Web Services

Web Services

- Approach for synchronous communications between applications
- Based on standards
  - HTTP
  - XML / JSON
- Provide a language and platform neutral way for applications to integrate

SOAP

- Simple Object Access protocol
- Common message structure for requests/ responses
- Not protocol specific - can be used over HTTP or other messaging mechanism
- Interfaces defined using WSDL
- Heavyweight approach with very verbose XML

SOAP Encoding

- Data is formatted based on SOAP Encoding
    - Encoding (http://schemas.xmlsoap.org/soap/encoding)
    - Schema instance (http://www.w3.org/2001/XMLSchema-instance)
- Building blocks for sending typed values using SOAP

Supported Encodings

- Simple data types
  - String, floats, ints
- Arrays
- Compound types
  - Structs or classes
- Null values
- Enumerations

SOAP Header

- Directives to the SOAP processor
- Example use
  - login credentials

SOAP Body

- Payload of the message
- Sender request
- Server response (including errors)
- XML document

WSDL

- Web Service Definition Language
- Contract for a web service
  - just like an Interface in Java / .NET
- Describes
  - how to access the service
  - method names
  - method parameters
  - method return types

SOAP Benefits

- Great tool support (especially in Microsoft tools)
- Client-side proxy code is easily generated
  - avoids hand coding of XML
- Industry standard
- Lots of WS-* standards
  - WS-Security, WS-ReliableMessaging
- Formal Contracts

SOAP Issues

- Very verbose
  - slow to send
  - slow to process
  - XML has those sharp angle brackets (<>), if you’re not careful you might cut yourself
- Difficult to cache responses
- Overkill for simple retrieval of information
- Limited / no browser support

RESTful Web Services

- REpresentational State Transfer
- Divid application state and function into "resources"
- Use URLs to request resource
- Resource may return XML, JSON, links to other resources, binary data, etc, etc, etc.
- HTTP method used can tell the server what operation to perform

Pros / Cons

- Good response times
  - lightweight
  - more easily cached
- Simplified programming model
- Browser friendly - easy to test without a specialized tool
- Works well for stateless operations
- Informal contract
- Not an industry standard per see

REST with Grails

- Controllers model RESTful API conventions
- Rather than returning view-centric information data can be returned
  - XML
  - JSON
  - Text/HTML
- The request method can be inspected to perform different operations
- Grails support for URL Mappings can also make REST easy to implement

URL Mapping Default

```
class UrlMappings {
    static mappings = {
        "/$controller/$action?/$id?" {
            constraints {
                // optional constraints
            }
        }
    }
}
```

http://localhost:8080/calltrack/product/show/1
same as:
http://localhost:8080/calltrack/product/show?id=1

URL Mapping for REST

- "/product/$id?"(resource:'product')
  - maps the /product uri to ProductController
- "/**api**/product/$id?"(resource: 'product')
  - maps the /**api**/product uri to ProductController

