CS253 Web Application Engineering

Unit 1

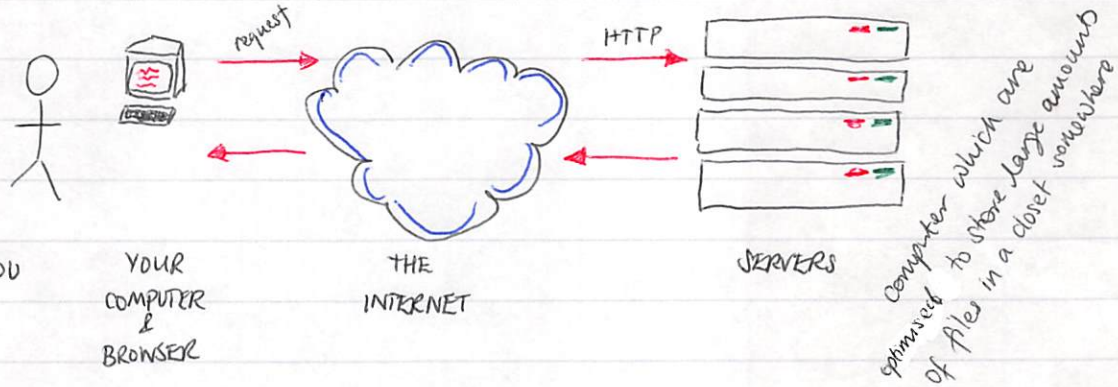
1.2 The basics of the web:
- HTML (main doc type)
- URL (reference to doc on web)
- HTTP (protocal)
- Web applications

1.3 The world wide web is made up of a collection of htmle documents

HTML = Hypertext Markup Language

Hyper-links to other documents make the web what we recognise it as
Invented in 1990s it's made up of ≈ 30bn pages

1.4



You make requests through your browser via the internet using a protocal called http.
The server responds with files that your browser displays

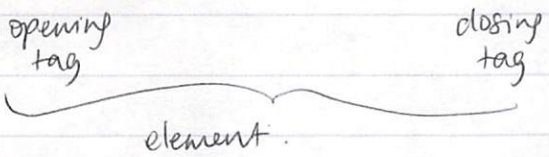
1.6 Intro to HTML HyperText Markup Language

HTML is made up of
- text content (what you see)
- markup (what it looks like)
- references to other documents (e.g. images, videos etc.)
- links to other pages

www.udacity.com/html_playground

1.7 HTML markup

(<NAME> Content </NAME>



1.8 Bold Tag

 Content b - bold

1.9 Italics

em - emphasis - italics

 Content

1.10 missing end tag (default puts it at the end)

1.11 Making links - HTML Attributes

<TAG ATTR="value"> contents </TAG>

Anchor - <a> - for making links

 derp

href: attribute name
"www.reddit.com": value

1.12 Images - <img> - for inserting an image

<img src="url" alt="text">

VOID TAG
no closing tag
- no content for an image so no closing tag required

alt:
- required
- text that gets displayed if the content isn't found

* images just appear inline in text

1.13 Whitespace

[text on
multiple
lines] → [text on multiple lines]

Break -
 - for newline VOID TAG

1.14 Paragraph -

 contents

1.15 Inline vs Block - so far everything we've looked at apart from is inline

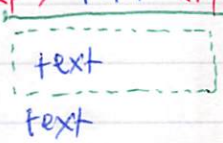
 - inline

text
 text

text \n
text

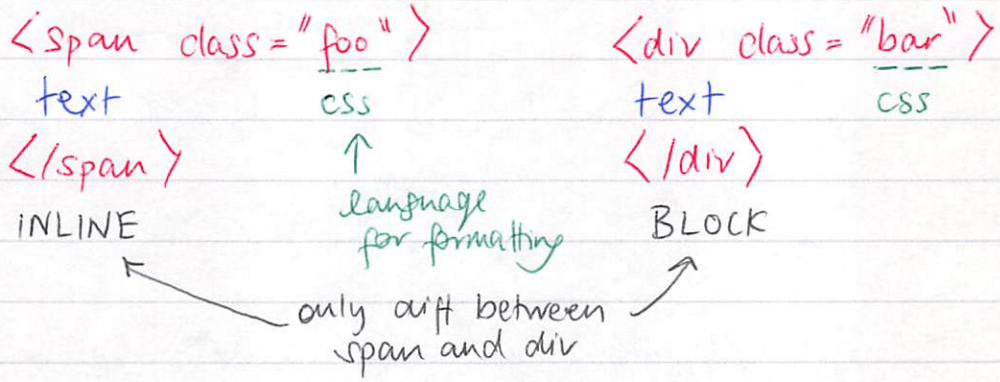
 - block

 text text

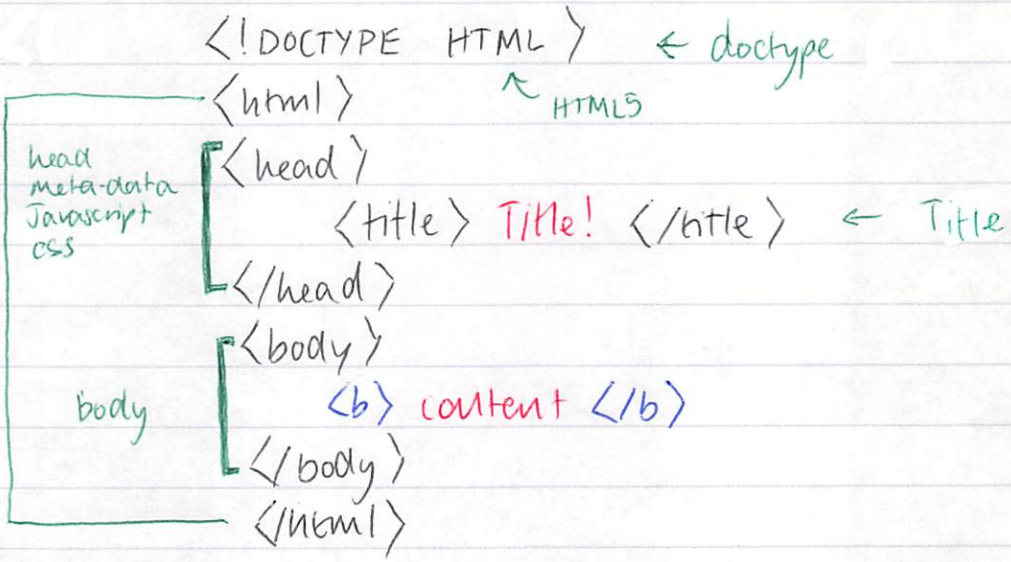


text

1.16 Span and Div



1.17 Documents structure



1.18 Introducing URL

URL = Uniform Resource Locator

http://www.udacity.com/cs253x/wipmunk.png

protocol | host | path

1.20 Query parameters / GET parameters

http://example.com/foo?p=1&q=neat

? first sep separator
& separators thereafter

NAME = VALUE

1.21 Fragments

http://www.example.com/foo #fragment

separator

http://www.example.com/foo?p=1 #fragment

* fragments are not sent to the server
* used to reference a part of the page you are looking at
* other uses in javascript

1.22 Ports

http://localhost:8000/path

port default = 80

port 8000 will be used for the web development

1.23 HTTP - HyperText Transfer Protocal

http://www.example.com/foo

GET /foo HTTP/1.1 - request line - the text is sent over the internet to the server

METHOD PATH VERSION

1.1/1.0

GET

POST

Note: The hostname is used for making the connection while the path is used to make the request

1.25 HTTP Request

GET /foo?p=1 HTTP/1.1 – request line is followed by a number of headers

Headers:

Name: value ← FORMAT

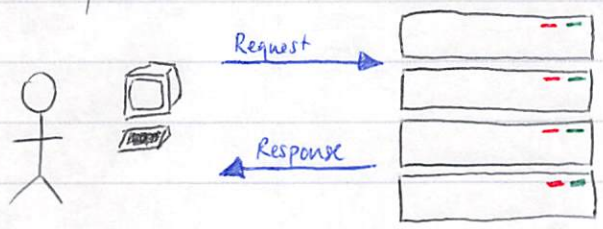
← COMMON TYPES

- each server hosts many websites so it needs to know which one you want
- who is making the request i.e. browser and version

Host: www.example.com
User-Agent: chrome v.17

No spaces allowed in the Name

1.28 HTTP Response



Status line – HTTP/1.1 200 OK

status code / reason phrase

Status codes:

200	OK	2--	sucess	find the document
302	Found	3--	need to do something different to ^	
404	Not found	4--	error on browser/client side	
500	Server error	5--	error on server/host side	

1.29 HTTP Responses also have headers

Date: Tue Mar 2012 04:33:33 GMT
Server: Apache/2.2.3 ← don't use this – info for hackers
Content-type: text/html
Content-length: 1539

1:30 $ telnet www.udacity.com 80

Trying 74.125.127.121...

