CSC309 Winter 2016 Lecture 6

Larry Zhang

Admin

- A1 marking will be done soon (probably tomorrow)
- You can submit remarking requests by Feb 15
    - Read TA’s annotations first, then clearly describe why your marking need to be changed.

A2 Notes

- Use PostgreSQL on the CS.UTM server, we will do the marking there. (CSLINUX and DHxxxxPCxx may not work)
- Functionality first, appearance second. Try the artistic things only if you have time for it.
- Basic word cloud can be generated using some JavaScript generated CSS applied to HTML (don't need to be images).
- Important requirement: the main page must be AJAX based (never refreshes).
- Twitter API rate limit is typically 180 requests per 15 minutes, but check the documentation to be sure.
- There is also a Twitter API call that tells how close you are to the rate limit.
- Convenient way to get profile pictures: http://lorempixel.com/

MVC (Model-View-Controller)

Model-View-Controller

- MVC is a software architectural patterns mostly used for applications with **graphical user interfaces**.
- Popular for web applications.
- Invented by Trygve Reenskaug in 1970s.
- Well known frameworks using MVC: Ruby on Rails, Django

Design goals

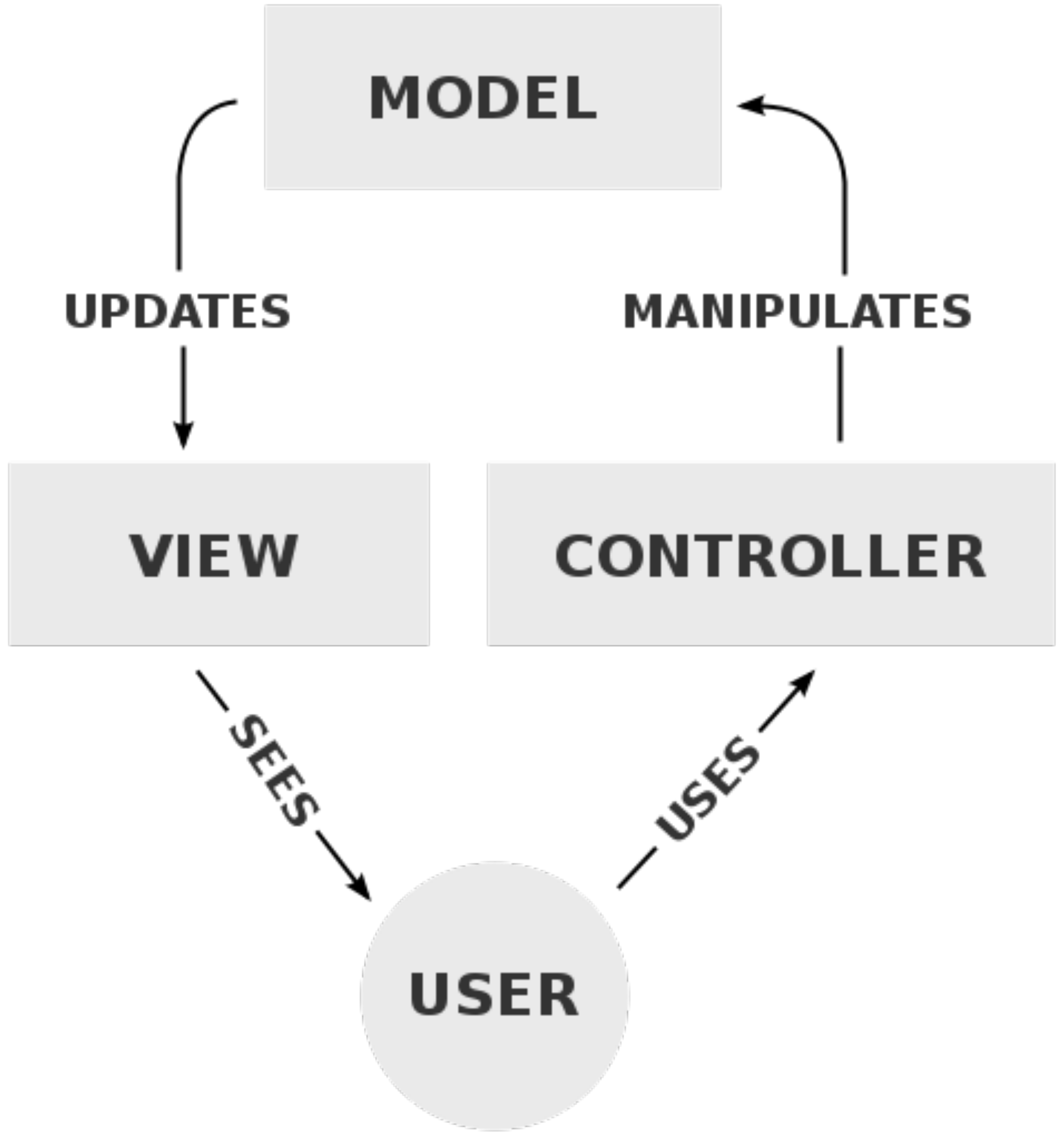
- Separate the core functionality from the presentation and control logic that uses this functionality.
- Allow multiple views to share the same data model
- Make the app easier the implement, test and maintain.

