Web Development and API Design

Lesson 10:
Online Multi-Player Game

Prof. Andrea Arcuri

Goals

- See a full working example of a non-trivial, multi-player, online game based on what learned so far in class

- Learn how to deploy your app on a cloud provider

- Some discussions on databases…

Online Game

Multi-player Connect4

- Turn-based, but need *WebSockets* to get informed when opponent has done his/her move

- How to pair players on different machines?

- How to prevent cheating?

- etc.

- REST API call to start match, same endpoint (IP:port)
- Put on queue until at least 2 players

POST /api/match

POST /api/match
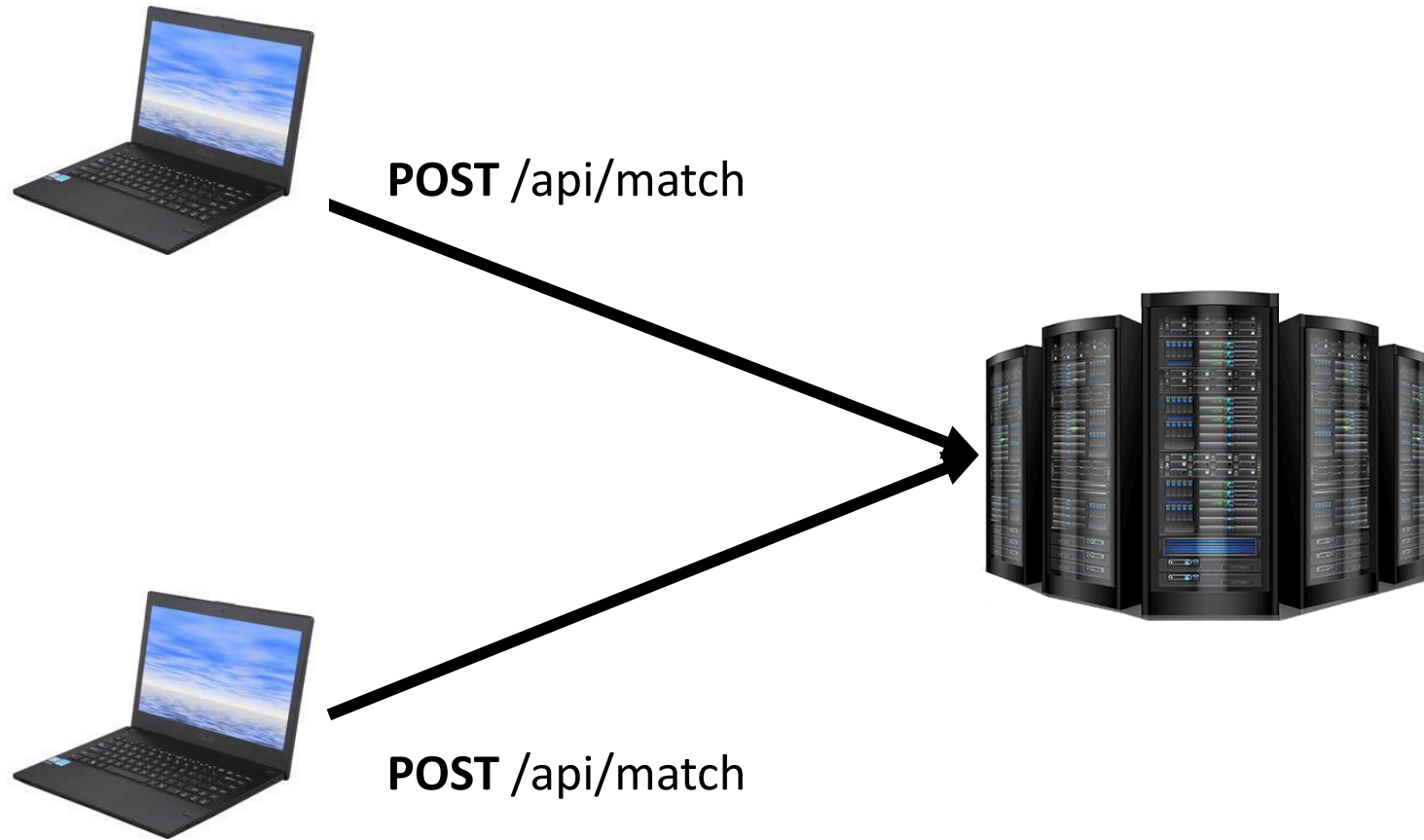
- When match starts, **each player** creates a *WebSocket* toward the server
- To avoid cheating, the **state** of the match is on server, the players only see a **view** of it (recall users can do whatever on his/her browser…)
- Match state: **board** plus 2 *WebSockets* for the 2 players
- At each state change on server, opponent gets informed via *WebSocket*
- Actions out of sequence must be discarded (possible cheater)
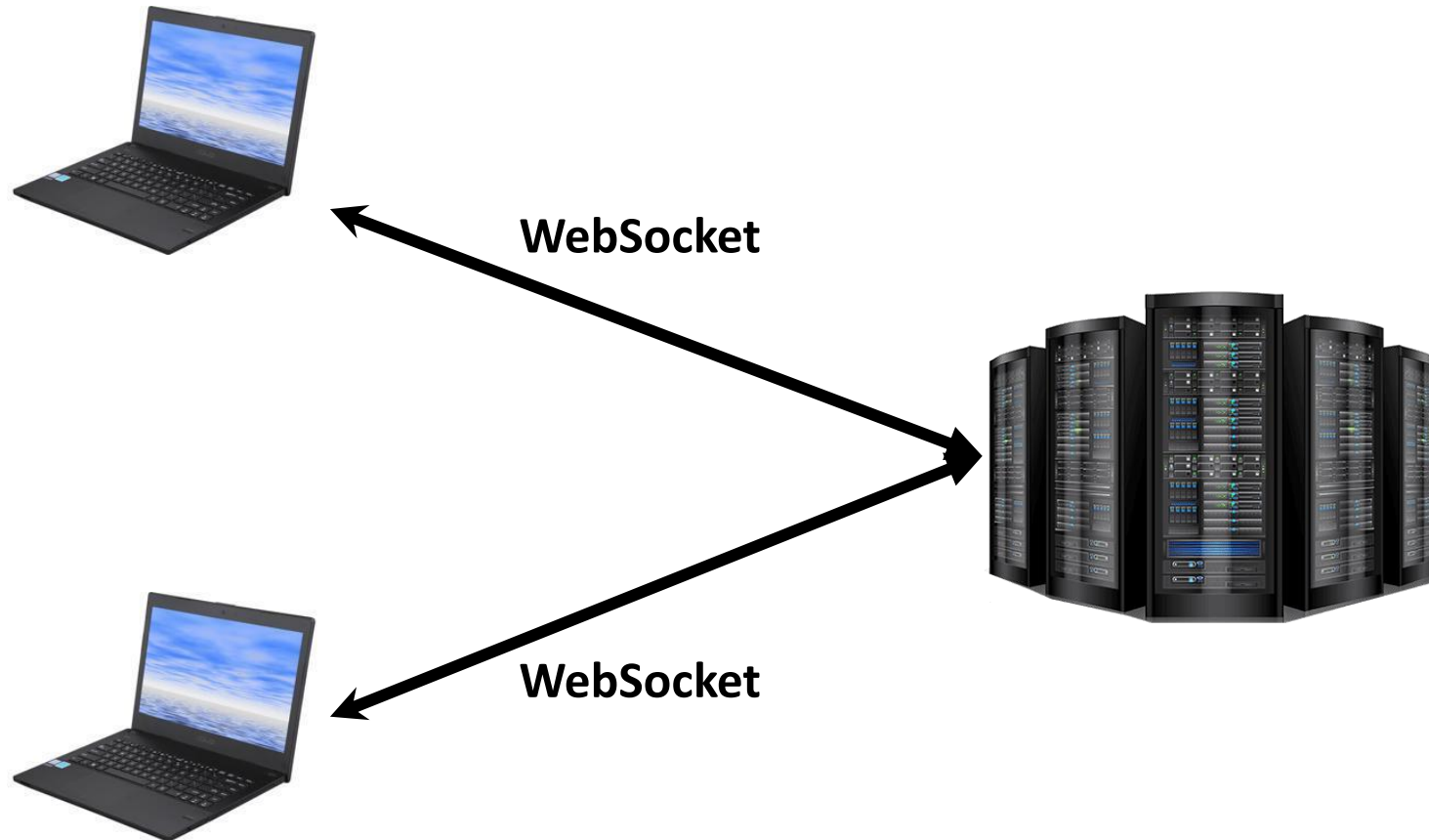
WebSocket Security