Connected to ghs.l.google.com.

Escape character is '^]'

GET / HTTP/1.0

Host: www.udacity.com

← default behaviour in HTTP/1.1 is to not close the connection

HTTP/1.0 200 OK ← status line

Cache-Control: no-store

content-type: text/html ; charset = utf-8

Set-Cookie: ...

Vary: ...

Date: Wed, 21 Mar 2012 01:25:43 GMT

Server: ...

Expires: Wed, 21 Mar 2012 01:25:43 GMT

} headers

<!DOCTYPE html>

<html>

<head>

<title>Udacity - Educating the 21st Century</title>

:

</head>

<body>

:

</body>

</html>

} html do

Connection closed by foreign host.

1.31 Web Applications

Server: purpose = to respond to http requests

Requests / Response have two types

① Static:
- → pre-written file
- → image

② Dynamic
- → made on the fly by programs called... WEB APPLICATIONS

Unit 2

2.1 Forms

write a text file called "play.html" and open it in your browser

HTML Forms

<form>
</form>

<input name = "name">
<input type = "button_name">

<input name = "query_param_name">
creates a text box in the
writing text "query_param_value" in the box
and entering it updates the url with the
query parameter and its value, i.e. ?name=value

<input type = "button_name">
creates a button in the

clicking the button has the same effect as pressing enter – it updates the url with the query parameter value

```
<form action="/foo">
    <input name="query_param_name">
    <input type="query_param_submit_button_name">
</form>
```

2.8 Google App Engine Hello, world!

```
import webapp2

class MainPage(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/plain'
        self.response.out.write('Hello, webapp World!')

app = webapp2.WSGIApplication([('/', MainPage)],
                              debug=True)
```

MainPage inherits from webapp2.RequestHandler

method of the class

by default this is 'text/html'

global response object

url mapping section

url maps to a handler called

```
import webapp2

form = """
<form method="post" action="/testform">
    <input name="q">
    <input type="submit">
</form>
"""
```

method is "get" by default

action is go to the path "/testform"

```
class MainPage(webapp2.RequestHandler):
    def get(self):
        #self.response.headers['Content-Type'] = 'text/plain'
        self.response.out.write(form)

class TestHandler(webapp2.RequestHandler):
    def post(self):
        # q = self.request.get("q")
        # self.response.write(q)

        self.response.headers['Content-Type'] = 'text/plain'
        self.response.out.write(self.request)

app = webapp2.WSGIApplication([('/', MainPage),
                               ('/testform', TestHandler)],
                              debug=True)
```

this means it will read the form as plain text instead of html

set header 'Content-Type' to 'text/plain' in the response ↗ this means the format of the response will be 'text/plain'

put this in the body of the response

q is the query param on path /

hash to comment out

put the query in the response body

put the request into the body of the response

N.B. query params are written into the body i.e. q='q_name'

2.13 Get vs Post

GET	POST
• parameters in URL	• parameters in the body
• used for fetching docs	• used for updating data
• max URL length	• no max length by default
• Okay to cache	• Not okay to cache
• shouldn't change the server	• okay to change the server

2.16 Passwords

<input type="password" name="q">

not sent securely to the server! appears as q param in url

2.17 Checkboxes

<input type="checkbox" name="q">

if the box is checked then q=on is added to body
else not much beyond that?

2.19 Radio Buttons

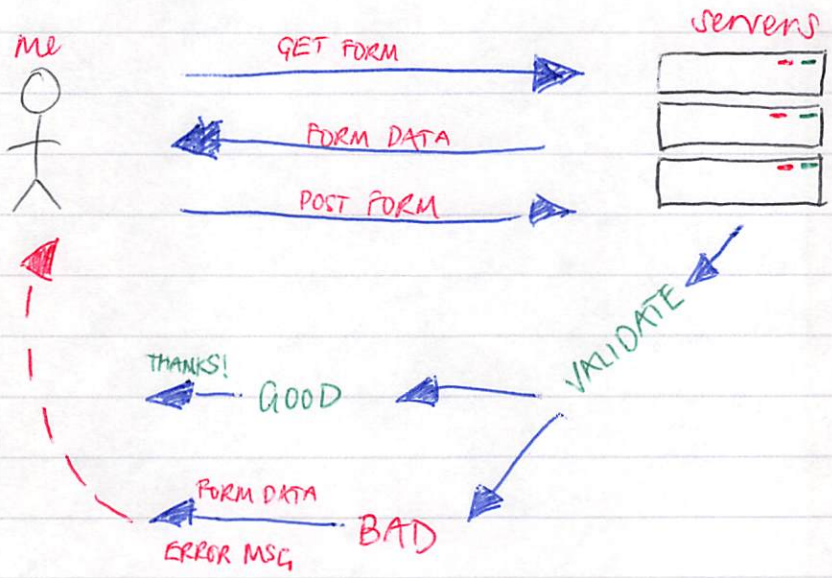
```
<form>
<label>
    One
    <input type="radio" name="q" value="one">
</label>
<label>
    Two
    <input type="radio" name="q" value="two">
</label>
<label>
    Three
    <input type="radio" name="q" value="three">
</label>
```

2.23 Drop downs

```
<select>
    <option value="1">the number one</option>   ← query param value
    <option>two</option>
    <option>three</option>
</select>
```

↑ default value

2.26 Validation



① verify the user's input
② on error, render form again
③ include error msg

2.35 String substitution

" some bold text "

a = "some bold text"
" %s " % a

some bold text

2.36 Substituting multiple strings

"text %s text %s" % (var1, var2)

2.37 Advanced string substitution

"text %(NAME)s text" {"NAME": value}

2.39 Preserving user input

```
<input type="text" value="cool">
```

default value

preserving the users month:

```
<input name="month" value="%(month)s">
```

2.40 html escaping

" → "
> → >
< → <
& → &

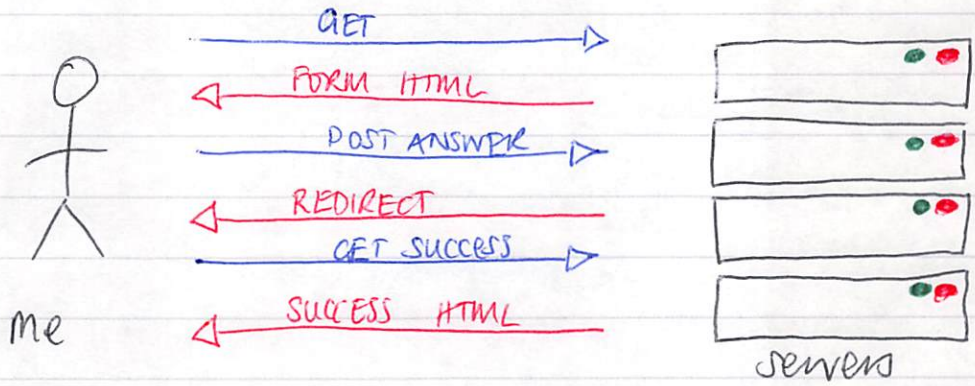
```
html_replace = [('&', '&amp;'),
                ('<', '&lt;'),
                ('>', '&gt;'),
                ('"', '&quot;')]

def escape_html(s):
    for (i,o) in html_replace:
        s = s.replace(i, o)
    return s
```

```
import cgi

def escape_html(s):
    return cgi.escape(s, quote=True)
```

2.47 Redirection

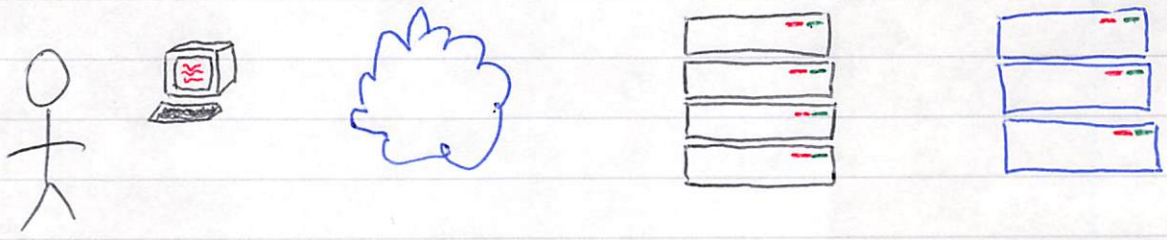


We want to be able to
* redirect to another page on receipt of validated information in a form
* we want to avoid form resubmission on reload

