome Student Example

```
@RequestMapping(value = "/welcomeStudent", method = RequestMethod.GET)
    public @ResponseBody String[] displayWelcome( ) {
               .  .. .. ..
          String [] result = {
               String.valueOf(totalCells),
               String.valueOf(cellCounter),
               "Welcome to " + currentStudent
          };
      return result;
```

To Kick off AJAX: setInterval(welcome, 750);

```
function welcome() {
        $.ajax({
            url : 'welcomeStudent',
            success : function(data) {
            total =  parseInt(data[0]);
            counter =  parseInt(data[1]);
                $('.welcome').html(data[2]);
            duke();
```

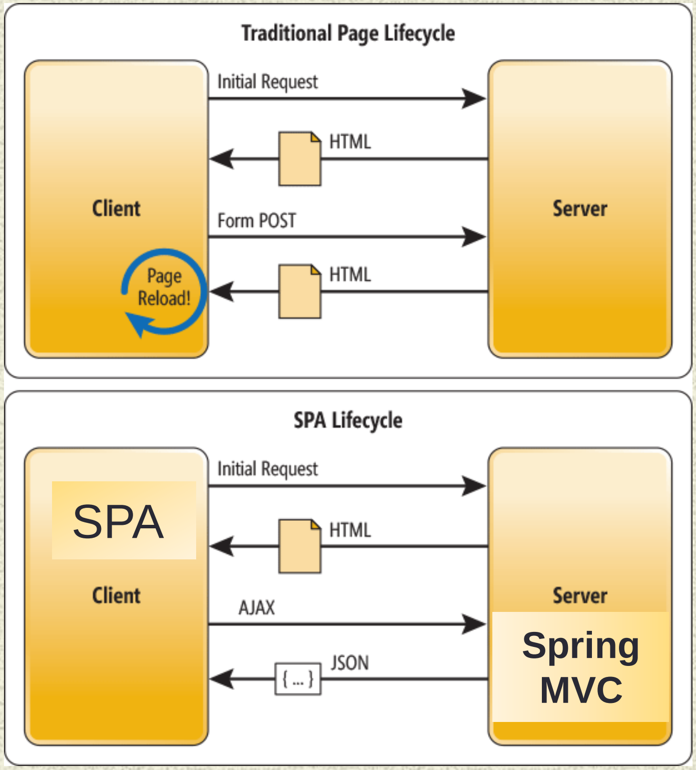
Spring MVC Rest Controller & SPA

SPA – Single Page Application

Web application that fits on a single to provide a more fluid user experience
All the "heavy lifting" presentation-wise is done on the client side, in JavaScript.

Server interaction with a SPA involves dynamic communication with the web server behind the scenes [RE: Ajax]

Spring RESTful Controller is a perfect fit for Server side support of SPA.



Main Point

Ajax uses Javascript in a browser to access data or functionality residing on a server and then quickly and efficiently selectively update the browser.

In general, Nature is maximally efficient.

WebSockets [TCP for the Web] HTTP Alternative

Use Case [VERSUS HTTP & REST]

Low latency and high frequency messaging

Ideal for rich web applications

Examples:
Real-time gaming
Social feeds –real time
Collaborative editing
Financial tickers.