MVC definitions

- **Model**: the core functionalities, used to store and manipulate data, typically stored in database, session, etc.

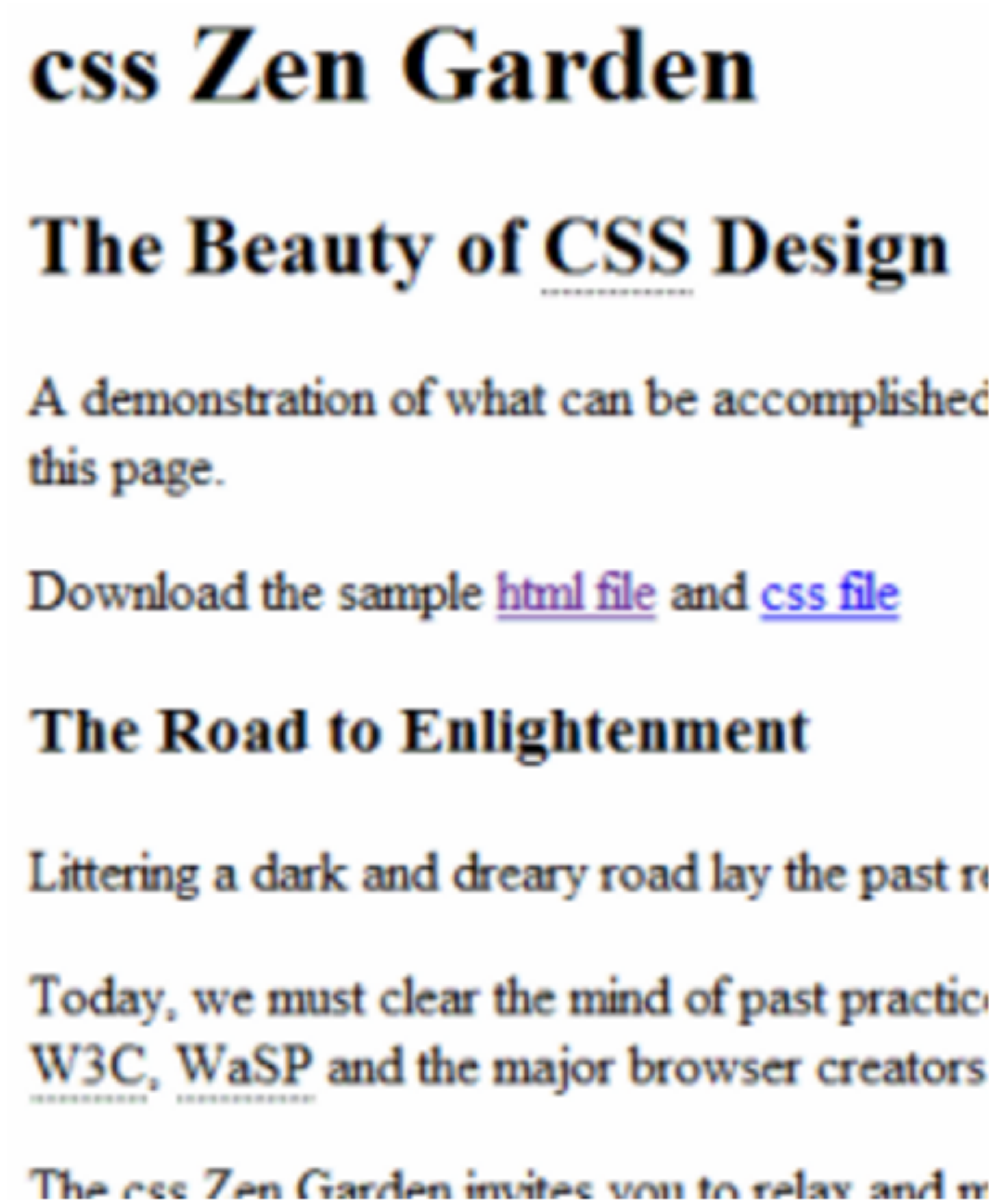
- **View**: The user interface, a rendered version of the model.

- **Controller**: The brain of the application. Change the state of the applications according to user input and current state. Basically, a **finite state machine**.



An easy example of MVC

Model = HTML



css Zen Garden

The Beauty of CSS Design

A demonstration of what can be accomplished
this page.