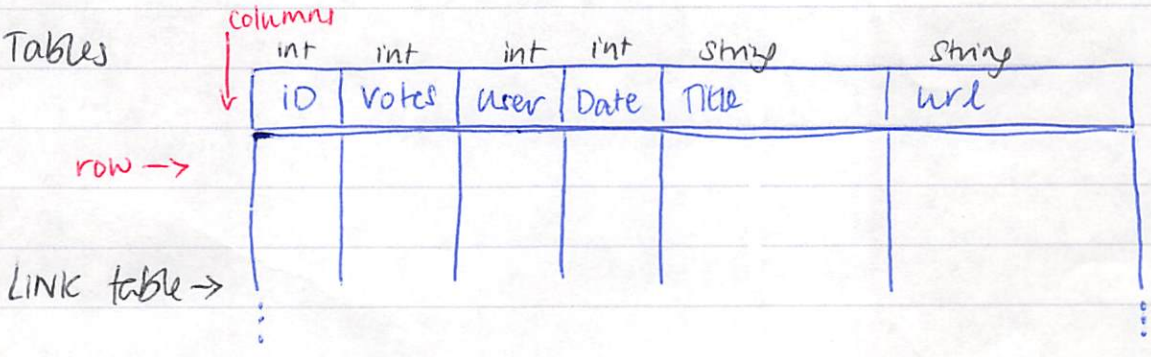
Unit 3

3.1 What is a database?

→ a program that stores and retrieves large amounts of structured data.



3.4 Tables



3.5 Implementing tables in python

```
from collections import namedtuple

Link = namedtuple('Link', ['id', 'sub_id', 'sub_time', 'votes',
                           'title', 'url'])

links = [
  Link(0, 60398, 1334014208.0, 109,
       "C overtakes Java as the no. 1 programming lang",
       "http://pixelstech.net/article/index.php?id=1334017256"),
  Link(1, 60254, 1333962645.0, 891
       "This explains why tech books are overpriced",
       "http://prog21.dadgum.com/65.html"), ... ]


def query():
    for x in links:
        if x.id == 15:
            return x.votes
```

3.7 Downsides of querying data by hand

- → error prone
- → tedious
- → slow

3.8 Types of databases

relational databases ("sequel" databases) | Others

("sequel" = sql; Structured Query Language)

relational databases:

- * Postgresql – reddit / hipmonk
- * MySQL – facebook, everybody
- * SQLite
- * Oracle

FREE: Postgresql, MySQL, SQLite

oracle bought MySQL

all use tables

Others:

- * Google Apps Datastore
- * Amazon Dynamo
- * NoSQL – mongo / couch

3.9 SQL – Structured Query Language

↝ invented in the 1970s

```
SELECT * FROM links WHERE id=5
```

- SELECT: fetch data
- *: all columns (could put title or url for example)
- links: table
- id=5: constraint

3.13 Order By

```
select * from links
where votes>10
order by votes desc
```

not required

ascending by default

asc – ascending
desc – descending

3.14 Joins

ink table:

id	votes	user_id	date	title	url
5	206	22	18/01/01	blah	example.com

ser table:

id	name	password	date
22	Spez	hunter2	20/05/99

```
select link.* from link,user where link.user_ID = user.id
                                and user.name = 'Spez'
```

3.15 Indexes

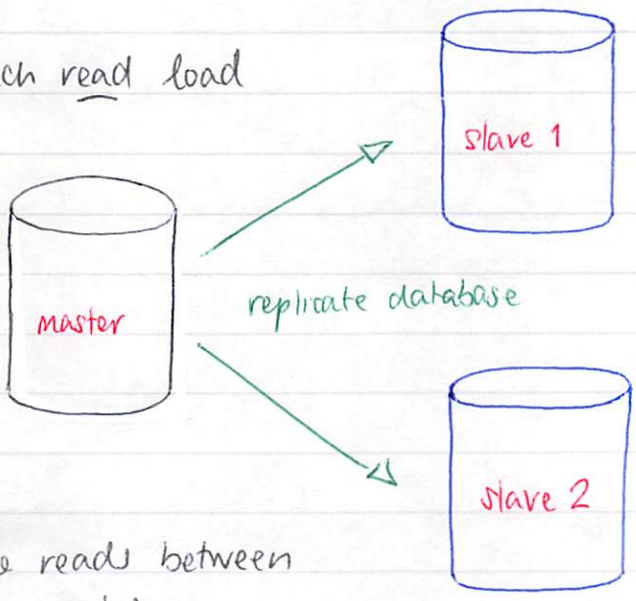
- faster than sequencial scan
- index = { 1: link1,
  2: link2,
  ⋮ ⋮
  key value
  ⋮ ⋮
  n linkn }
- works much like the index of a book
- takes longer to update the database i.e. decreases the speed of database inserts

3.21 Indexes for sorting

hashtable – not sorted – O(1) look-up time
tree – sorted – O(log n) look-up time

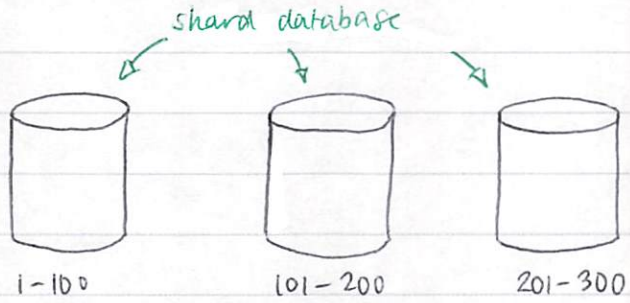
3.23 Scaling databases

* too much read load



* split the reads between the slave databases
* replication lag

* too much data



* complex queries (range query)
* joins become difficult

Google datastore has these limitations

3.25 ACID

Atomicity - all parts of a transaction succeed or fail together

Consistency - the database will always be consistent

Isolation - no transaction can interfere with another's

Durability - once the transaction is comitted, it won't be lost

3.26 Google App Engine Datastore

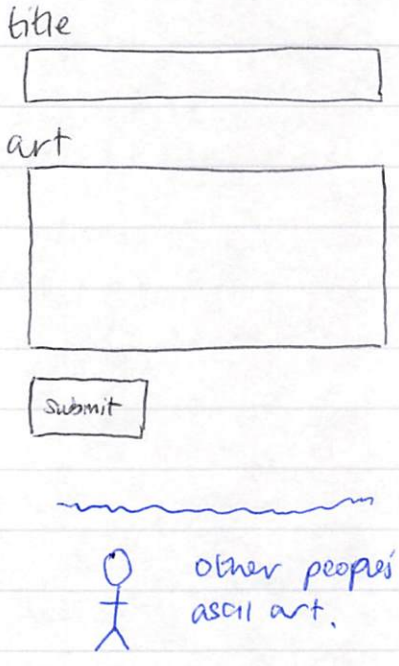
tables → entities
- → columns are not fixed
- → all have an ID field
- → parents/ancestors

SQL → GQL
- → all queries begin with SELECT *
- → no joins
- → all queries must be indexed

Google app engine datastore is sharded and replicated
→ won't have to think about scaling (too much)
→ queries will be quick (because they have to be simple)
→ will have to think about consistency

3.29 ASCII Chan - website for sharing ASCII art

Structure: Form - one page



3.35 Creating entities

```
from google.appengine.ext import db

class Art(db.Model):
```

Art class inherits from the db.Model class

3.36 Datastore types

Integer, Float, String, Date, Time, DateTime
Email, Link, PostalAddress
String: <500 chars, indexed
Text: >500 chars, not-indexed

3.37 Creating entities continued...

```
from google.appengine.ext import db

class Art(db.Model):
    title = db.StringProperty(required=True)
    art = db.TextProperty(required=True)
    created = db.DateTimeProperty(auto_now_add=True)
```

throws an exception if an instance of the class Art is created without a title

automatically will set created to be the current DateTime when the instance of the class Art is created

```
if title and art:
    a = Art(title=title, art=art)
    a.put()
```