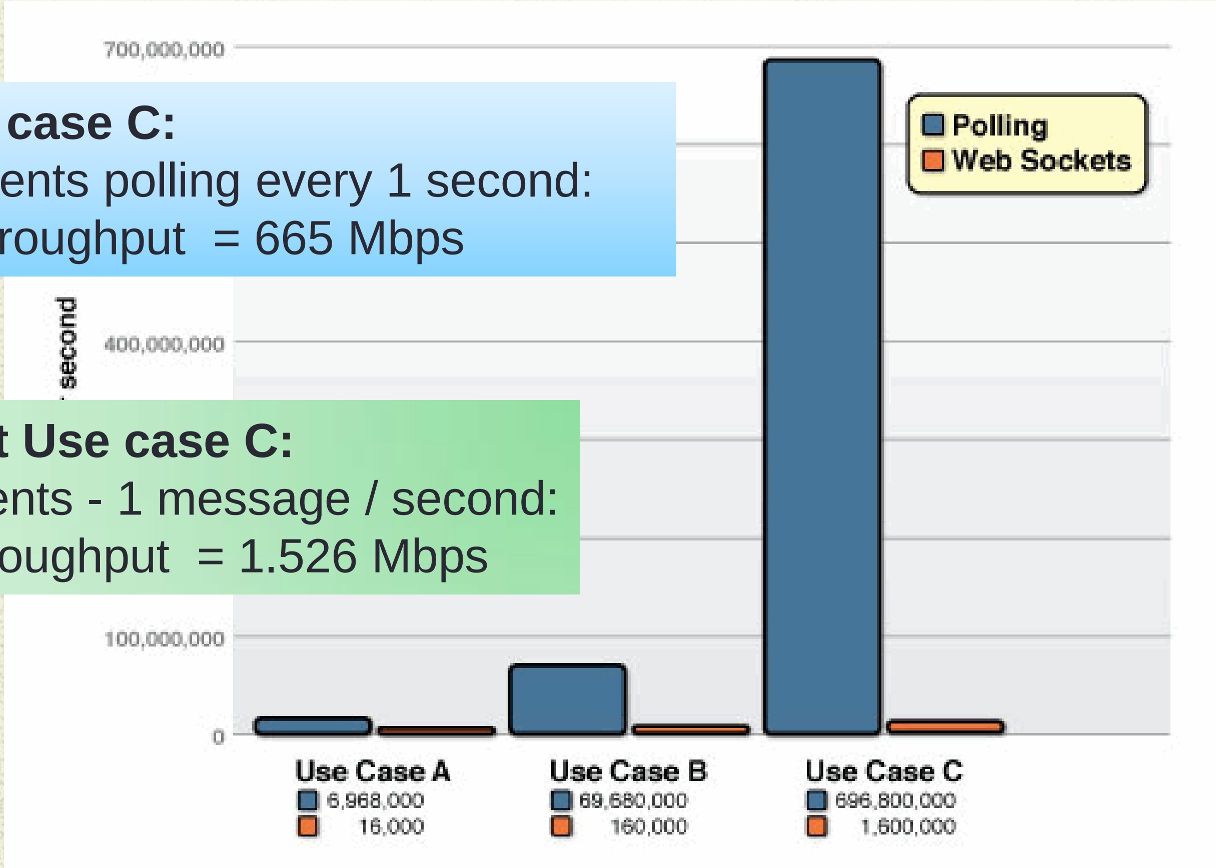
HTTP resorts to polling for “dynamic” web applications

Leads to Exceptional Overhead and Latency

Throughput

Throughput

"Reducing kilobytes * of data to 2 bytes…and reducing latency from 150ms to 50ms is far more than marginal. In fact, these two factors alone are enough to make Web Sockets seriously interesting to Google.“
* HTTP headers,etc.



HTTP Use case C:
100,000 clients polling every 1 second:
Network throughput = 665 Mbps

WebSocket Use case C:
100,000 clients - 1 message / second:
Network throughput = 1.526 Mbps

WebSocket - REST/HTTP differences

Bi-directional

HTTP - uni-directional protocol

Request initiated by client to server = request/response

WebSocket - bi-directional protocol

no defined message pattern [request/response]

Either client or server can send a message to other party.

Full-duplex

HTTP Half duplex - either client is talking or server is talking but NOT both.

WebSocket Full-duplex - client and server to talk independent of each other.

Single TCP Connection:

HTTP - new TCP ***connection for each HTTP request /response cycle***.

WebSocket - Single TCP ***connection for the lifecycle of WebSocket***.

Lean protocol:

HTTP is a chatty protocol. HTTP Headers

WebSocket [STOMP] ***is a simple, messaging protocol***

]

Technical details

WebSocket protocol RFC 6455

Full-duplex, 2-way communication between client- server.

Event-driven, reactive messaging architecture.

HTTP is used for the initial handshake

Mechanism built into HTTP to request a protocol switch

HTTP is switched to STOMP — a simple, messaging protocol

Streaming Text Oriented Messaging Protocol

Browser Support

Websocket support still ***"maturing"***

Client-side "fallback" support required

SockJS java script library [CDN library]

provides cross browser compatibility

supports STOMP protocol

.

