Real-time Web

How to push information from server to client?

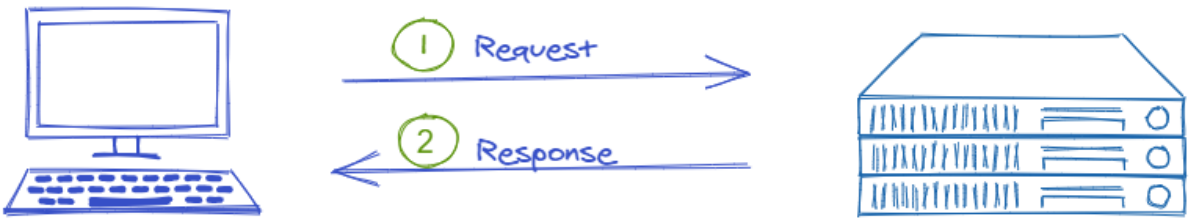
Slides are based on scripts from Prof. Grüneis

Introduction

- Observer-Pattern used in Web-Applications
- Server sends information to the client
- Various options:

    1. Polling
    2. Long Polling
    3. Server Sent Events
    4. WebSockets
    5. SignalR

Traditional Request-Response-Scenario



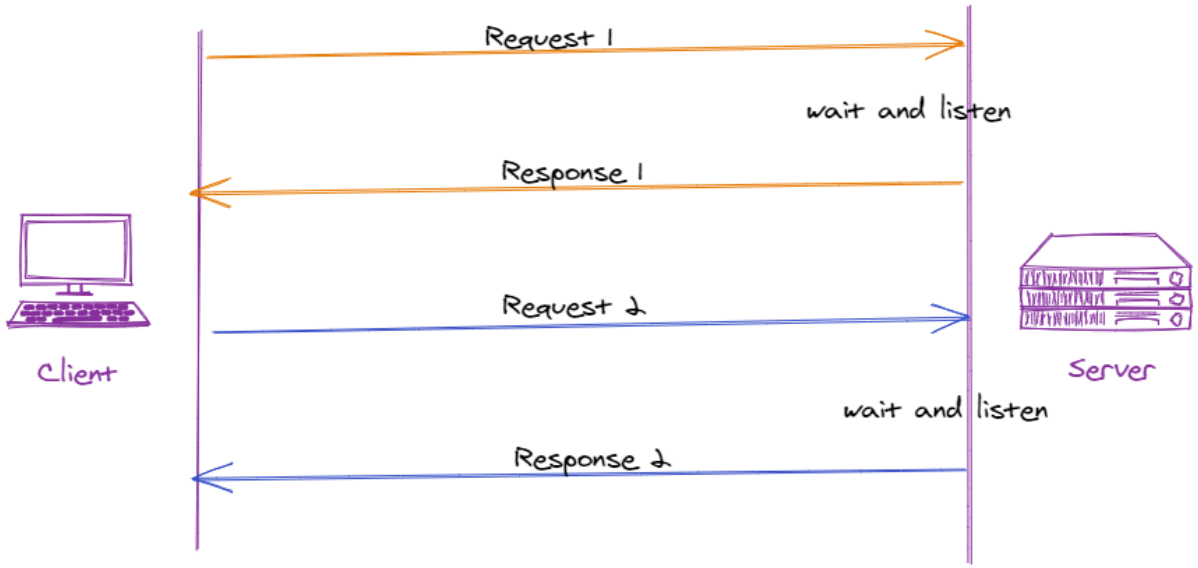
- Client requests information from server
- Server sends reponse to client

Source: https://medium.com/geekculture/understand-whats-behind-real-time-web-apps-e60168129480

Polling

- Client periodically asks the server (using a GET request) if there is an update
- Responses:
  - Status **200** + updated information
  - Status **204** (no content)
- Cons:
  - Many (most of the time useless) HTTP request
  - Response is delayed and not in real-time