3.39 Running queries

```
arts = db.GqlQuery("SELECT * FROM Art "
                   "ORDER BY created DESC")
```

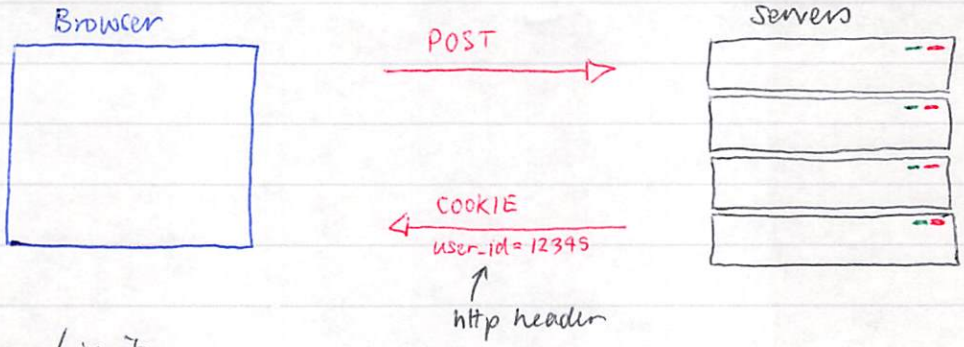
```
{% for art in arts %}
    <div class="art">
        <div class="art-title">{{art.title}}</div>
        <pre class="art-body">{{art.art}}</pre>
    </div>
{% endfor %}
```

Unit 4

4.2 Cookies - a small piece of data stored in the browser for a website (<4 kb)

name = value

user_id = 12345



Browser Limit

* 20 cookies per website
* < 4 kb
* only for one website

4.3 Use of cookies

* storing login information
* storing small amounts of data to avoid hitting a database
* cookies can be cleared by the user from their browser so should not be used for data you don't want to loose
* tracking you for adverts

4.4 Cookie headers

Http Response

Set-Cookie : user_id = 12345

header name — name — value < 4kb

Set-Cookie : last_seen = Dec 25 1985

multiple cookies are listed one below another

Http Request

Cookie : user_id = 12345; last-seen = Dec 25 1985

header name

multiple cookies are separated by a semi-colon

4.5 telnet www.google.com 80

Trying 173.194.79.99...
Connected to www.l.google.com.
Escape character is '^]'.

GET / HTTP/1.0
Host: www.google.com

curl -I www.google.com ← for viewing headers

Request Header:
Cookie: name1=value1; name2=value2;...
Response Header:
Set-Cookie: name1=value1
Set-Cookie: name2=value2
⋮

4.8 Cookie domains

Set-Cookie: name=steve; Domain=www.reddit.com; Path=/foo

Path: the cookie applies to. This is set to '/' by default

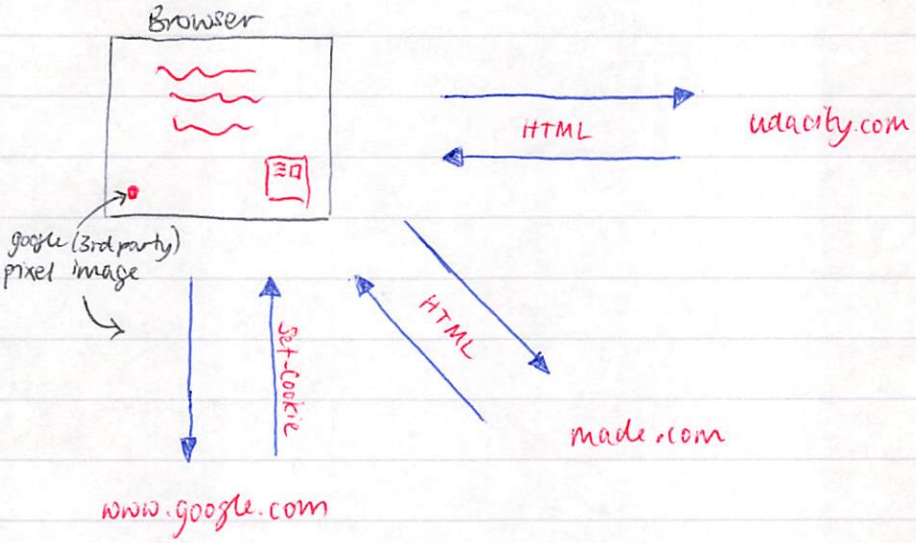
Domain:
www.reddit.com ✓
foo.www.reddit.com ✓
reddit.com ✗
bar.reddit.com ✗

- the domain must have atleast 2 periods, if there is only 1 another will by added to the from
- which domain a browser will send a cookie to

The domain who is serving the request may only set a cookie to its domain or higher. By default the browser will only accept cookies for

www.reddit.com
.reddit.com
foo.reddit.com
bar.reddit.com

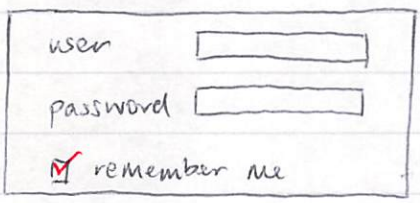
4.12 Ad networks



4.13 Cookie expiration

Set-Cookie: user_id=123 ; Expires= Tue 1 Jan 2025 00:00:00 GMT

If there is no Expires param the cookie expires when you close your browser- it is a "session" cookie



← sets the expiry of the cookie

4.15 Cookies in App Engine

```
class MainPage (Handler):
    def get(self):
        self.response.headers['content-Type'] = 'text/plain'
        visit = self.request.cookies.get('visit', '0')
        if visit.isdigit():  visit = int(visit) +1
        else:                visit = 0
        self.response.headers.add_header('Set-Cookie', 'visit=%s' % visit)
        if visit > 100:  self.write("You are the best ever!")
        else:            self.write("You have been here %s times"
                                    % visit)
```

There is a java script interpeter in chrome; click on the 'console' icon on the developer tools menu

```
> document.cookie
  "visit=23"
> document.cookie = "visit=100001"
  "visit=100001"
```

4.18 Hashing (cs 387)
↳ applied cryptography

$H(x) \rightarrow y$

x is data

y is fixed length bit string 32-256 bit

* Difficult to find the x that generates a particular y
* difficult to generate a specific y
* can't modify x without modifying y

4.19 Hash algorithms

Don't write your own

crc32 - check sums, fast
md5 - fast, ~~secure~~
sha 1 - secureish
sha 256 - pretty good

FAST
↓
SLOW

collision - when two things hash to the same value

4.20 Hashing in python

```
import hash.lib
```

hashlib.

hashlib.algorithms	hashlib.new	hashlib.sha224	hashlib.sha385
hashlib.md5	hashlib.sha1	hashlib.sha256	hashlib.sha512

```
y = hashlib.md5("foo!")

y
<md5 HASH object @ 0x108720470>

y.hexdigest()
'35af8b7a9490467f75f19c1e5459f7e7'
```

4.22 Hashing cookies

Set-Cookie: visits=5, [hash] → to browser

→ from browser

5, [abc123]

```
if H(val) == hash:
    valid!
else:
    not!
```

4.23

```
import hashlib

def hash_str(s):
    return hashlib.md5(s).hexdigest

def make_secure_val(s):
    return s + ',' + hash_str(s)
    return "%s,%s" % (s, hash_str(s))
```

```
class MainPage(Handler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/plain'
        visits = 0
        visit_cookie_str = self.request.cookies.get('visits')
        if visit_cookie_val:
            cookie_val = check_secure_val(visit_cookie_str)
            if cookie_val:
                visits = int(cookie_val)
        visits += 1
        new_cookie_val = make_secure_val(str(visits))
        self.response.headers.add_header('Set-Cookie', 'visits=%s' % visits)
        if visits > 10000:
            self.write("You're the best ever!")
        else
            self.write("You've been here %s times" % visits)
```

4.26 Cookie hashing

visit = 1, [HASH]
H(1) =
H(SECRET + 1) = [HASH]

hashlib

HMAC
Hash-based message authentication code
hmac(secret, key, H) → [HASH]

```
import hmac

hmac.new("secret", "udacity").hexdigest()
```

visits = 1 | HMAC(secret, 1)

4.29 Password Hashing

user

name	password hash
spe3	H(hunter2)
unknoiny	H(metalica)

```
def valid_user(name, pwd):
    user = get_user(name)
    if user and user.password_hash == H(pwd):
        return user
```

4.33 Rainbow Tables

a ← H(a)

password ← H(password)

user

name	hash
spe3	H, salt

h = H(pw + salt)

↑ random characters

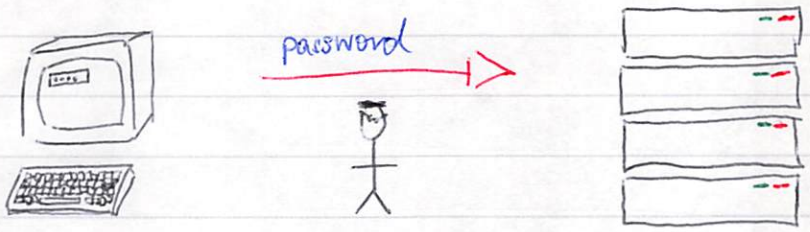
4.37 Bcrypt

Hashing functions is designed to be fast - this means it's also fast to generate rainbow tables

