**PROTOCOLLO STOMP:**

* Protocollo testuale (simile a http)
* Supporta comandi da inviare ad un MessageBroker (simile a AMQP, JMS)
* In conseguenza di questi comandi un client puo’:
  + Inviare messaggi ad una MessageBroker ’’Destination’’
  + Eseguire una ’’Subscribe’’ verso una destination (abilitando in tal modo una ricezione event-driven di tutti i messaggi che verranno inviati su quella Destination)
* I messaggi (frame) inviati da un client STOMP hanno la seguente struttura:
  + Comando
  + Coppie di (key=val) headers (opzionali)
  + Body (opzionale)
* Stomp e’ un protocollo di livello application che si basa quindi su un protocollo piu a basso livello per gestire le comunicazioni (Es TCP)

**WEBSOCKET:**

* Protocollo bidirezionale per scambio di messaggi.
* E’ un piccolo wrapper intorno all ordinario TCP (su cui si basa anche HTTP a basso livello)
* Viene attivato su server che lo supportano con una prima richiesta HTTP da parte di un client che richiede un upgrade del protocollo verso websocket.

**WEBSOCKET + STOMP:**

* Una volta stabilita una connessione Websocket e’ necessario che client e server siano sincronizzati col tipo di messaggi che si scambieranno (app protocol)
* L’obiettivo principale dei websocket e’ abilitare la ricezione su client web (browseres) di eventi generati sul server.
* Per ottenere tale risultato una delle possibilita e’ utilizzare Websocket come protocollo di comunicazione e STOMP come protocollo applicazione.
* Tramite stomp un client (browser) puo’
  + spedire messaggi su Stomp Destinations,
  + Ricevere in automatico messaggi spediti su una Destination presso la quale ha eseguito (precedentemente) una subscribe.

**IN PRATICA:**  
WS funziona come il TCP e STOMP come protocollo di messaggistica (tipo AMQP)

**QUINDI:**

**Client Side:**

A livello WS ci saranno comandi per stabilire una connessione e per chiuderla.  
Eventualmente possibilita di svuotare il buffer.  
  
A livello Stomp bisognera’ gestire api per send e subscribe verso una Destination

**Server Side:**1) Endpoint per gestire messaggi che arrivano via websocket. (separati da quelli http ordinari)  
2) Handler per messaggi STOMP: che gestisce i diversi comandi e instrada i messaggi verso le specifiche Destination.

**EVENTI E TIMELINE:**

**WEBSOCKET:**

1. Browser tramite libreria js si collega ad un Endpoint ws ad un certo url http.
   1. E’ qui che avverra’ l upgrade a ws
   2. Apertura connessione:  
      Un socket tcp lato server verra’ collegato in modo permanente al client ws (tcp)
   3. Il client attraverso la connessione potra’ inviare dati (di qualsiasi tipo) all Endpoint

**STOMP:**

1. Una volta aperta la connessione WS il client potra’ inviare e ricevere dati
   1. Inviare dati (SEND) ad una DESTINATION
   2. Richiedere sottoscrizione (SUBSCRIBE) verso una DESTINATION
2. Client esegue una SUBSCRIBE verso una Destination
   1. Invia un messaggio STOMP SUBSCRIBE al WS Endpoint
   2. Il WS Endpoint inoltrera’ il messaggio al gestore (Handler) STOMP
   3. Il MessageHandler STOMP aggiungera’ il client alla collection di clients connessi a quella Destination
3. Client esegue un SEND
   1. Invia un messaggio STOMP SEND al WS Endpoint
   2. Il WS Endpoint inoltrera’ il messaggio al gestore (Handler) STOMP
   3. Il MessageHandler STOMP inoltrera’ via WS il messaggio a tutti i client collegati alla DESTINATION del messaggio.
   4. I clients collegati alla DESTINATION riceveranno il messaggio (sotto forma di evento per la lib usata)

**ALTRE POSSIBILI INTERAZIONI:**

1. Il Server riceve degli aggiornamenti da un sistema esterno (es Queue) o ha uno scheduler che periodicamente produce dei dati.
2. Inoltra i dati su una specifica Destination STOMP
3. Tutti i clients connessi a quella Destination ricevono i dati pubblicati (in b)

**WEBSOCKET E SPRING:**

**Basso livello:**

1. Creare una classe che implementa WebSocketHandler (ci sono classi per Text e Binary e una classe abstract per op comuni)
2. Registrare la classe 1) in un contesto incluso nella config della MessageDispatcher (se in SMVC)
3. Se non in SMVC necessario usare WebSocketHttpRequestHandler (caso particolare)

**Alto Livello:**

1. Necessario configurare un Endpoint dove indirizzare le richieste WS.
   1. Apertura/Chiusura connessione
   2. Dati che arrivano via WS
2. Necessario mappare uno STOMP Handler per ogni Destination

**IPOTESI TUTORIAL:**

1. Websocket server e client (no Stomp)
   1. Intro Spring messaging
      1. Channel
      2. Message
   2. Scheduler eventi server che spedisce String al client
   3. Client SockJs
2. Stomp senza Websocket
   1. RabbitMq o ActiveMQ (Stomp service)
   2. Client SockJs

**Channels:**

****

****

**Component**

****