FALMOUTH UNIVERSITY

Lecture 9: Users and User Security

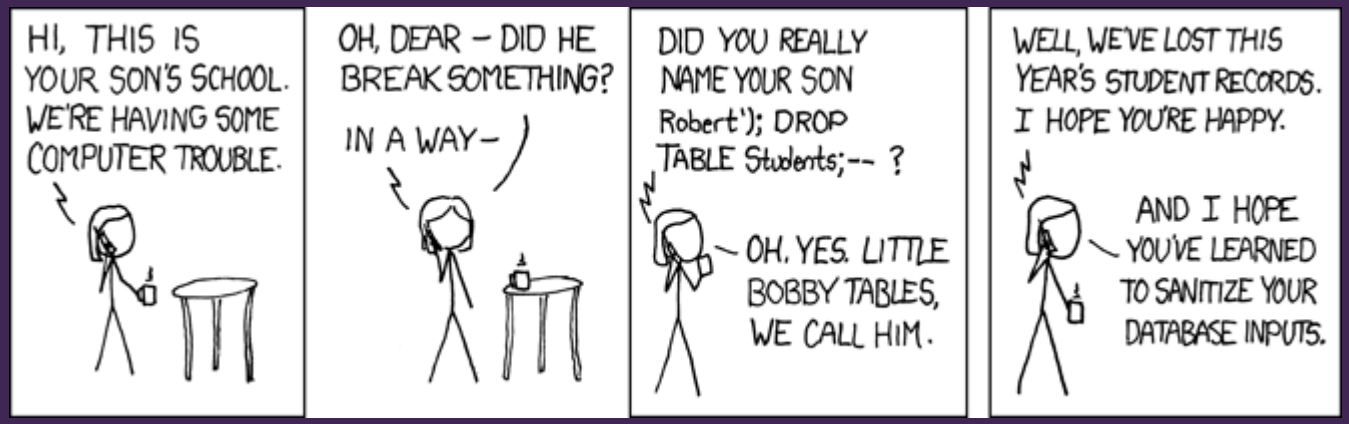
COMP260: Distributed Systems
BSc (Hons) Computing for Games

- Today's session:
  - SQL
  - Password-based User Accounts
  - Adding User Accounts to GaaS
  - Workshop preview

- SQL

- SQL
  - Did your experiments in SQL yield success last week?
    - SQL *CRUDS* testbed
    - SQL SUD – dungeon database
    - Persistent SUD
    - SQL MUD

- SQL
  - SQL Injections



- Typically, add extra data to input data to allow extra SQL commands to be called
  - Hence input santitation

- SQL
  - SQL Injections
    - http://bobby-tables.com/python.html

Python

Using the Python DB API, don't do this:

```
# Do NOT do it this way.
cmd = "update people set name='%s' where id='%s'" % (name, id)
curs.execute(cmd)
```

This builds a SQL string using Python's string formatting, but it creates an unsafe string that is then passed through to the database and executed.

Instead, do this:

```
cmd = "update people set name=%s where id=%s"
curs.execute(cmd, (name, id))
```

So if you are using MySQL or PostgreSQL, use `%s` (even for numbers and other non-string values!) and if you are using SQLite use `?`.

- SQL
  - SQL Injections
    - https://www.pythoncentral.io/introduction-to-sqlite-in-python/

```python
cursor = db.cursor()
name1 = 'Andres'
phone1 = '3366858'
email1 = 'user@example.com'
# A very secure password
password1 = '12345'

name2 = 'John'
phone2 = '5557241'
email2 = 'johndoe@example.com'
password2 = 'abcdef'

# Insert user 1
cursor.execute('''INSERT INTO users(name, phone, email, password)
                  VALUES(?,?,?,?)''', (name1,phone1, email1, password1))
print('First user inserted')

# Insert user 2
cursor.execute('''INSERT INTO users(name, phone, email, password)
                  VALUES(?,?,?,?)''', (name2,phone2, email2, password2))
print('Second user inserted')

db.commit()
```

- SQL
  - SQL Injections
    - Obviously, I thought I'd have a go

```
if key is '7':
    try:
        result = cursor.execute("insert into table_phonenumbers (name,number) values (\"bbb\", \"666\");drop table if exists table_phonenumbers;")
        print(str(result))
    except Exception as e:  e: You can only execute one statement at a time.
        print('Failed to drop tables - ' + str(e))

if key is 'x':
```