- Assume match with id **X** between players **A** and **B**
- How does server know that incoming *WebSocket* request for player **A** on match **X** is actually made by player **A**???
- At least 2 options
  1. As first WS message is in HTTP, can use same auth
  2. Create a **unique, one-time-use random token** in a **secured** way for **A** (eg authed REST endpoint), associate such token to **A** on server, auth the WS when such token is provided
     - This is a more general approach, not tied to implementation details of how the WS is established
     - Recall that WS is for a single user (own TCP socket), so need auth only once, and not like on every single request like in HTTP (eg using cookies)

WebSocket Cost

- WSs are **EXPENSIVE**, as TCP sockets are OS resources
- Need to keep open a TCP socket **for each** WS
- In HTTP, if running out of ports, can close TCP after resolving each incoming request
  - Recall a TCP connection is defined by 4 coordinates: IP and port of client, and IP and port of server
    - So, there can be at most $2^{16}=65,536$ TCP connections at same time on server
  - Ie, in HTTP keeping a TCP on after a request is only for performance reasons, eg if expecting other following requests
    - See also: "*Connection: Keep-Alive*" HTTP header

WebSocket Topics

- There can be different types of communications between a client and server over WS
  - eg many different functionalities and operations
- Too expensive to open a different TCP socket for each operation
- **Must re-use same WS** for each different kind of operation
- (Simple) Solution: wrap JSON data into an object defining a discriminating "*topic*" field, which identifies the operation
  - Note: there are more sophisticated message protocols like STOMP
- Server will decide what to execute based on topic's field value

Example

```
{
  "topic": "…"
  "data": {…}
}
```

```
{
  "topic": "update"
  "data": {
        "matchId": 42,
        "boardDto": {…},
        "isX": true,
        "opponentId": 1234
  }
}
```

Cloud Deployment

Cloud Deployment

- Different companies provide cloud hosting solutions for your applications, which frees you from hardware issues, but for a price

- *Amazon Web Services* (AWS) is perhaps the most famous/used one
  - eg, *Netflix* runs on AWS

- *Automated scaling*: if you need more load, automatically rent more nodes, and automatically scale down if less load
  - this is also good for applications targeting a specific country (eg Norway), in which you will not get much load during the night

Definition of "Cloud"

There is no cloud
it's just someone else's computer

Heroku

- One of the main cloud providers
- At the time of this writing, it provides *easy* to use *free* hosting
  - note, this might change at any time
- Supporting NodeJS applications
  - and many others