Download the sample html file and css file

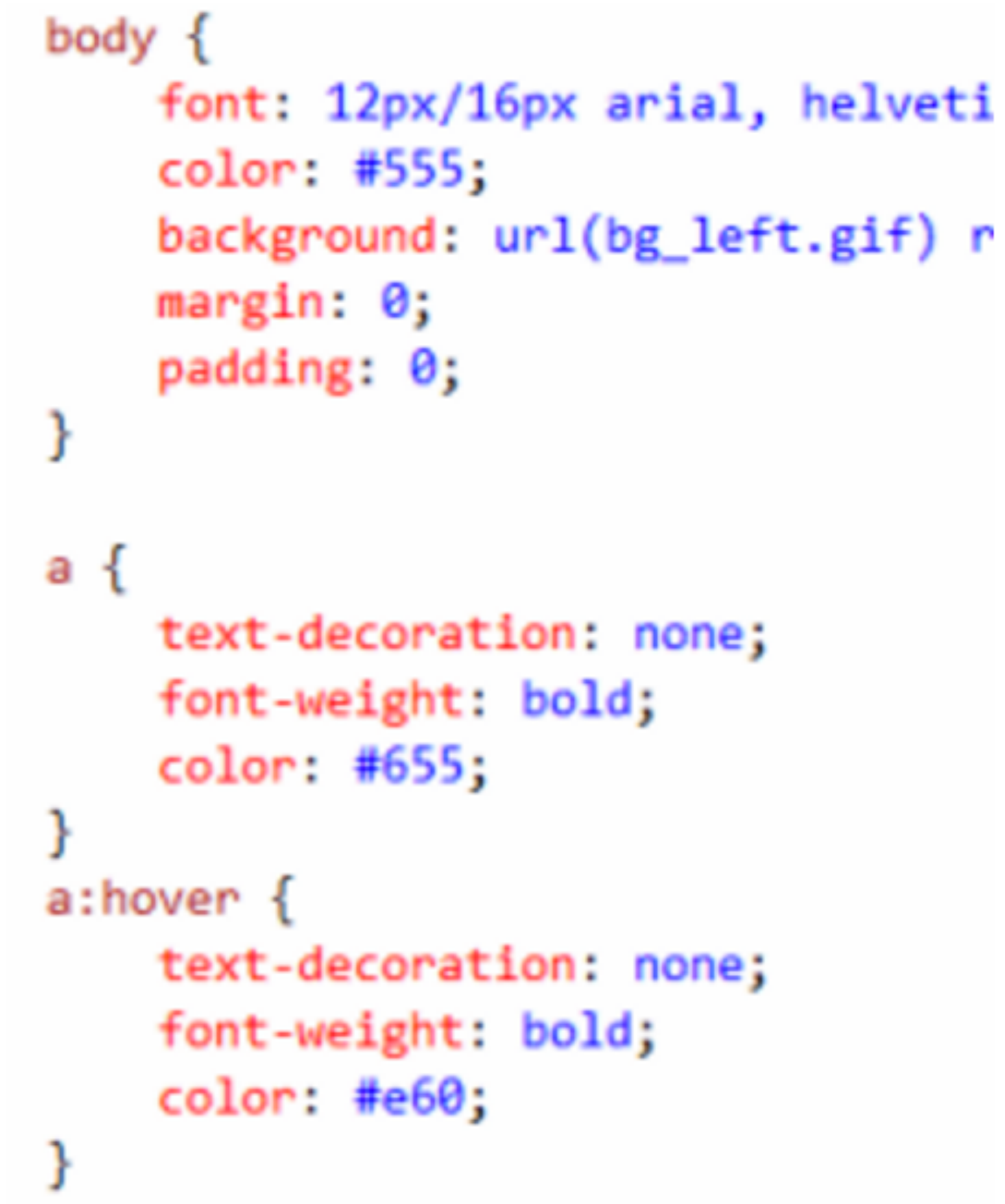
The Road to Enlightenment

Littering a dark and dreary road lay the past r

Today, we must clear the mind of past practic
W3C, WaSP and the major browser creators

The css Zen Garden invites you to relax and m

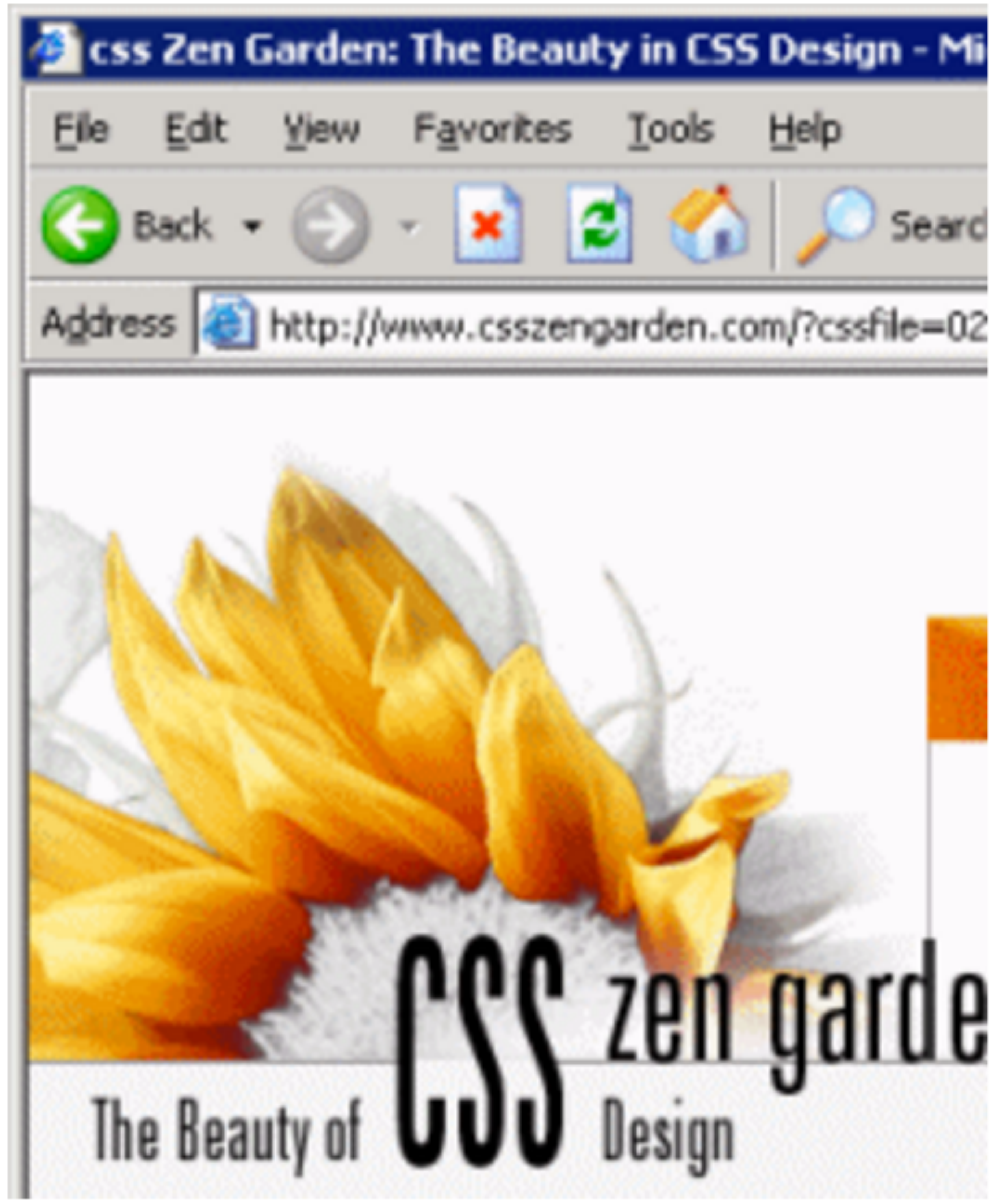
View = CSS



```
body {
     font: 12px/16px arial, helveti
     color: #555;
     background: url(bg_left.gif) r
     margin: 0;
     padding: 0;
}

a {
     text-decoration: none;
     font-weight: bold;
     color: #655;
}
a:hover {
     text-decoration: none;
     font-weight: bold;
     color: #e60;
}
```