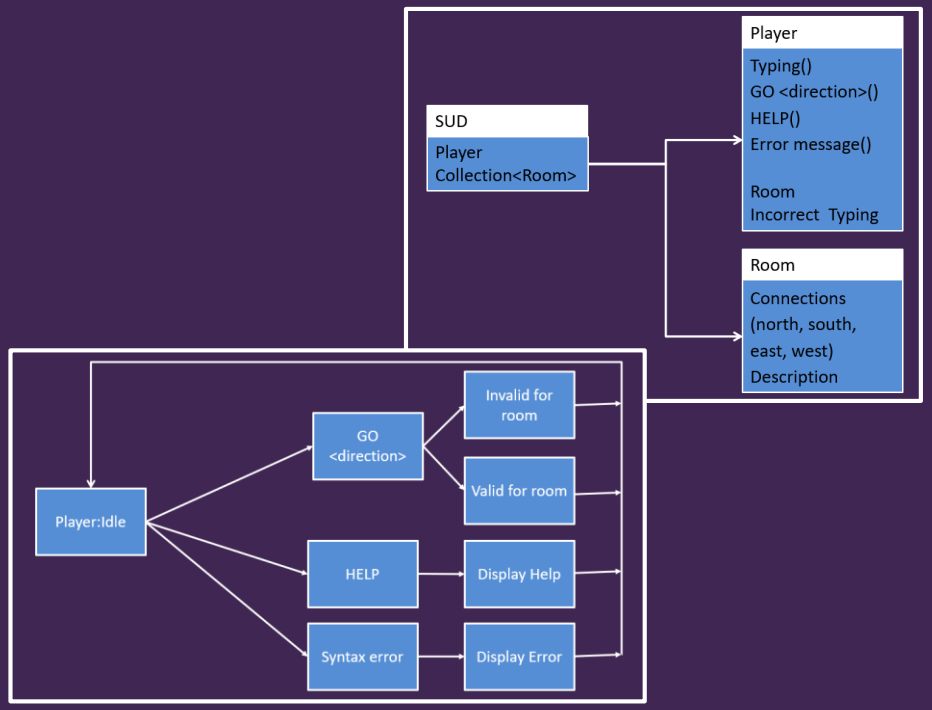
    - Python sqlite implementation may be more robust than we give it credit for

- SQL
  - Data oriented programming

- SQL
  - Data oriented programming
    - We've seen that OOA / OOD / OOP gives us a powerful paradigm to go from loose textual requirements to form (class hierarchies / component & package) and function (flowcharts, state diagrams, activity diagrams etc)

The SUD is a collection of rooms that are laid out on one level (there are no up or down exits). Rooms can be joined by north, south, east or west connections. A room does not have to have connections in all directions and it may not have more than one connection per direction and the connections do not have to be bidirectional.

When the player enters a room, they will be presented with a textual description of the room along with a list of exits they can take from that room. Typing help will give the player all of the text commands. Entering a command incorrectly will result in an error message.

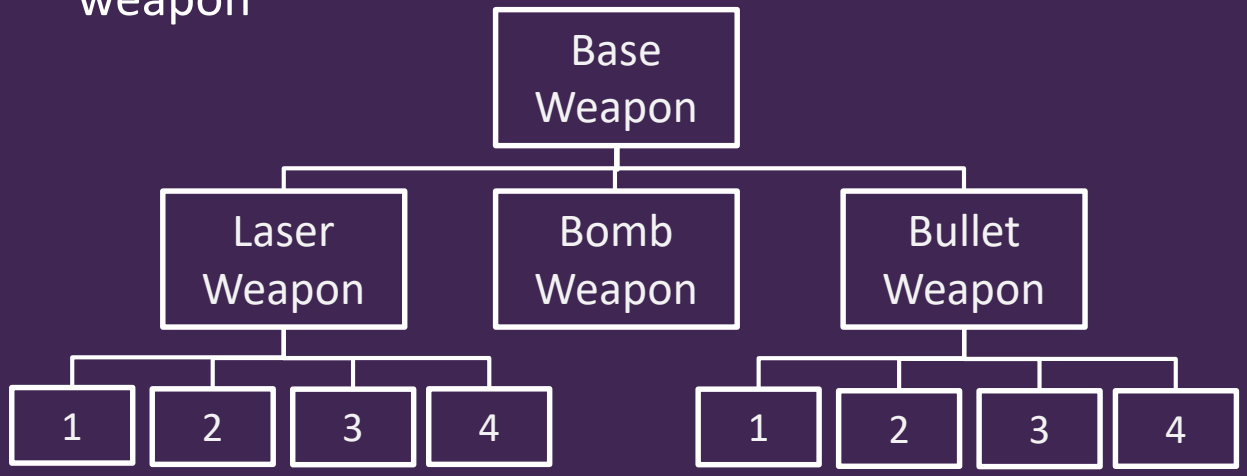


- SQL
  - Data oriented programming
    - OOA / OOD / OOP works really well, until we end up with large class hierarchies
    - By large
      - Lots of depth (base-r / parent classes)
      - Lots of width (derived classes per base/parent class)
    - 2 killer problems
      - 1. compile time
        - » Touching baser-class definition will lead to large rebuilds
        - » Can be a performance issue in C++
      - 2. designer impact
        - » Designers need to touch their data
        - » Having it defined in code may not be the best place for this
          - To a degree, Unity side-steps this with the Inspector, but it’s not a perfect vehicle and struggles under scale