Using Heroku

- First you need to create an account at www.heroku.com
- Most instructions on Heroku shows how to deploy with Git, but *I do not like it*… we will use a CLI
  - however, you can use whatever you like…
- Install *Heroku CLI*, which allows you to interact with Heroku from command line
- On the web interface, create an "app" with a name of your choice. In these slides, I will use "*pg6300-c4*"
  - as names are unique, you will need to choose a different name
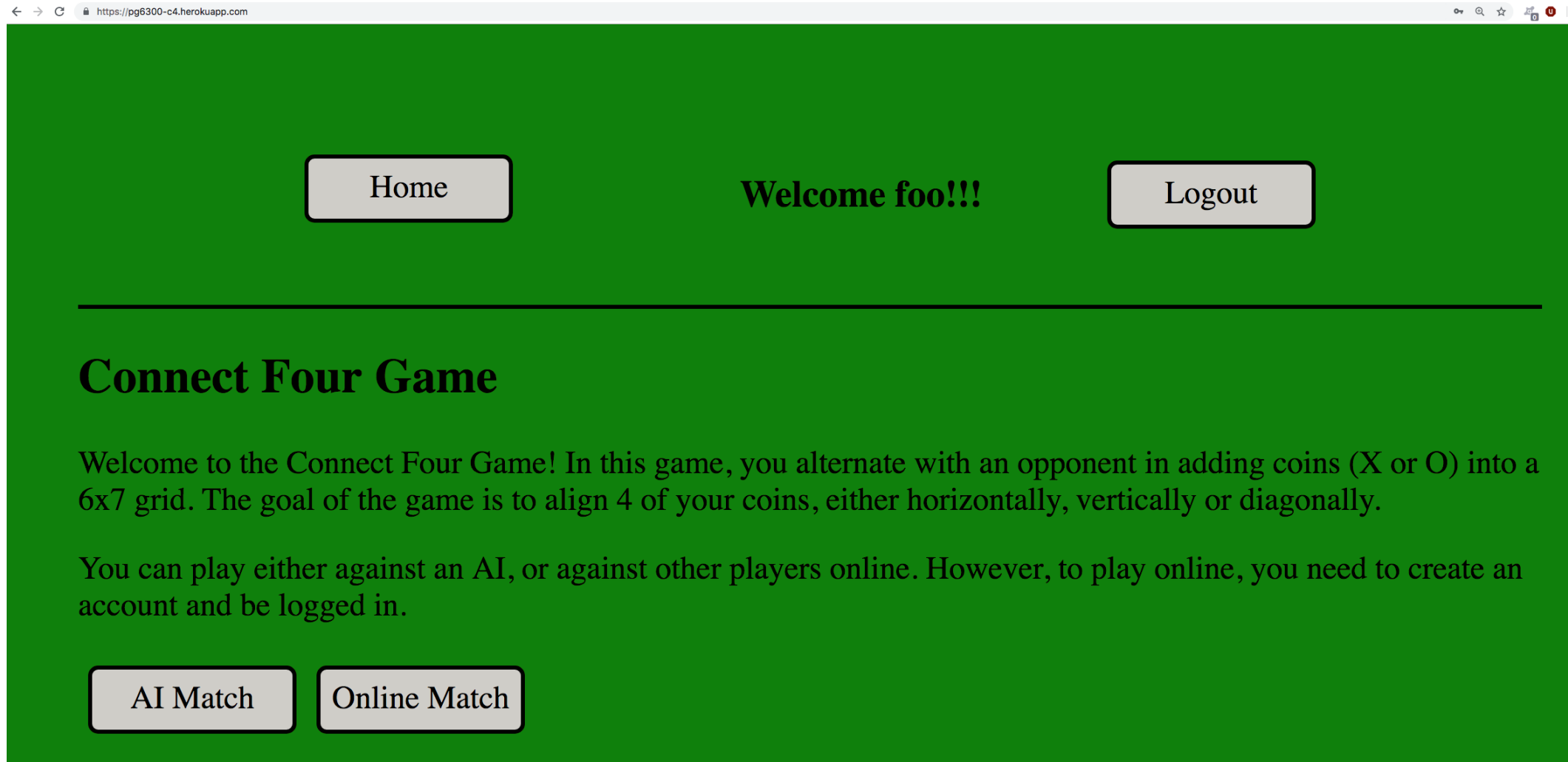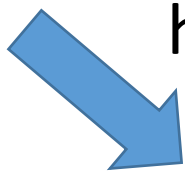
From Command Line (CLI)