Bcrypt allows you to specify how long you want the encryption to take

4.38 Https - Hypertext Transfer Protocal Secure

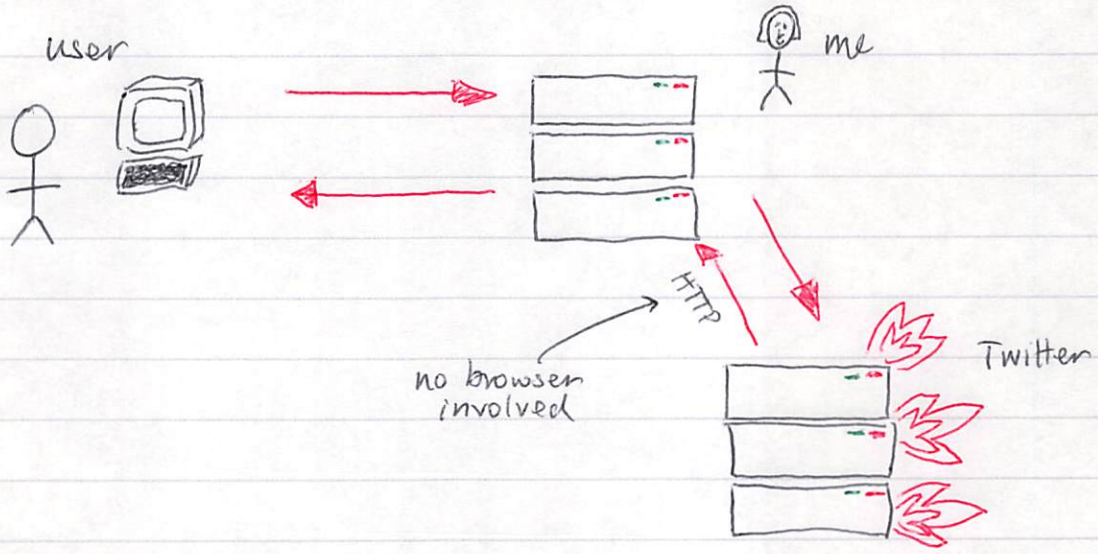
Http but encrypted over ssl



HTTPS = HTTP + SSL

Unit 5

5.2



5.3 URLlib

```
import urllib2   ← follows redirects automatically while
import urllib    ← does not
p = urllib.urlopen("http://www.google.com")
```

<addinfourl at 4423695464 whose fp=<socket._fileobject object at 0x107a85050>

```
c = p.read
c   spits out google's front page

dir(p)   lists function you can use on p

p.url
'http://www.google.com'
```

p.headers

<httplib.HTTPMessage instance at 0x107ac3ab8> ← dictionary

p.headers.items()

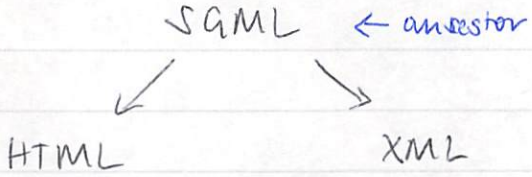
prints out 'keys' and 'values' in the dictionary

p.headers['Content-Type']

'text/html; charset=ISO-8859-1'

5.5 Xml

Html is not the best lang for inter-computer communication.



Xml is similar to Html but more rigorous. There are no void tags

<!DOCTYPE xhtml>

HTML can be expressed as XML

5.7 Parsing XML

```
from xml.dom import minidom
```

document object model ↑ simple and fast

```
minidom.parseString
```

<function xml.dom.minidom.parseString>

```
x = minidom.parseString("<mytag>contents!<children><item>1</item><item>2</item></children></mytag>")
x
```

```
x
<xml.dom.minidom.Document instance at 0x1028d59e0>

dir(x)   lists functions on x

print x.toprettyxml()
<?xml version="1.0" ?>
<mytag>
        content!
        <children>
                <item>
                        1
                </item>
                <item>
                        2
                </item>
        </children>
</mytag>

x.getElementsByTagName("mytag")
[<DOM Element: mytag at 0x1028d5a70>]
x.getElementsByTagName("item")
[<DOM Element: item at 0x1028 5bd8>,
 <DOM Element: item at 0x1028d5cb0>]
x.getElementsByTagName("item")[0].childNodes
[<DOM Text node "u'1'">]
x.getElementsByTagName("item")[0].childNodes.nodeValue
u'1'
 ↑ u tells us it's a unicode string
```

58 RSS RDF Site Summary
↑ Restore Description Framework
Really Simple Syndication

RSS is an xml document that summarizes the content of a website

5.9 Parsing RSS

```
import urllib2
from xml.dom import minidom

content = urllib2.urlopen("http://www.nytimes.com/services/xml/rss/
                          nyt/GlobalHome.xml").read()
d = minidom.parseString(content)
d.getElementsByTagName("item")

len(d.getElementsByTagName("item"))
```

5.10 JSON JavaScript Object Notation

```
{"itineraries": [ {"from": "SFO",
                   "to": "IAD"},
                  {"from": "IAD",
                   "to": "SEA"} ]
```

leg 1 = {"from": "SFO", "to": "IAD"}
leg 2 = {"from": "IAD", "to": "SEA"}

Dictionaries: {"key": value}

Lists: [1, 2, "three", [true, 5.6]] []

int, string, boolean, float, null

```
import json

j = '{"one": 1, "numbers": [2,3,45]}'
json.loads(j)        loads = load string  load expects a file
d = json.loads(j)
d['numbers']
  [2,3,4.5]
```

```
d['one']
1
eval(j)
{'numbers': [2,3,4.5], 'one': 1}
d.keys()
[u'one', u'numbers']
d['numbers'][0]
2
```

5.13 JSON Escaping

```
json.loads('{"story": "once upon a time"}')
{u'story': u'once upon a time'}
```

```
json.loads('{"story": "once u\\"on a time"}')
{u'story': u'once u"on a time'}
```

in python you need one \ to escape to it knows you mean " and then you need another \ to tell JSON you mean "

```
json.loads(r'{"story": "once u\"on a time"}')
{u'story': u'once u"on a time'}
```

r tells python that this is a raw string

5.14 Escaping JSON in python

```
json.dumps([1,2,3])
'[1,2,3]'
```

dumps = dumpstring

```
json.dumps({"one": 1, "two": 2})
'{"one": 1, "two": 2}'
```

```
json.dumps({"one": 1, "two": 'the man said, "cool!"'})
'{"one": 1, "two": "the man said, \\"cool!\\""}'
```

JSON does not use ' to delineate strings (I tried)

```
print json.dumps({"one": 1, "two": 'the man said, "cool!"'})
'{"one": 1, "two": "the man said, \"cool!\"}'
```

5.15 Being a good citizen

* use a good user-agent
  * the header that describes what browser you're using or what program you're using to access somebody
  → who you are, what your name is, maybe a link to your site

* rate-limit yourself

```
import time
while more:
    get_more()
    time.sleep(1)
```

says wait 1 sec before making another request

5.16 Other protocols for communicating between two machines

→ SOAP – based on xml (very complicated)
  – microsoft ← HTTP + JSON
→ protocalbuffers (google) } JSON (data encoding not protocal aswell)
→ Thrif (facebook) }
→ plain-text, custom formats
→ xml } we talked about these
→ JSON }

NOT RECOMMENDED!

5.19 Geolocation

Note: every machines local IP address is 127.0.0.1 this is called the 'loop back' address and this is how the machine refers to itself. This is the ip 'localhost'... refers to

```
import urllib2
from xml.dom import minidom

IP_URL = "http://api.hostip.info/?ip="
def get_coords(ip):
    url = IP_URL + ip
    content = None
    try:
        content = urllib2.urlopen(url).read()
    except URLError:
        return
    if content:
        d = minidom.parseString(content)
        coords = d.getElementsByTagName("gml:coordinates")
        if coords and coords[0].childNodes[0].nodeValue:
            lon, lat = coords[0].childNodes[0].nodeValue.split(',')
            return db.GeoPt(lat, lon)
```

google data type

```
class Art(db.Model):
    title = db.StringProperty(required=True)
    art = db.TextProperty(required=True)
    created = db.DateTimeProperty(auto_now_add=True)
    coords = db.GeoPtProperty()
```

not required

for backwards compatibility

5.20 Debugging

get_coords returns a python object when you print a python object it has angle brackets around it. HTML will treat it as a tag and it won't print properly

repr puts in extra quotes so it prints properly

```
def get(self)
    self.write(self.request.remote_addr)
    self.write(repr(get_coords(self.request.remote_addr)))
    return self.render_front()
```

5.21 Updating the database

```
def post(self)
    title = self.request.get('title')
    art = self.request.get('art')

    if title and art:
        p = Art(title = title, art = art)
        coords = get_coords(self.request.remote_addr)
        if coords:
            p.coords = coords

        p.put()
```

localhost:8082/_ah/admin/datastore

↑ admin consol default to datastore

5.22 Querying Coordinate Information

https://developers.google.com/maps/documentation/staticmaps

```
points = []
for a in arts:
    if arts.coords:
        points.append(a.coords)
```

← equivalent

```
points = filter(None, (a.coords for a in arts))
```

generator

http://maps.googleapis.com/staticmap?size=512x512&sensor=false&markers=color:blue%7Clabel:S%7C40.702147,-74.015794&...