Stomp.js

Facilitates STOMP over WebSocket

Stomp over WebSocket

Spring MVC WebSocket Java Configuration Example

/topic prefix indicates [pub-sub]
queue would indicate [point-to-point]
Client registers “there” for messages

/TickerApp prefix - identifies messages to be processed by server
stompClient.send("**/TickerApp/addStock**“)

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig extend
AbstractWebSocketMessageBrokerConfi
 @Override
 public void configureMessageBroker(MessageBrokerRegistry config) {
 config.enableSimpleBroker("/topic");
 config.setApplicationDestinationPrefixes("/TickerApp");
}
  @Override
 public void registerStompEndpoints(StompEndpointRegistry registry) {
     registry.addEndpoint("/ticker").withSockJS();
  }
```

/ticker is the resource that the client will connect to…

```
Client side Javascript:
var socket = new SockJS("/WebSockerTicker/ticker");
var stompClient = Stomp.over(socket);
```

Spring MVC WebSocket XML Configuration Example

/TaxiApp prefix - identifies messages to be processed by server

```
<websocket:message-broker application-destination-prefix="/TaxiApp">
    <websocket:stomp-endpoint path="/taxi">
      <websocket:sockjs/>
    </websocket:stomp-endpoint>
    <websocket:simple-broker prefix="/topic"/>
  </websocket:message-broker>
```

/taxi is the resource that the client will connect to…

/topic prefix indicates [pub-sub]
queue would indicate [point-to-point]
Client registers "there" for messages

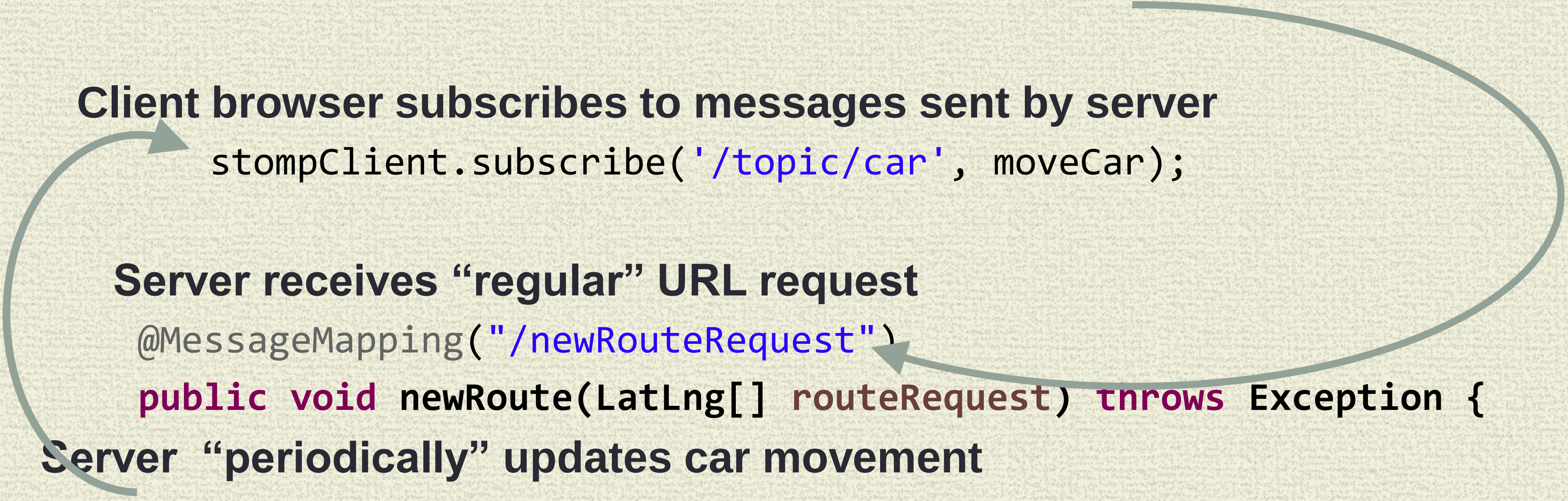
```
    var socket = new SockJS("/ComproTaxi/taxi");
    var stompClient = Stomp.over(socket);
```

ComPro Taxi

Two way communication between Client and Server.

Client Browser application sends Request to Server

```
// Send new Route request to server
stompClient.send("/TaxiApp/newRouteRequest ", {}, sendData);
```



Client browser subscribes to messages sent by server