Controller = Browser


css Zen Garden: The Beauty in CSS Design - Mi
File
Edit
View
Favorites
Tools
Help
Back
Searc
Address
http://www.csszengarden.com/?cssfile=02
The Beauty of CSS zen garde Design

A web app example: Guess Game

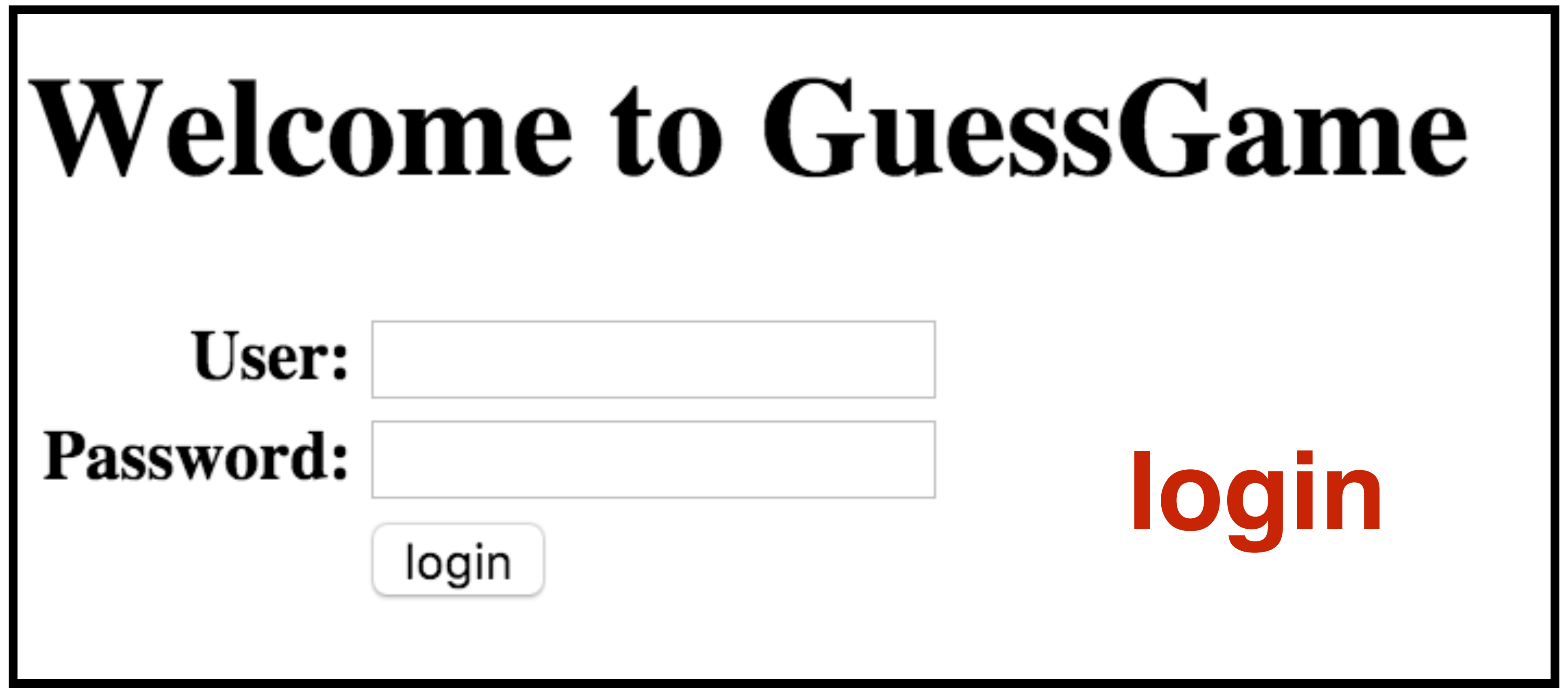
- demo
  - https://cs.utm.utoronto.ca/~zhangy33/guessGameMVC/