N.B. required in red

```
class MainPage(Handler):
    def render_front(self, error="", title="", art=""):
        arts = db.GqlQuery("SELECT *"
                           "FROM Art"
                           "WHERE ANCESTOR IS :1"
                           "ORDER BY created DESC"
                           "LIMIT 10")
        arts = list(arts)   <- caches result of query to avoid making repetetive queries.
        points = filter(None, (a.coords for a in arts))

        self.render('front.html', title=title, art=art,
                    error=error, arts=arts)
```

5.24 Putting it all together

```
GMAPS_URL = "http://maps.googleapis.com/maps/api/staticmap?
            size=380x263&sensor=false&"

def gmaps_img(points):
    markers = '&'.join('markers=%s,%s' % (p.lat, p.lon)
                       for p in points)
    return GMAPS_URL + markers
```

In the html file ...

```
{% if img_url %}
    <img src="{{img_url}}">
{% endif %}
```

Unit 6

6.02 Why scale:

→ so we can serve more requests concurrently
→ so we can store more data
→ so we are more resilient to failure
→ so we can serve requests faster

6.3 one can scale

→ bandwidth
→ computers (cpu, memory)
→ power
→ storage

6.4 Techniques for scaling

→ optimise code:
cost of machine + its maintenance vs the cost per developer
→ cache complex operations
→ upgrade machines
more memory / more disk space / faster cpu
→ add more machines

6.5 Caching

Caching refers to storing the result of an operation so that future requests return faster

When do we cache?

→ when the computation is slow
→ it will run mutiple times
→ when the output is the same for a particular input
→ your hosting provider charges for db access

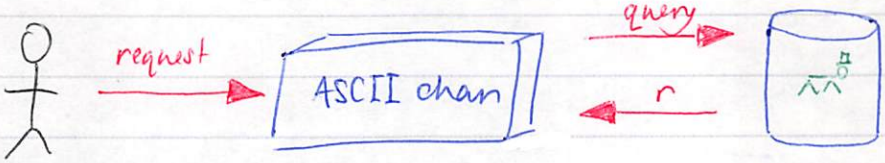
Cache is a hashtable

```
if request is in cache:
    return cache[request]
else:
    r = db_read()
    cache[request] = r
    return r
```

cache hit (<1 ms)

cache miss (100 ms)

6.6



→ process the request
→ query the database ← Takes time, worth optimising
→ collate the results
→ render the HTML

optimise code - indexes
more machines
bigger machine
cache

6.9

```
CACHE = {}
def top_arts():
    key = 'top'
    if key in CACHE:
        arts = CACHE[key]
    else:
        logging.error("DB QUERY")
        arts = db.GqlQuery("SELECT *"
                           "FROM Art"
                           "WHERE ANCESTOR IS :1"
                           "ORDER BY created DESC"
                           "LIMIT 10",
                           art_key)
        arts = list(arts)
        CACHE[key] = arts
    return arts

class MainPage(Handler):
    def render_front(self, error='', title='', art=''):
        arts = top_arts()
        img_url = None
        points = filter(None, (a.author_loc for a in arts))
        if points:
            img_url = gmap_img(points)
        self.render('front_html', title=title, art=art,
                    error=error, arts=arts, img_url=img_url)
```

import logging

Then in the post method we just clear the cache after adding the art to the database so...

```
        p.put()
        CACHE.clear()
```

So now, if you go to the homepage...

```
INFO 2012-05-08 18:51:31 988 dev_appserver.py:2884]
    "GET / HTTP/1.1" 200 -
INFO 2012-05-08 18:51:32 130 dev_appserver.py:2884]
    "GET /favicon.ico HTTP/1.1" 404 -
```

If you submit some art...

```
INFO 2012-05-08 18:53:07 844 dev_appserver.py:2884]
    "POST / HTTP/1.1" 302 -
ERROR 2012-05-08 18:53:07 855 asoiichan.py:74]
    DB QUERY
INFO 2012-05-08 18:53:07 866 dev_appserver.py:2884]
    "GET / HTTP/1.1" 200 -
INFO 2012-05-08 18:53:08 086 dev_appserver.py:2884]
    "GET / favicon.ico HTTP/1.1" 404 -
```

So it only hits the database if some art is submitted

6.10 Cache stampede

Problem with the above approach...
While the cache is empty everyone who goes to the homepage queries the database

Solution would be to overwrite the cache when art is submitted rather than clearing it

```
def top_arts(update = FALSE)
    key = 'top'
    if not update and key in CACHE:
```

all the other code is the same.

and in the post method we
→ add to the database
→ re-run the query
→ and update the cache

```
p.put()
top_arts(TRUE)
```

So now when we submit a new piece of art...

```
ERROR 2012-05-08 20:04:30 104 asciichan.py:74]
    DB QUERY
INFO 2012-05-08 20:04:30 116 dev_appserver.py:2884]
    "POST / HTTP/1.1" 302 -
INFO 2012-05-08 20:04:30 135 dev_appserver.py:2884]
    "GET / HTTP/1.1" 200 -
INFO 2012-05-08 20:04:30 511 dev_appserver.py:2884]
    "GET /favicon.ico HTTP/1.1' 404 -
```

so it the person who submits the art that queries the database

6.11 Caching techniques

Approach	DB read / page view	DB read / submit	Bugs
no caching	every	none	
naive caching	cache miss	none	
clear cache	cache miss	none	
refresh cache	rarely	1	
SIMPLE USERS SHOULDN'T TOUCH THE DATABASE!			
update cache	0	0	

The last line of the table is possible by not querying the database to update the cache and updating it directly.

This means writing more complex code - you may have multiple tables to update depending on how you allow users to view the data in your database

However no users ever query the database they only write to it so there is a speed gain.

N.B. The more accurate the cache
the more complex the code

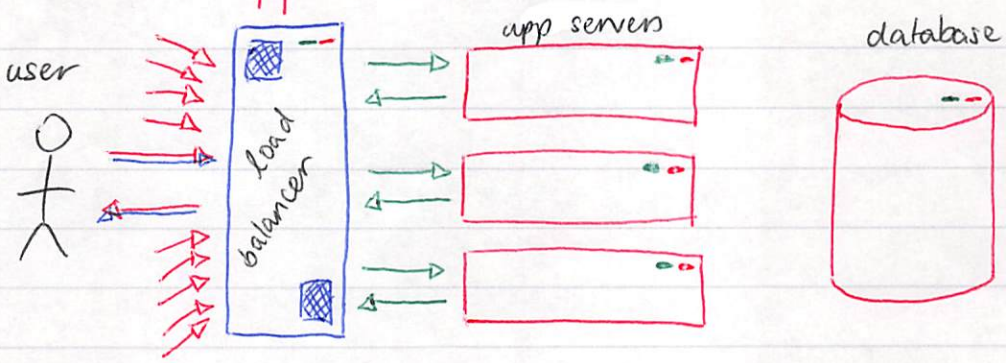
ultimate solution for high traffic website

6.12 App server scaling

On every request we...