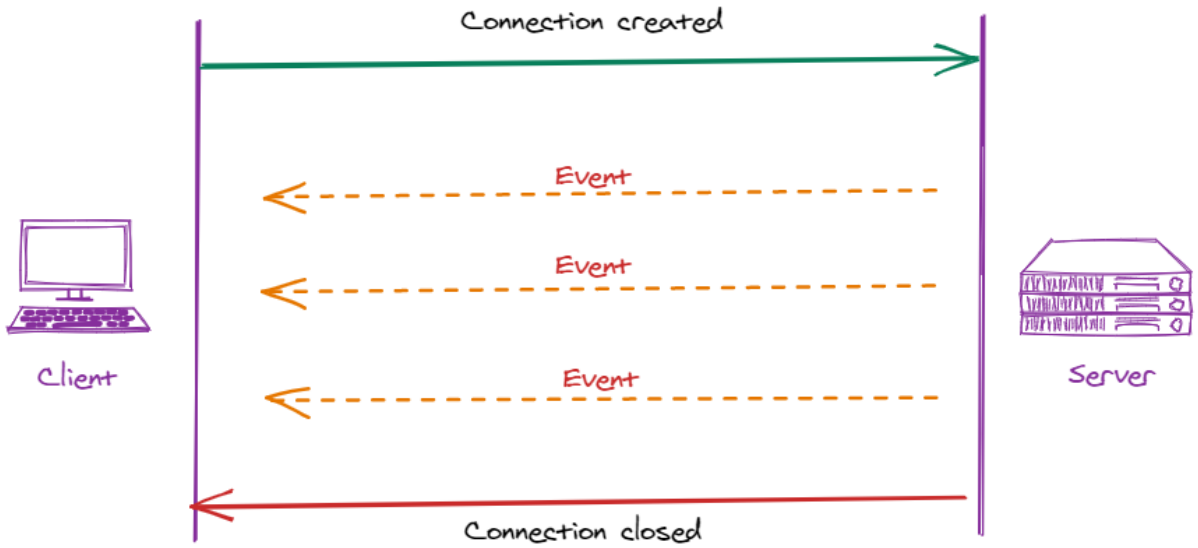
Long Polling



- Client send GET request to server, canceles if after some time and repeats request
- Server does not answer immediatelly but response as soon as ...
  1. ... there is an update availably on the server.
  2. ... the connection was closed.
- Pros (compare to polling):
  - Fewer HTTP requests
  - Response will be sent immediatelly from the server if update is available

Source: https://medium.com/geekculture/understand-whats-behind-real-time-web-apps-e60168129480

Server Sent Events



- Is an **HTML 5 feature**
- Server creates an HTTP connection with the client (content-type `"text/event-stream"`)
- Client actively listens to the connection as a stream until connection is closed
- Client opens `EventSource` on URL of the service
- `onmessage` event ... data is received from the server (other events: `onopen`, `onerror`)

Source: https://medium.com/geekculture/understand-whats-behind-real-time-web-apps-e60168129480

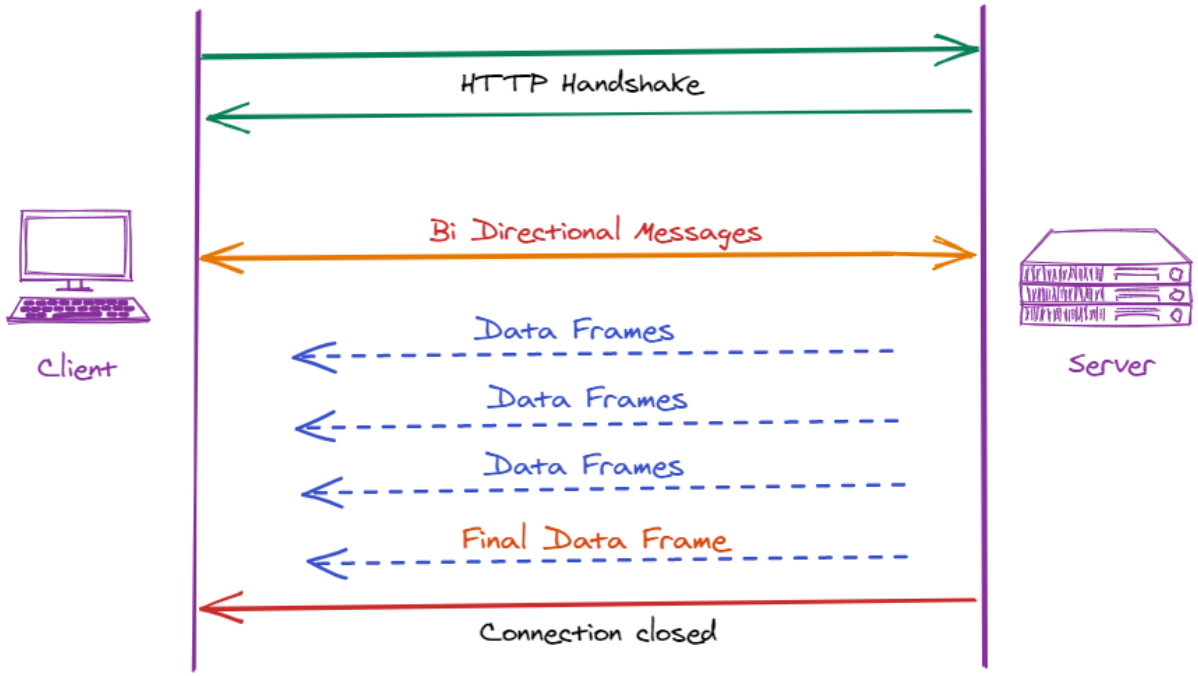
Pros and Cons of Server Sent Events

- Server sent events are Simple HTTP requests.
- Older browsers do not support server-sent events
- It can auto reconnects once the connection is dropped
- A maximum number of 6 HTTP connections can be created using server-side events
- Server-side events only support text messages. It does not support binary data therefore passing video and audios are not possible with server-side events.

- The connection is one way.