Guess Game MVC: Model

Core functionality (data API)

- Start Game:
  - randomly generate a secret number
  - number of guesses = 0
  - history = []
- Make a guess:
  - check guessed number against the secret number, produce response message (too high or too low, or correct)
  - number of guess ++
  - add guess to history

Guess Game MVC: Views

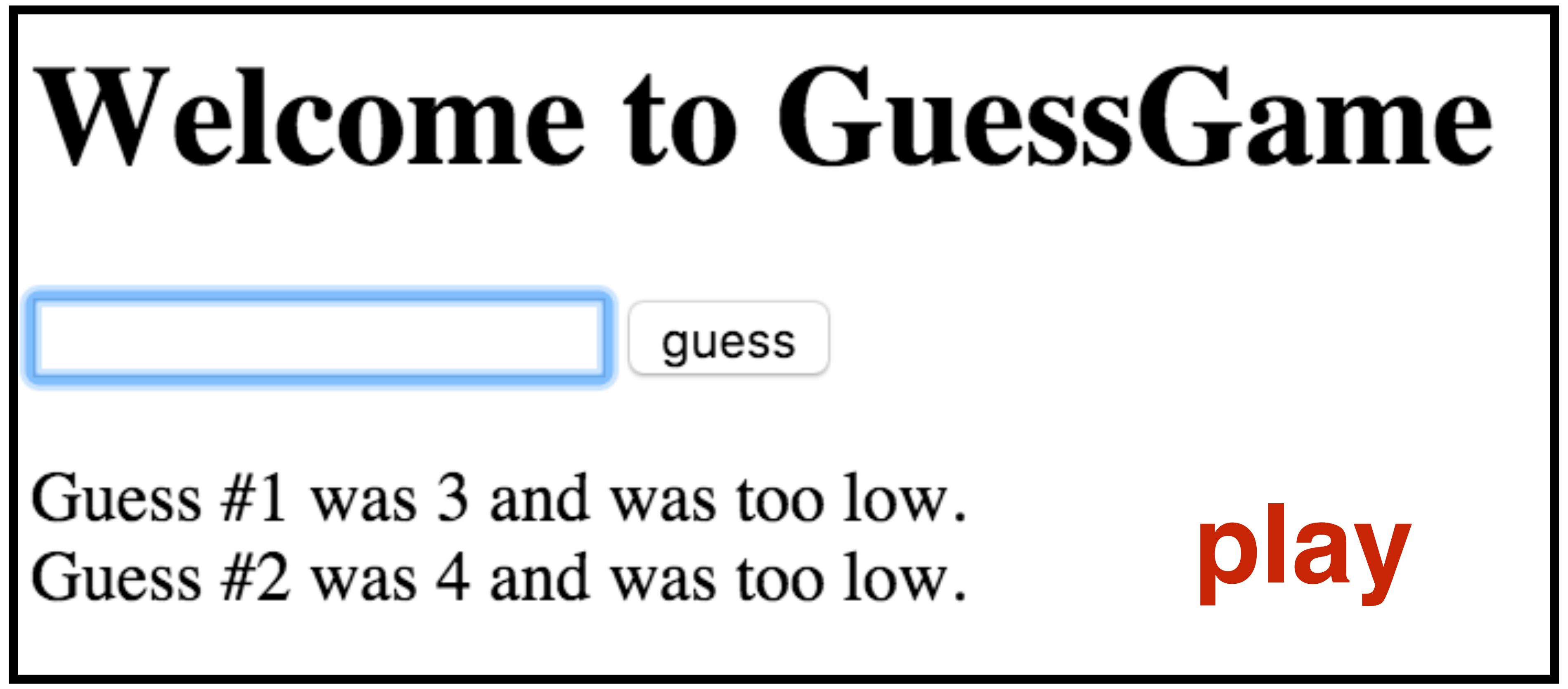


Welcome to GuessGame

User:

Password:

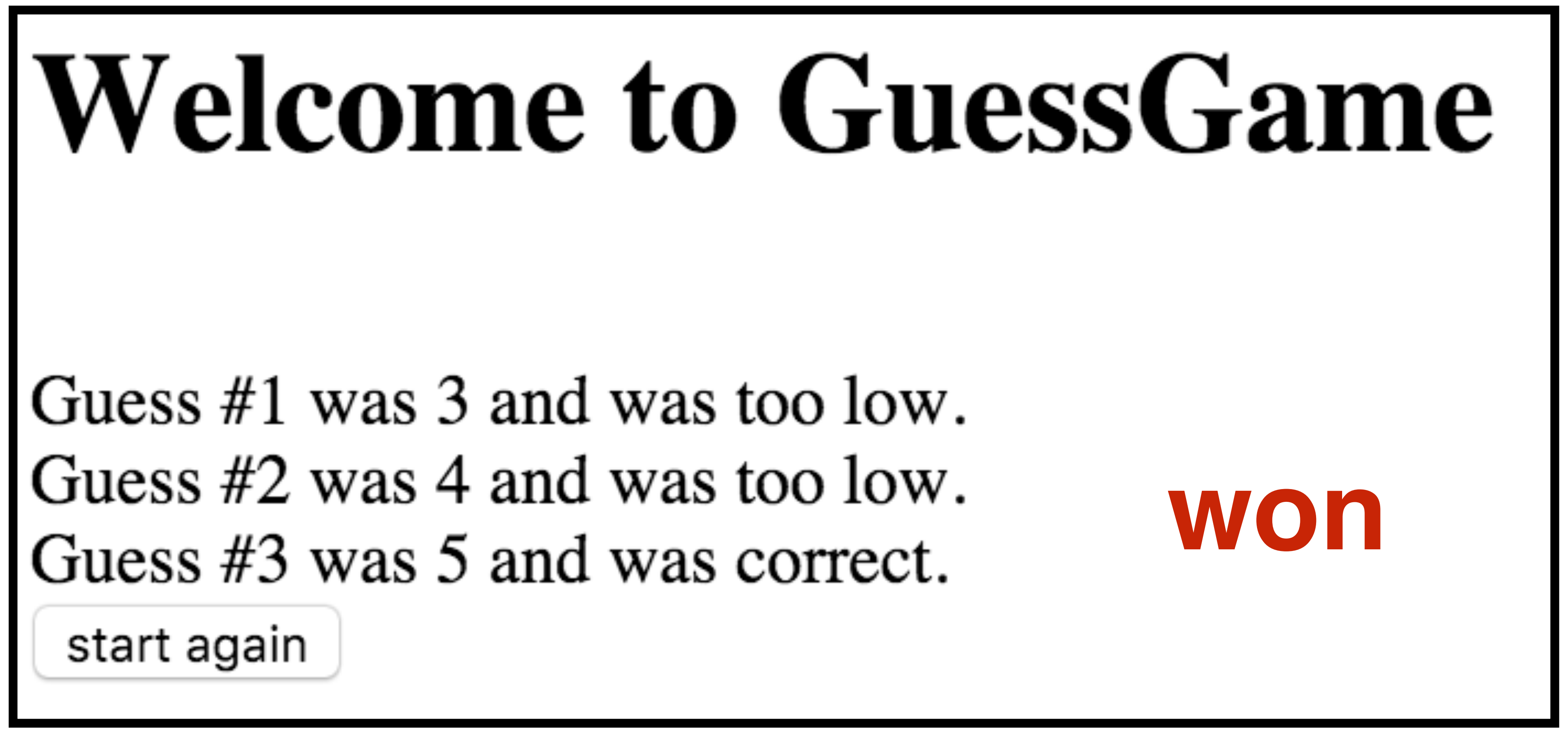
login



Welcome to GuessGame

Guess #1 was 3 and was too low.
Guess #2 was 4 and was too low.