Content Negotiation

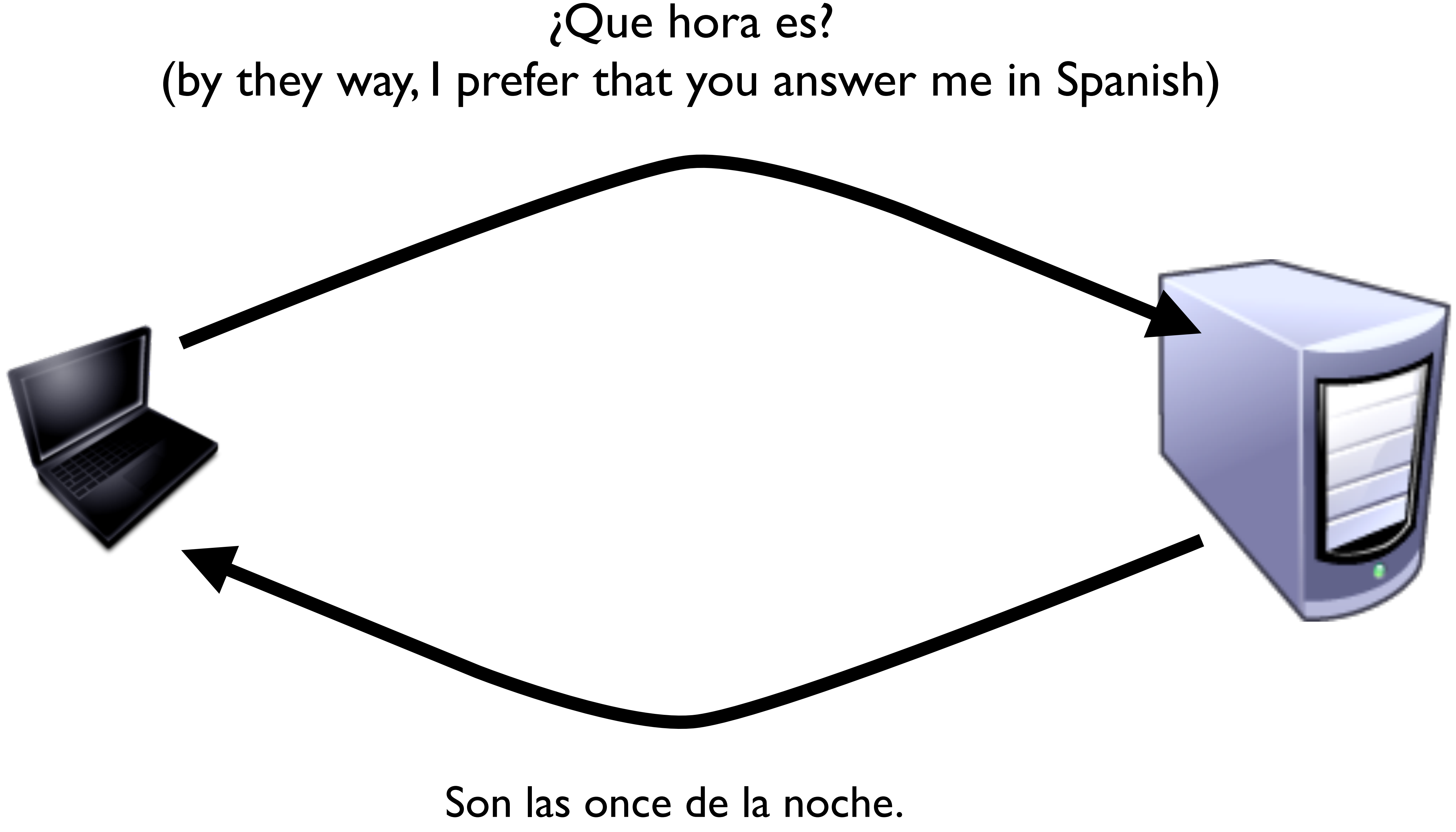
- A resource is a resource is a resource
  - what language the client / server speak (JSON, XML, HTML etc) is not relevant to the resource itself
- Use Accept header to respond to requests appropriately

Content Negotiation

- When a controller action is invoked the request includes an ACCEPT header describing the kind of content accepted by the requestor
  - MIME types (text/html, application/xml, etc.)
- Browsers typically supply a comma delimited list of these values with each request
  - Each one can be quality rated to allow the server to rank preferred response types (q value)
  - Chrome Example:
    - Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

