

# 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...

# 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.
  - details will be in the code

# 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”



# 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](http://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



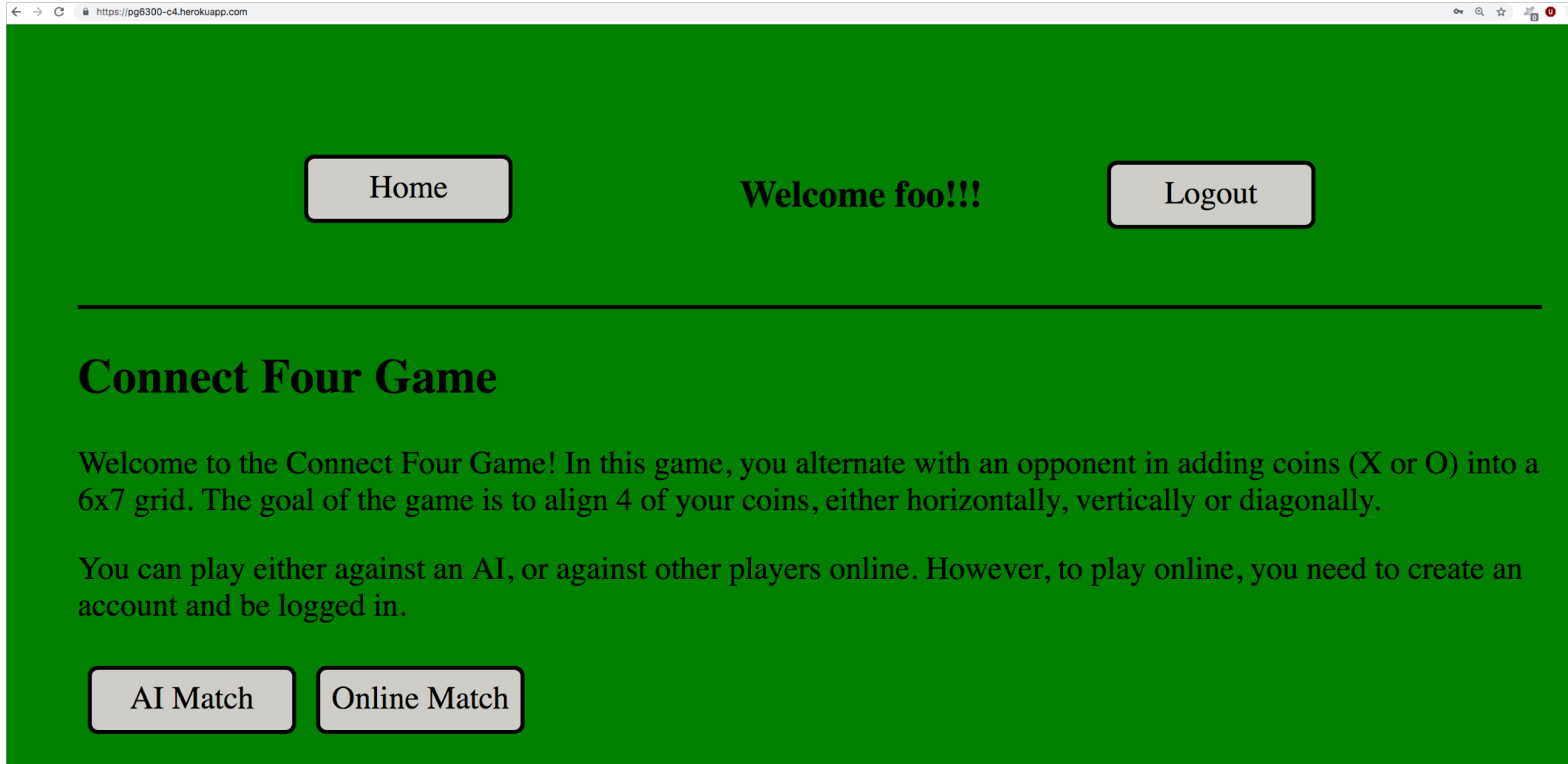
# From Command Line (CLI)

- **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/>



# 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-DltXtZs>
- <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 “rubbish”. Today is better
  - eg ACID transactions added in 2018