play

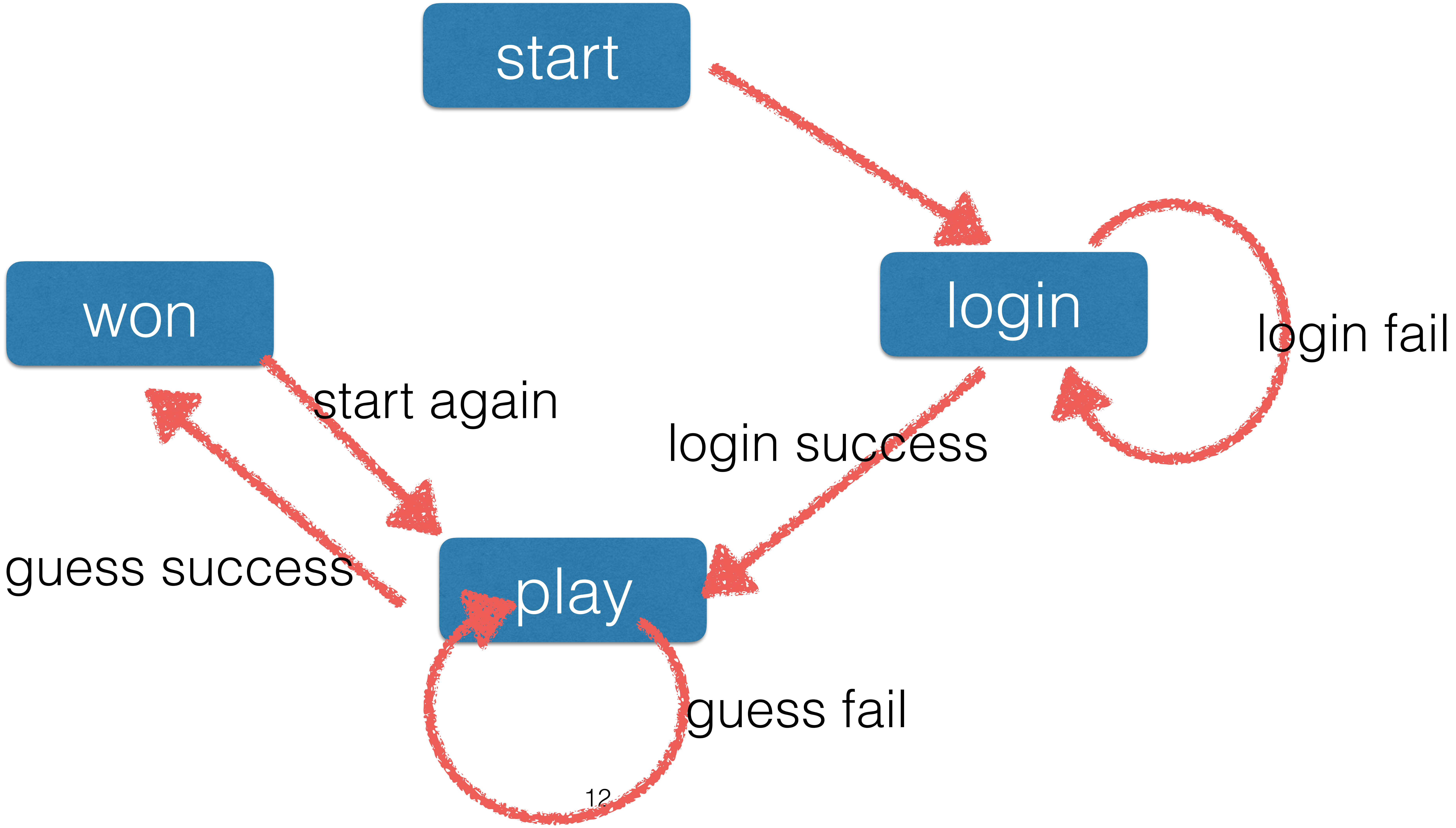


Welcome to GuessGame

Guess #1 was 3 and was too low.
Guess #2 was 4 and was too low.
Guess #3 was 5 and was correct.

won

Guess Game MVC: Controller



Guess Game MVC

- code

Summary

- The **model** code provides the data API, i.e., how to update and get the data for the application
- The **view** code defines the appearance of each page, and it **only** defines the appearance and has nothing to do with the functionalities
- The **controller** code (index.php) runs a finite state machine, which switches between application states, access core functionalities to access data, and update the views of the application.

MVC advantages

- Clarity of design: model provides API to data access; view and controller easy to write
- Efficient modularity: different modules can be worked on by different developers; each module can easily be replaced
- Easy support of multiple views, each of which uses the same data API provided by model
- Easy to maintain
- Easily distributable to different machines

Other patterns also exist

- Hierarchical model–view–controller
- Model–view–adapter
- Model–view–presenter
- Model–view–viewmodel
- Naked objects
- Observer pattern
- Presentation–abstraction–control

REST API and RESTful Web Services

Representational **S**tate **T**ransfer

- People don’t really agree on what REST really means
- Common perception: it is just a convention about how to use HTTP
- Use URLs to access **resource**.
- If we following this convention, it will enables us to build high-performing and more maintainable web apps

REST

- Instead of having randomly named setter and getter URLs and using GET for all the getters and POST for all the setters, we try to have the URLs identify resources, and then use the HTTP actions GET, POST, PUT and DELETE to do stuff to them.
- So instead of
    - **GET /get_article?id=1 POST /delete_article?id=1**
- You would do
    - **GET /articles/1/**
    - **DELETE /articles/1/**

We access resources

- Two basic types of URLs
    - One for a collection of the resource
        - /dogs
    - One for a particular resource
        - /dogs/42

REST Request Types

- **GET**: Read a specific resource (by an identifier) or a collection of resources.
- **PUT**: Update a specific resource (by an identifier) or a collection of resources. Can also be used to create a specific resource if the resource identifier is know beforehand.
- **DELETE**: Remove/delete a specific resource by an identifier.
- **POST**: Create a new resource. Also a catch-all verb for operations that don't fit into the other categories.

REST is Stateless

- No client context being stored on the server between requests
- What you get from an API call depends only on information provided in the current request.
- no sessions
- e.g., /dogs?color=white&location=toronto

Why use REST?

- Everyone else is using it.
- A REST service is essentially a lightweight web service, suitable for access by application clients (like a smartphone app)
- You need to use if you want developers to access your web service’s data in a programmed way.
- Easy to scale and maintain.

Example: Twitter API

- GET statuses/home_timeline
- POST statuses/retweet/:id
- POST friendships/destroy
- GET search/tweets.json?q=uoft

Twitter API

First, **OAuth**

The authentication used by Twitter as well as most of the service providers

Why OAuth

- When Alice wants a third-party app (e.g., App309, a registered Twitter app) to access your Twitter account, and be able to post message to your Twitter account.
- The naive and insecure way: Alice gives away her Twitter username and password to App309.
- With OAuth, Alice can allow App309 to access her Twitter account without telling App309 her Twitter username and password.
- It takes a few steps to setup …

How OAuth works

- Step 1: user shows intent
    - Alice: Hey App309, I would like you to access my Twitter account.
    - App309: Great! Let me ask Twitter for permission.