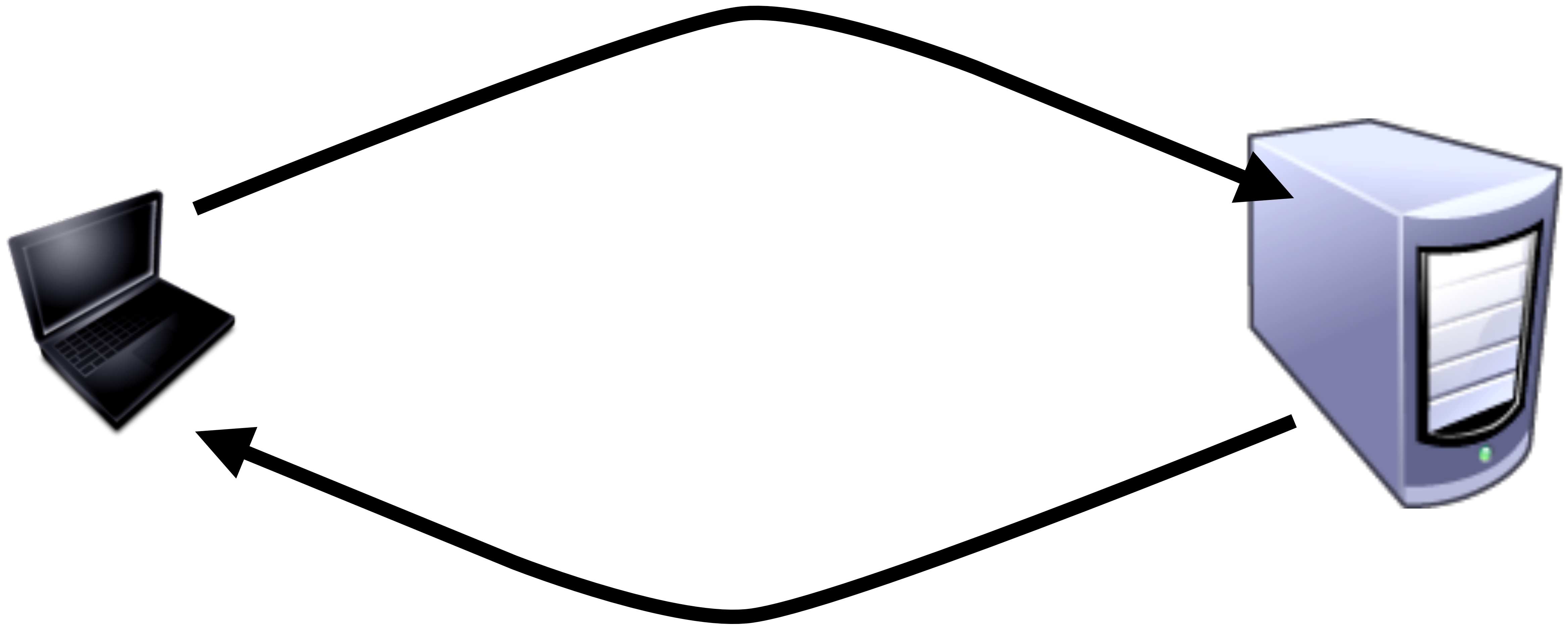
Alternatives to Accept Header

- File extension
  - include a file extension on the URL of the type you want
  - http://localhost:8080/theapi/book/show/1.json
- Request parameter
  - http://localhost:8080/theapi/book/show/1?format=json


¿Que hora es?
(by they way, I prefer that you answer me in Spanish)
Son las once de la noche.

Request: ¿Que hora es?