Source: https://medium.com/geekculture/understand-whats-behind-real-time-web-apps-e60168129480

WebSockets



- Use a TCP socket.
- Full duplex (client -> server, server -> client)
- Text and binary data supported
- Handshake mechanism: upgrade HTTP -> TCP
- If server accepts HTTP request from client (101 ... `"Switching Protocols"`), upgrade to WebSockets protocol
- Up to 50 connections between one client and server possible.

Source: https://medium.com/geekculture/understand-whats-behind-real-time-web-apps-e60168129480

SignalR

- Is an cross platform, fast and lightweight Open Source framework to combine and simplify the usage of the technologies described before.
- Client needs own JavaScript library
- Implements a fallback mechanism
  - Tries to establish WebSockets connection
  - If it fails, tries to establish a Server Sent Events connection
  - If it fails, tries long polling
- Used technology is hidden from the client and server

SignalR (cont.)

- Implements **Remote Procedure Calls (RPC)** in **both directions**
  - Server can invoke method on client
  - Client can invoke method on server

SignalR - Hub

- Is a component on the server.
- Implements the RPC functionality.
- Enables method invocations on the client and client can invoke methods on the server using the Hub.
- Protocol used: JSON (binary alternative: MessagePack)
- Implements the Observer-Pattern (clients connect to Hub; Hub can inform clients of new connection, new messages received etc.)

SignalR - Cookbook: ASP.NET 6 Server

- No special NuGet Packages required for SignalR (provided in **`Microsoft.AspNetCore.SignalR`**)
- Create new Hub

```
public class DiceHub : Hub
{
    public override Task OnConnectedAsync() ...
    public override Task OnDisconnectedAsync(Exception exception) ...
}
```

SignalR - Cookbook: ASP.NET 6 Server (cont.)

- Provide public methods which should be available to be called by clients and, if required, inform other attached clients:

```
public class DiceHub: Hub
{
    public void NewDice(DiceDto dice)
    {
        // perform some business logic
        Clients.All.SendAsync("diced", dice);
    }
    ...
}
```

- **Important:** Clients need to have a *`diced()`* method implemented which can be invoked from the server (see later)

SignalR - Cookbook: ASP.NET 6 Server (cont.)

- Register Hub as a service (singleton) in *`Program.cs`* (to enable dependency injection in e.g. a controller)
- Hub can request e.g. a database service via DI
- [Authorize] works as in controllers
- You can define client groups (using Clients.Groups)

SignalR - Cookbook: ASP.NET 6 Server (cont.)

- Enable SignalR in *Program.cs* (order of method invocations is important!)

```
...
string corsKey = "_myCorsKey";

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy(corsKey,
        x => x.SetIsOriginAllowed(_ => true)
            //.AllowAnyOrigin()
            .AllowAnyMethod()
            .AllowAnyHeader()
            .AllowCredentials()
    );
});
...
```

SignalR - Cookbook: ASP.NET 6 Server (cont.)

```
...
// Add services to the container
...
builder.Services.AddSingleton<StockHub>();
builder.Services.AddSignalR();
builder.Services.AddMvc(options => options.EnableEndpointRouting = false);

builder.Services.AddControllers();
...
var app = builder.Build();
...
app.UseCors(corsKey);
// app.UseHttpsRedirection();
app.UseRouting();
app.UseAuthorization();
app.UseEndpoints(endpoints => endpoints.MapHub<DiceHub>("/dice"));
// app.UseEndpoints(endpoints => endpoints.MapHub<DiceHub>("/dice",
```

```
//    options => options.Transport = HttpTransportType.LongPolling
//                                 | HttpTransportType.ServerSentEvents ));
app.UseMvc();
app.MapControllers();
app.Run();
```

SignalR - Cookbook: Angular Client

- Install required Node package

  ```
  npm i @microsoft/signalr
  ```

- Initialize and attach to server

  ```
  import { HubConnection, HubConnectionBuilder,
    HubConnectionState } from '@microsoft/signalr';
  ...
  private hubConnection!: HubConnection;
  ...
  ngOnInit(): void {
    this.hubConnection = new HubConnectionBuilder()
        .withUrl('http://localhost:5000/dice'
         /* , HttpTransportType.WebSockets |
          ↪  HttpTransportType.LongPolling) */
        .build();

    this.hubConnection.on('diced',
      (dice: Dice) => this.messages.push(`${dice.name} ${dice.count}`)
    );

    this.hubConnection
      .start()
      .then(() => this.messages.push('*** connection established'))
      .catch(err this.messages.push('*** error while establishing
      ↪  connection'));
  ```

SignalR - Cookbook: Angular Client (cont.)

- Check connection

```
public get isConnected(): boolean {
  return this.hubConnection.state == HubConnectionState.Connected;
}
```

- Invoke method on server

```
public sendDice(): void
{
  if (!this.isConnected) {
    this.messages.push('*** not connected');
    return;
  }
  this.hubConnection
    .invoke('newDice', {
      name: this.nickname,
      count: Math.floor(Math.random() * 6 + 1)
    })
    .catch(err => console.log(err));
}
```