- SQL
  - Data oriented programming
    - With data orientation
      - Move game data that is important to designers into frameworks that they can work on WITHOUT involving programmers
      - Make game architecture more generic and less hierarchical
    - Databases / designer legible formats (SQL, Excel, XML) are good for this

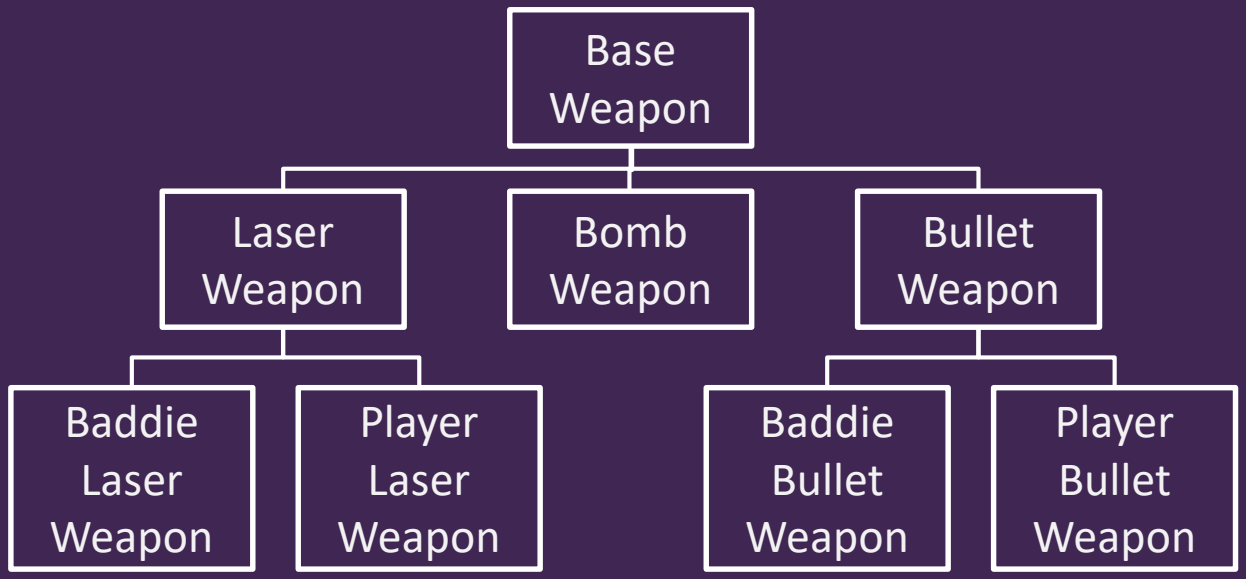
- SQL
  - Data oriented programming
    - An Example: Underzone Game
      - This game has 20 weapons that are shared between player and baddies
        - » Guns, lasers and bombs
      - Each has their own ammo and damage types, vfx, sfx, in-game text and so on
      - Also have their own level ups (procedural weapons)
        - » Damage, range, speed, spread, blast radius, shot time, dps etc
      - TL;DR
        - » Lots of weapons for player and baddies
        - » Lots of data that I (programmer) don't want to touch

- SQL
  - Data oriented programming
    - With OOD/P
      - Build a massive class hierarchy and populate with each weapon