- ~~process request~~
- ~~db query~~ we've improved this
- ~~collate results~~
- ~~render HTML~~

we can improve the other three by adding additional app servers



Load balancer is optimised to choose which app server to send the request to

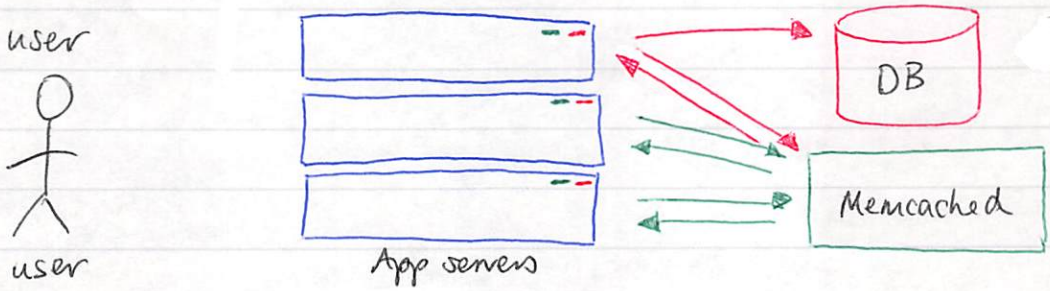
Load balancing algorithms:
- → random
- → round robin (in cyclic order)
- → load based

6.14 Caching with multiple servers

Our dict cache is problematic when we have multiple app servers
- → multiple app servers = multiple caches
  How do we keep them in sync?
- → each app server may have to hit the database to update its cache

6.15 Memcached ← used by... Facebook, Reddit, YouTube, Twitter, ... Basically everyone!

A very fast in-memory cache built by Danga to power Live Journal which was a popular site in 2003



Memcached is really just a key-value store. (both strings)

It has methods: set(key, value)
get(key) → value
delete(key)

6.17 Memcached properties

Stored entirely in memory so...

- -> fast
- -> not durable (looses everything if you restart)
- -> limited by memory

If there is not space in memcached it just throws away the data that was least recently used (LRU) - as opposed to least frequenty used (LFU)

6.18 Using memcached

```
from google.appengine.api import memcache

def top_arts(update=False):
    key = 'top'
    arts = memcache.get(key)
    if arts is None or update:
        logging.error("DB QUERY")
        arts = db.GqlQuery("SELECT * "
                           "FROM Art "
                           "WHERE ANCESTOR IS :1 "
                           "ORDER BY created DESC "
                           "LIMIT 10",
                           art_key)
        arts = list(arts)
        memcache.set(key, arts)
    return arts
```

You can go to:

localhost:8082/_ah/admin/memcache

to 'administrate' your cache

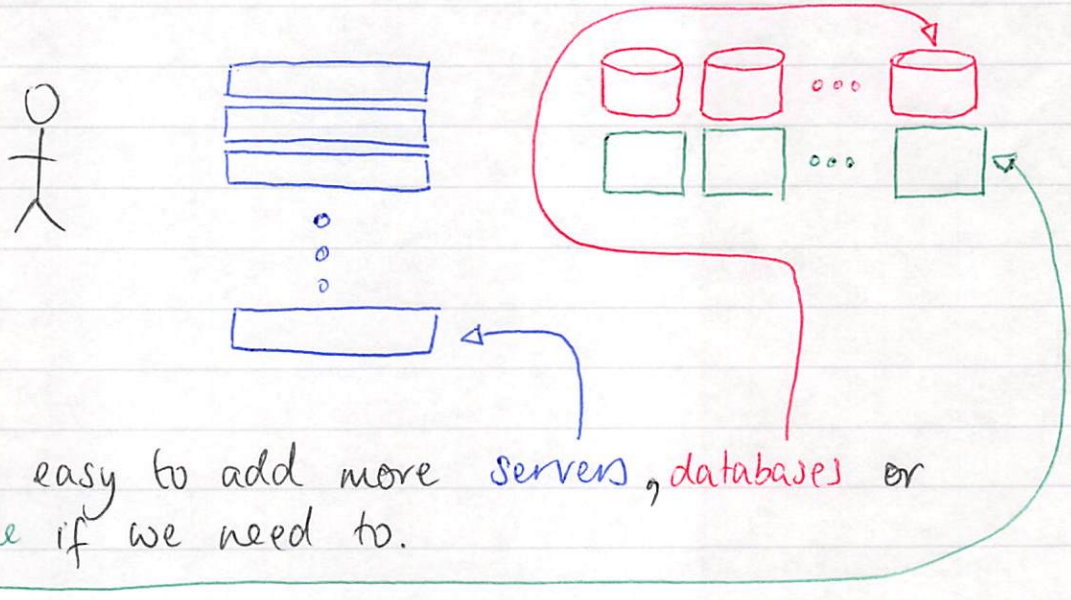
6.19 Stateless

→ cache survives restarts
  → not cleared after code refactors (unlike the dict approach)
→ app is now stateless → stores no 'state' between requests
  → no states between requests
  → apps are now interchangably
  → adding apps is easy
  → apps can be scaled independent of cache and db

Now all states are stored in
- → cookies
- → database
- → memcached

} not the apps

this makes our app more scalable because...



it's now easy to add more servers, databases or memcache if we need to.

6.20 Advanced cache updates

Problem: multiple users submit at the same time, update cache at the same time, overwriting each other

6.21 CAS (Check and set)

get(key) → value, unique
cas(key, value, unique) → True
→ False

App 1
```
v, u = mc.get(k)
mc.cas(k, y, u)
  → True
```

App 2
```
v, u = mc.get(k)
r = mc.cas(k, y, u)
while r == False:
    v, u = mc.get(k)
```

6.23 Separating services

i.e. separating the work between servers, databases and memcache

-> so they can be scaled independently
-> to increase fault tolerance
-> so two very different processes aren't competing for resources
-> so they can be updated independently

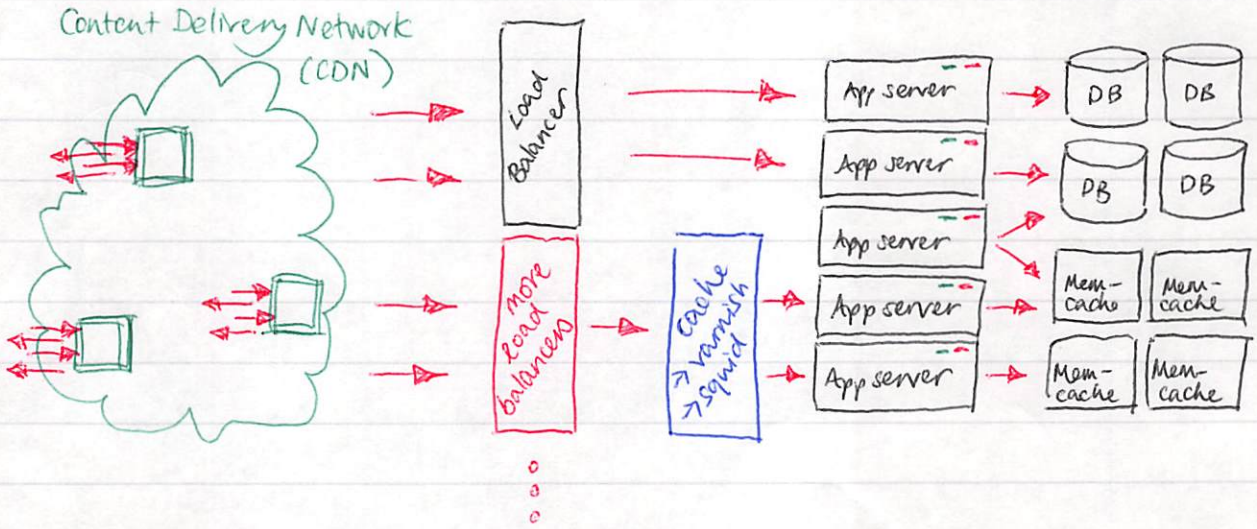
6.24 Additional pieces

* can have multiple load balancers
  -> use DNS round robin to load balan these
     ↑ system which converts domain name to ip address
  -> one domain may map to several ips - one for each load balancer

* could have another cache before the app servers
  -> used for HTML, images, things that don't change
     e.g. users that aren't logged in might not make dynamic request

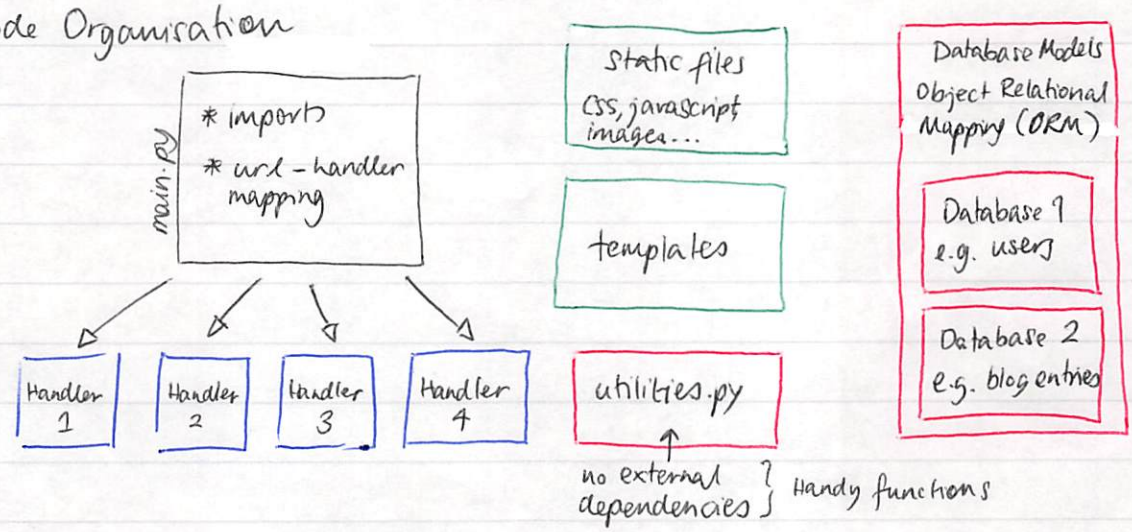
e.g. varnish, squid or custom solution

* could have a Content delivery network (CDN) before the load balancers to cache your content all over the internet for requests that don't require dynamic request - perhaps the response is already cached?
  -> usually 3rd party companies
  -> facebook and google have their own data centres all over the world - the closer the cache to the customer the faster they can serve