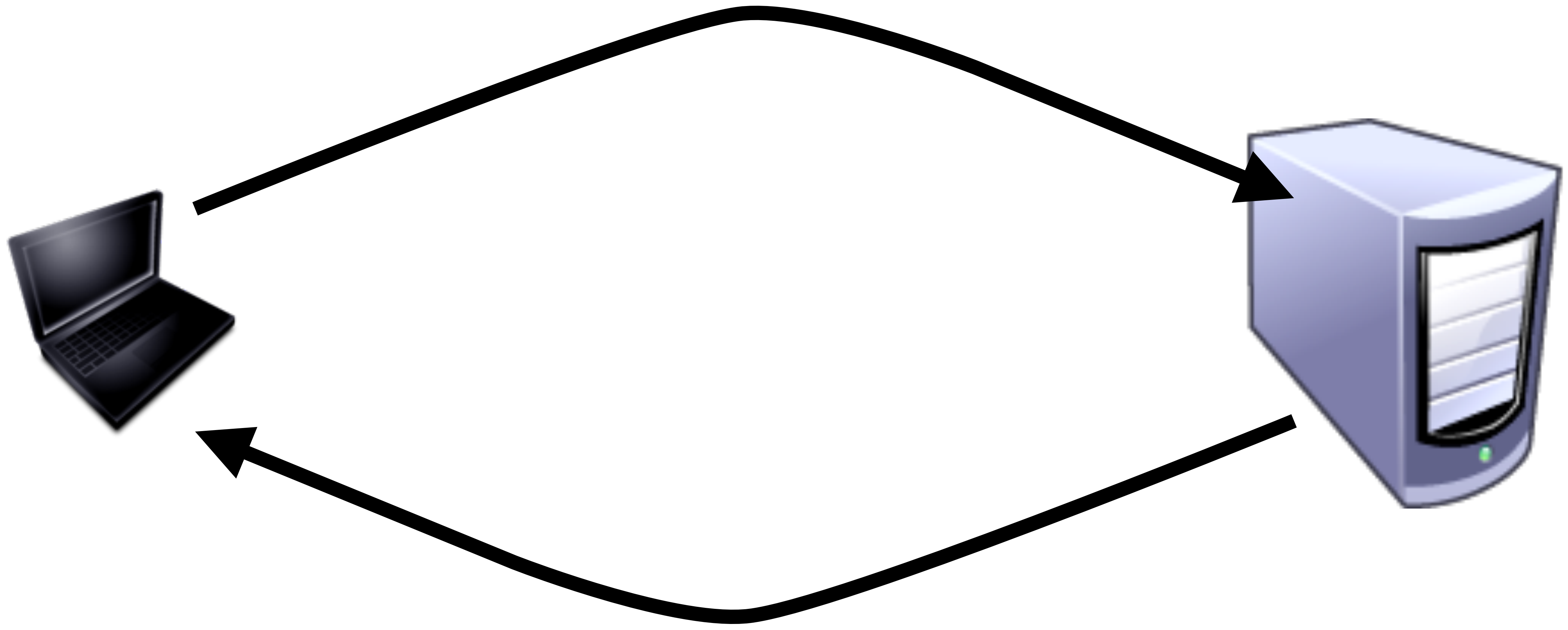
Accept: (by they way, I prefer that you answer me in Spanish)



Response: Son las once de la noche.