```
stompClient.subscribe('/topic/car', moveCar);
```

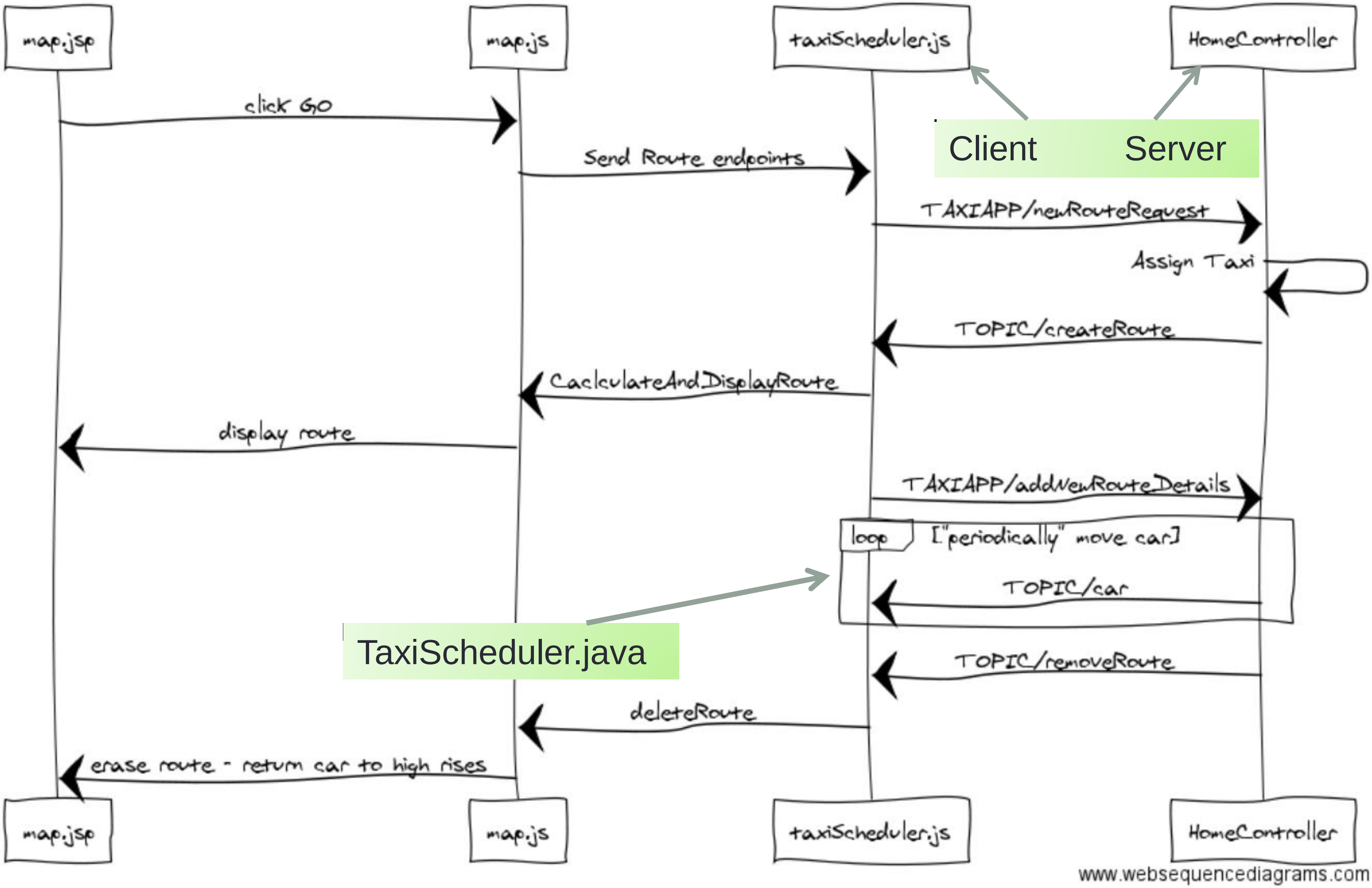
Server receives “regular” URL request

```
@MessageMapping("/newRouteRequest")
public void newRoute(LatLng[] routeRequest) throws Exception {
```

Server “periodically” updates car movement

```
TaxiMessage taxiMessage = new TaxiMessage(taxi.getName(),latLng);
template.convertAndSend("/topic/car", taxiMessage);
```

ComPro Taxi "Routing"



Client Server

TaxiScheduler.java

Main Point

- WebSockets is an HTTP alternative. It is a complementary technology that handles low latency and high frequency messaging scenarios.