Unit 7

7.3 Code Organisation



File structure

```
/main.py
 handlers.py
 /lib
    /DB
       models
    utilities.py
 /templates - base.html
 /static
```

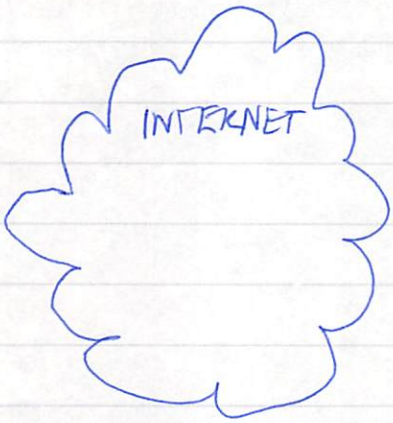
7.4 Hosting

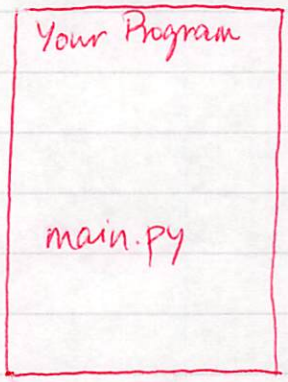
Local
- → fine for single user
- → not always on
- → not always accessible
- → need to get a static IP

Co-locate
- → control the machines
- → pay rent, power, bandwidth
- → high work

Managed hosting
- → Amazon Web Services, Rackspace, Linode
- → Rent machines
- → medium systems admin

Google App Engine / Heroku
- → zero systems admin
- → no 'machines', 'os'
- → difficult customisation





Important features	Webapp2	non-important
direct GET/POST request headers	HTTP scheduling url-mapping	- sessions - caching - forms - DB ~ ORM

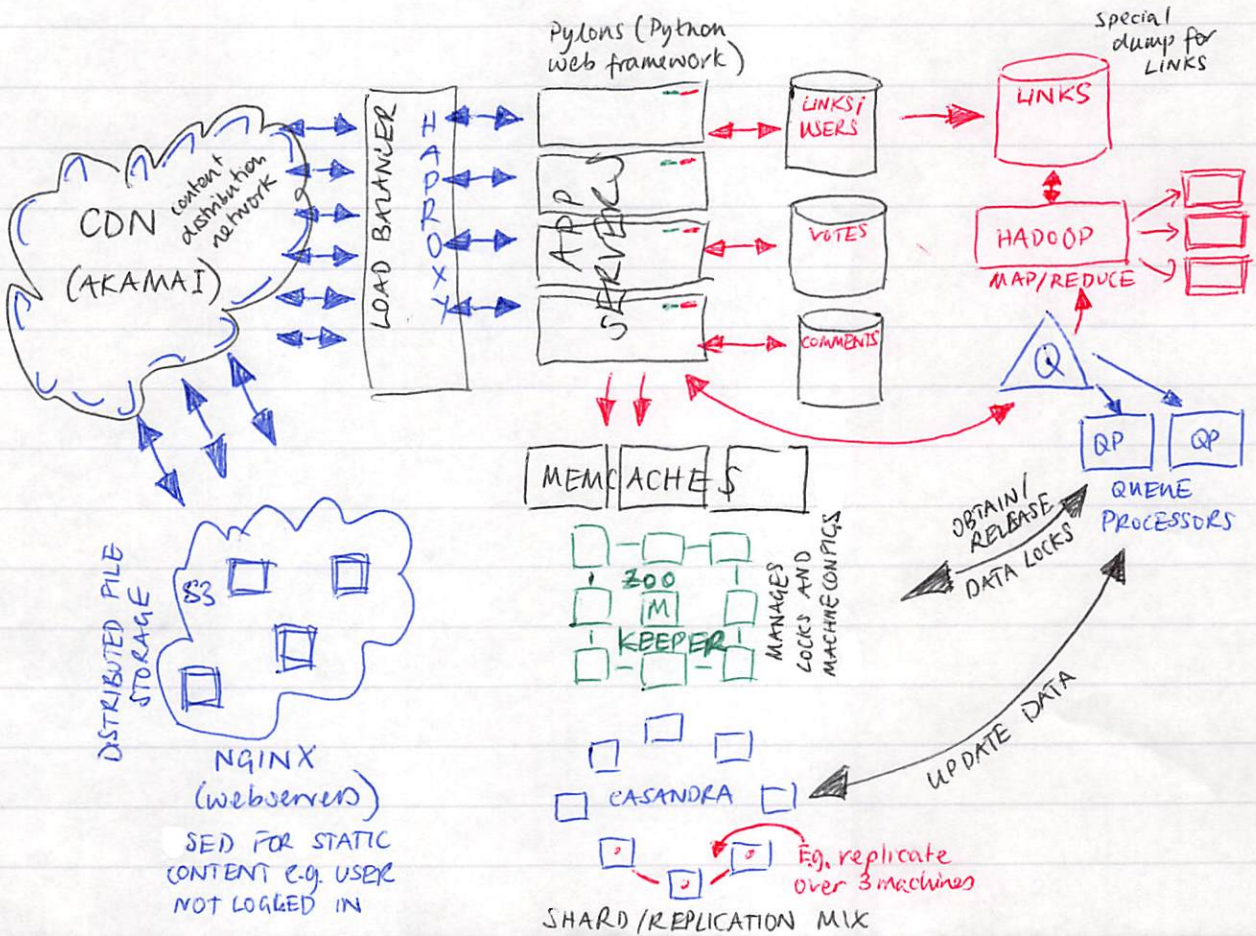
} too high level

Separate code and templates!

↑ mako / jinja2

use templates only to generate html or css

Reddit Architecture ...



7.30 Growing Reddit (social perspective)

-> fake content ← users
  -> set the tone
  -> feel alive
-> no comments
  -> MVP
-> no categorization
  -> low user investment
-> no emails
  -> low user investment
-> no censoring

7.31 Spam prevention

link submission
  <a rel="no follow" href ...
  -> links only relevant if there are enough votes
spammers use the same password
spammers didn't understand
  -> people who post stuff don't comment straight after
submission
  -> people don't submit in bulk
security by obscurity
  -> don't tell people when they are caught

7.32 Also good to learn...
  -> CSS
  -> Javascript
  -> AJAX (Asyncronis Javascrip And Xml)

733 Google App Engine

Pros → provide infrastructure
- → caching
- → datastore (highly replicated)
- → force you to think about consistency
- → good queueing
- → versioning
  - → AB testing for partial users admin
  - → can upload completely different codebase against same datastore and services
  - → run monitoring and admin alongside the app
- → Autoscaling – auto sharding and replication

Cons →
- → limited customizability
- → 10 minute limit on tasks
- → no precompiled c modules
- → ssl cert support for custom domains
- → DATABASE
  - * Queries limited to 1000 queries before paging
  - * Backup & restore – need to use 3rd party tools
    – expensive
  - * No data viewer for all data types

CS253 Web Application Engineering

Homework 1

Google App Engine notes:

* To run application locally click 'Run'
  → Go to 'localhost:port'

* To run application on app engine click 'Deploy'
  → Go to 'ApplicationName.appspot.com'

* Code in 'main.py'

Homework 2

udacity-cs253.appspot.com/unit2/rot13
udacity-cs253.appspot.com/unit2/signup

Homework 3

udacity-cs253.appspot.com/blog — front
/newpost
/1002 — permalink

Homework 4

udacity-cs253.appspot.com/signup
/login
/logout

Homework 5

udacity-cs253.appspot.com/blog.json
/1002.json