Algorithms and Programming IV

Web Dev: REST & WebSockets (24-3)

Summer Term 2021 | 05.07.2021
Barry Linnert (Michael Tebbe)
The “travel agent service” example
Recap: Simple Object Access Protocol (SOAP)

SOAP is designed to enable both client-server and asynchronous interaction over the Internet. It defines a scheme for using XML to represent the contents of request and reply messages as well as a scheme for the communication of documents.

It is used for information exchange and RPC, usually (but not necessarily) over HTTP.

(Very) basic SOAP architecture
REST stands for **Representational State Transfer** and it was invented by Roy Fielding in 2000.

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

REST is an architecture style for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines.

The World Wide Web itself, based on HTTP, can be viewed as a REST-based architecture.

(Elkstein, 2008)
REST Design Principles

1. Uniform interface
2. Client–server
3. Stateless
4. Cacheable
5. Layered system
6. Code on demand (optional)
The “travel agent service” example

Request

GET
/promotions/priceoffers/flights/ond/FRA/ROM?departureDate=2016-10-01&returnDate=2016-10-02&service=amadeusBestPrice

Server Response

<origin>FRA</origin>
<destination>ROM</destination>
<departureDate>2016-10-01</departureDate>
$returnDate>2016-10-02</returnDate>
<seatAvailability>7</seatAvailability>
<price>
  <amount>128.26</amount>
  <currency>EUR</currency>
</price>
<offerType>BestPrice</offerType>
<uuid>S003-b790715d-b2b8-4d23-ac27d4e88c4af</uuid>
REST Webservice with NodeJs and Express

Server

```javascript
var users = ['John Doe', 'Richard Roe'];

var express = require('express');
var app = express();

// respond with "Hello!" when a GET request is made to the homepage
app.get('/', function(request, response) {
  response.send('Hello!');
});

//Return the requested user
app.get('/user/:id', function(request, response) {
  response.send('user: ' + users[request.params.id])
});
```

Client

```html
<script>
const getUser = async (userId) => {
  //request data from Web Service with GET
  const response = await fetch('localhost:8080/user/?id=' + userId);

  //Print Response to Console
  console.log(response);

  //Print Response to Console
  console.log(response);
}
</script>
<button onclick="getUser(1)" >Print User 1</button>
```
Web Application Development I

WebSocket Protocol
Limitations of HTTP: Server Push

HTTP is unidirectional

- It follows a strict request-response scheme. The server cannot push data.
- A connection is only opened upon client request
- The connection is closed, once a server response has been sent (or after timeout)
- A new connection is opened for every request

**Long polling** has been developed as a workaround. Upon receiving a response, the client requests new data immediately. The server saves the request, and keeps the connection open, until new data is available. However, it is inefficient (many requests and connections, much overhead) and not robust (connection loss on time out).
WebSocket Protocol

A connection is established by Handshake.

It is kept open indefinitely or until one party closes the connection.

While open, both client and server can push data.
Request

GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBsb2NhdGlvbiBpdDE=
Origin: http://example.com

Server Response

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

=> Persistent connection is established; upgrade to WebSocket Protocol
WebSocket - Messages and Broadcasting

**Clients** can emit messages

**Server** can send message to one client or broadcast to multiple clients

Communication is event-based
Socket.io

NodeJS Module that enables bidirectional real time communication
Uses WebSocket, when possible.
const io = require('socket.io')(3000);

io.on('connect', socket => {

  // emit a message with custom name
  socket.emit('salutations', 'Hello!', { 'mr': 'john' }, Uint8Array.from([1, 2, 3, 4]));

  // handle the event sent with socket.emit()
  socket.on('greetings', (elem1, elem2, elem3) => {
    console.log(elem1, elem2, elem3);
  });
});