How OAuth works

- Step 2: App309 asks permission
  - App309: Hi Twitter, I have a user who wants give me access to her Twitter account. Could you give me a **request token**?
  - Twitter: Okay, here is a token, it needs to be **authorized** by the user. Also, here is a **secret** code, use it as signature next time you use this token so we can verify that it is actually you using the token.

How OAuth works

- Step 3: App309 redirects Alice to Twitter
    - App309: Hey Alice, I’m sending you over to Twitter so you can tell Twitter you authorize my access. Take this **request token** with you.
    - Alice: OK!

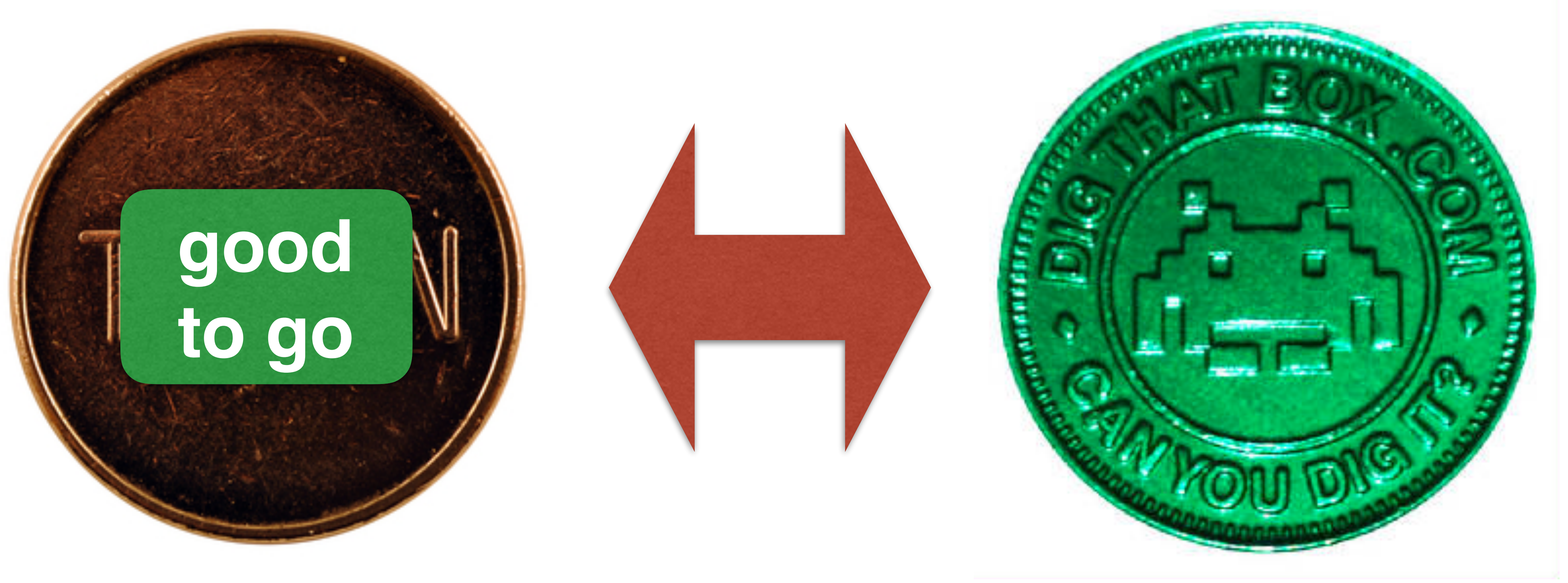
How OAuth works

- Step 4: Alice logs in to Twitter talks to Twitter
  - Alice: Hey Twitter, I’d like to authorize this **request token** that App309 gave me.
  - Twitter: OK. Just to be sure, you want App309 to do X, Y, and Z with your Twitter account?
  - Alice: Yes!
  - Twitter: OK, the **request token** is authorized. You can go back to App309 and let them know they can use the request token now.

How OAuth works

- Step 5: Alice go back to App309
    - Alice: OK, App309, the **request token** is authorized.
    - App309: Great! I will use it to get an **access token** from Twitter!

How OAuth works



- Step 6: App309 goes to Twitter again
  - App309: Hi Twitter, I’ve got this authorized request token, can I exchange it for an **access token**?
  - Twitter: Checking…, OK, here is your **access token** and **secret**, use the access token when you want to access Alice’s Twitter account, and sign your request with the secret, so we know it is you.
  - App309: No prob!

How OAuth works

- Step 7: App309 accesses Alice’s Twitter account
    - App309: Hi Twitter, I’d like to post a message to Alice’s timeline. Here is my **access token**, the request is signed by the secret.
    - Twitter: Checking…, OK. Done!

Summary

- Basically, when App309 wants access Twitter it needs to provide the following **4 strings**
    - CONSUMER_KEY and CONSUMER_SECRET: App309’s ID which was created when the app is registered on Twitter.
    - ACCESS_TOKEN and ACCESS_TOKEN_SECRET: proves App309’s permission to access Alice’s account
- The authentication process is simpler if App309 is accessing Twitter on its own behalf.
    - The access token and secret for App309 is created directly in App309’s developer console, without having to go back and forth some many times.

Twitter API demo