Request: /product/234

Accept: application/json



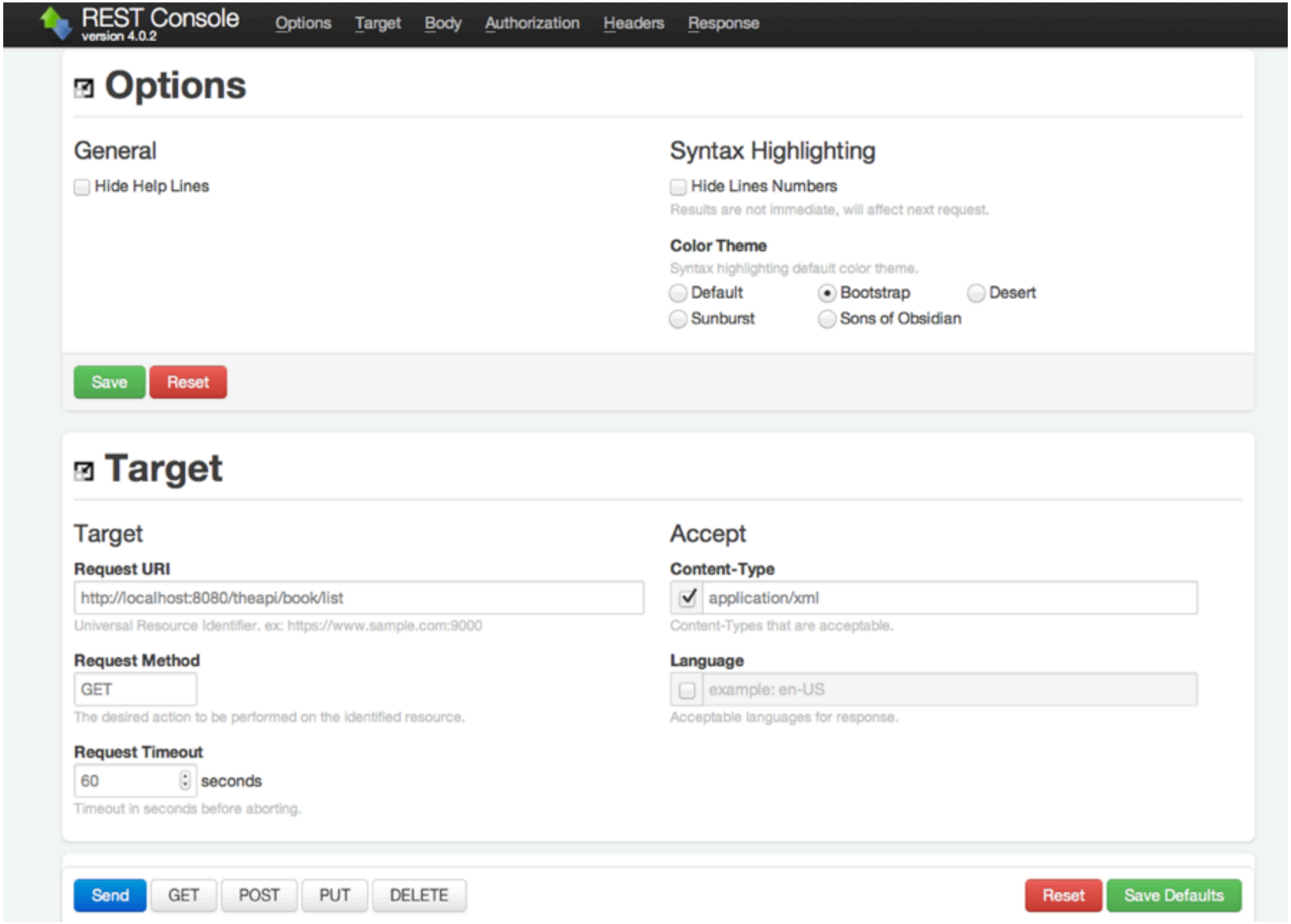
Response: { name: 'XBox 360', manufacturer: 'Microsoft' }

withFormat

- Grails is pre-configured to support common MIME types in grails-app/conf/Config.groovy
  - These can be customized
- The request object contains a format property that indicates which MIME type was requested
  - if (request.format == ‘xml’)...
- Preferably, use the **withFormat** method

Example

```
def list() {
    params.max = Math.min(params.max ?
        params.int('max') : 10, 100)
    Map model = [bookInstanceList: Book.list(params),
            bookInstanceTotal: Book.count()]
    withFormat {
        json {render(model as JSON)}
        xml {render(model as XML)}
        html { model}
    }
}
```



Options

General

Hide Help Lines

Syntax Highlighting

Hide Lines Numbers

Results are not immediate, will affect next request.

Color Theme

Syntax highlighting default color theme.

Default [✓ Bootstrap] Desert Sunburst Sons of Obsidian

Target

Target

Request URI: http://localhost:8080/theapi/book/list

Universal Resource Identifier. ex: https://www.sample.com:9000

Request Method: GET

The desired action to be performed on the identified resource.

Request Timeout: 60 seconds

Timeout in seconds before aborting.

Accept

Content-Type: ✓ application/xml

Content-Types that are acceptable.

Language: example: en-US

Acceptable languages for response.

REST console for Chrome
(http://restconsole.com)

Securing RESTful Services

- Basic Authentication (+ SSL)
  - sends username/password in clear text/ base64 encoded
- Private + Public key
  - http://www.thebuzzmedia.com/designing-a-secure-rest-api-without-oauth-authentication/
- OAuth
  - http://hueniverse.com/oauth/
  - http://oauth.net/