- **heroku plugins:install heroku-builds**
  - need to be run only once, to install the "*builds*" plugin
- **heroku login**
  - will setup credential for the other commands.
  - note: if using Windows, this might not work on GitBash, and need to do this command once from a regular Terminal
- **heroku builds:create -a pg6300-c4**
  - zip all your files in current folder, and deploy them in the app
  - note: use "*.gitignore*" to specify what to exclude

Settings

- Can have extra settings in "*engines*" under *package.json*, but not compulsory
  - eg., version of Node and YARN to use to run/build your app
- Your app MUST bind to a port specified by **process.env.PORT**
  - otherwise, Heroku will not know how to reverse-proxy to it
- Your app will be automatically built by Heroku with "*yarn build*", and started with "*yarn start*"

https://pg6300-c4.herokuapp.com/

https://pg6300-c4.herokuapp.com

Home   **Welcome foo!!!**   Logout

Connect Four Game

Welcome to the Connect Four Game! In this game, you alternate with an opponent in adding coins (X or O) into a 6x7 grid. The goal of the game is to align 4 of your coins, either horizontally, vertically or diagonally.

You can play either against an AI, or against other players online. However, to play online, you need to create an account and be logged in.

AI Match   Online Match

Cloud and WebSockets

- WebSockets are expensive resources
- Free-tier cloud options might allow only small number of WS
- Might shut them down automatically if inactive for even short period, e.g. 30-60 seconds
  - could implement some auto-reconnect when sockets are forcibly closed…

Databases

Why Databases?

- Need to store your data somewhere

- If in memory, lose everything as soon as the app restarts

- For a full app, you need a database
  - cloud providers like Heroku give you databases as well

- We are not going to see databases in this course, but need to briefly discuss them

The 3 Rules of Choosing a Database

1. New project or unsure what to do? **Choose Postgres**

2. If you are already using *MySQL* and migration to *Postgres* would be too expensive, can stick with *MySQL*

3. If you have a long experience with databases, know exactly what you are doing, and can measure objectively the performance benefits of different tradeoffs compared to just using *Postgres*, then, *and only then*, choose best database for the *specific* problem you are facing

Example: *MySQL*

- Open source, but own (and mainly developed) by *Oracle*... and let's not forget that one of its main commercial products is *Oracle Database*...
  - so, yes, in theory those 2 databases are competitors...
- For most use cases, *MySQL* is on par with *Postgres,* but usually slower at adding new advanced features
  - eg support for NoSQL features like JSON data type, or SQL compliance

Example: *MongoDB*

- Most famous *NoSQL* database
  - very, very popular in tutorials… especially in NodeJS
- Meant for *documents*, not for data with *relations*
  - Usually documents are in JSON format, where the only relations are hierarchical, eg nested objects
- Can be *fast* and *easy* to set up…
- … but you need to *sacrifice ACID* for it…
  - eg, when you "save" some data, can be just cached, and not actually saved…
  - ACID transactions added in v4.0, in 2018…
- *Postgres/MySQL* can save JSON fields, and be very fast at it
  - eg, in 2014, Postgres was actually faster than MongoDB in benchmarks at dealing with JSON

MongoDB Cont.

- Might start with JSON *documents*… but then one day you need to add relations between data: *you are screwed*
  - "screwed" meaning ending up implementing JOINs at application level, which is a nightmare and very inefficient… and/or duplicate data, which need to be kept always in sync…

- Or even worse, choosing *MongoDB* even when you deal with relational data, just because of *hype*…

- …or when you do not really deal with the amount of data of Google/Amazon/etc…

But... MongoDB is "Web Scale"!

- https://www.youtube.com/watch?v=b2F-DItXtZs

- http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/

- **echo "MongoDB is Web Scale!" > /dev/null**

- Note: video is from 2010. At that time MongoDB was total *"rubbish"*. Today is better
  - eg ACID transactions added in 2018