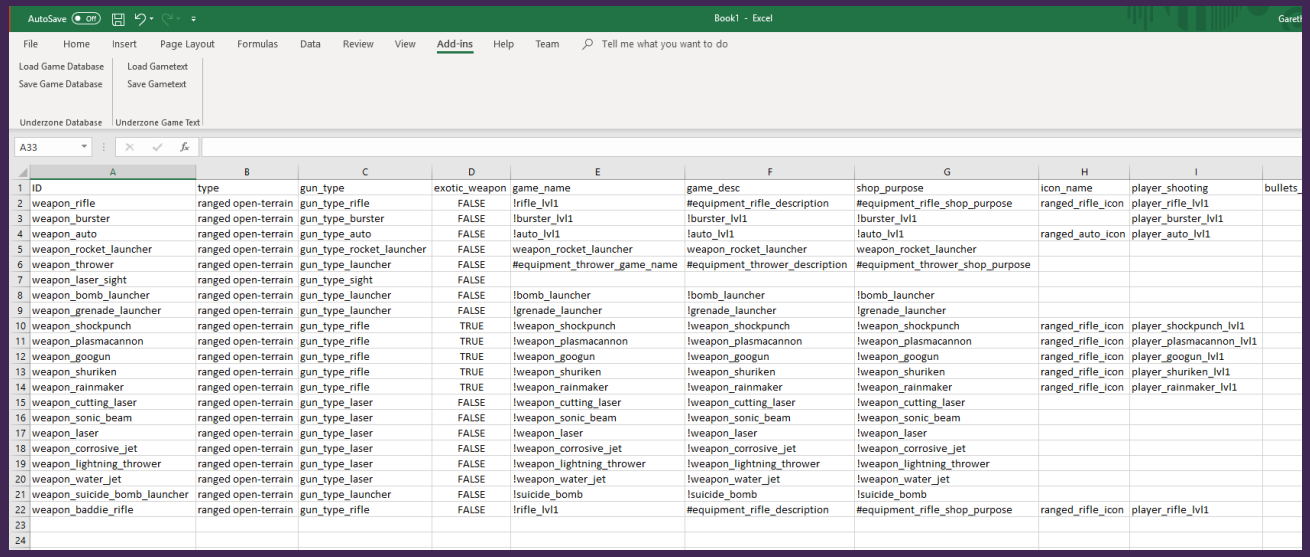


      - How do we distinguish between baddie and player weapon approaches?
        - » `if(is_baddie_owner == true)`
      - How do we manage all the weapon data?
        - » Store in class and let designers work with it?
      - Often end up with a lot of copypasta lightweight derived classes (1-4)

- SQL
  - Data oriented programming
    - With DOA
      - 1. Build a different class hierarchy



- SQL
  - Data oriented programming
    - With DOA
      - 2. Put all the weapon definitions in a database and let designers populate and manage



	A	B	C	D	E	F	G	H	I	
1	ID	type	gun_type	exotic_weapon	game_name	game_desc	shop_purpose	icon_name	player_shooting	bullets_
2	weapon_rifle	ranged open-terrain	gun_type_rifle	FALSE	!rifle_lvl1	#equipment_rifle_description	#equipment_rifle_shop_purpose	ranged_rifle_icon	player_rifle_lvl1	
3	weapon_burster	ranged open-terrain	gun_type_burster	FALSE	!burster_lvl1	!burster_lvl1	!burster_lvl1		player_burster_lvl1	
4	weapon_auto	ranged open-terrain	gun_type_auto	FALSE	!auto_lvl1	!auto_lvl1	!auto_lvl1	ranged_auto_icon	player_auto_lvl1	
5	weapon_rocket_launcher	ranged open-terrain	gun_type_rocket_launcher	FALSE	weapon_rocket_launcher	weapon_rocket_launcher	weapon_rocket_launcher			
6	weapon_thrower	ranged open-terrain	gun_type_launcher	FALSE	#equipment_thrower_game_name	#equipment_thrower_description	#equipment_thrower_shop_purpose			
7	weapon_laser_sight	ranged open-terrain	gun_type_sight	FALSE						
8	weapon_bomb_launcher	ranged open-terrain	gun_type_launcher	FALSE	!bomb_launcher	!bomb_launcher	!bomb_launcher			
9	weapon_grenade_launcher	ranged open-terrain	gun_type_launcher	FALSE	!grenade_launcher	!grenade_launcher	!grenade_launcher			
10	weapon_shockpunch	ranged open-terrain	gun_type_rifle	TRUE	!weapon_shockpunch	!weapon_shockpunch	!weapon_shockpunch	ranged_rifle_icon	player_shockpunch_lvl1	
11	weapon_plasmacannon	ranged open-terrain	gun_type_rifle	TRUE	!weapon_plasmacannon	!weapon_plasmacannon	!weapon_plasmacannon	ranged_rifle_icon	player_plasmacannon_lvl1	
12	weapon_googun	ranged open-terrain	gun_type_rifle	TRUE	!weapon_googun	!weapon_googun	!weapon_googun	ranged_rifle_icon	player_googun_lvl1	
13	weapon_shuriken	ranged open-terrain	gun_type_rifle	TRUE	!weapon_shuriken	!weapon_shuriken	!weapon_shuriken	ranged_rifle_icon	player_shuriken_lvl1	
14	weapon_rainmaker	ranged open-terrain	gun_type_rifle	TRUE	!weapon_rainmaker	!weapon_rainmaker	!weapon_rainmaker	ranged_rifle_icon	player_rainmaker_lvl1	
15	weapon_cutting_laser	ranged open-terrain	gun_type_laser	FALSE	!weapon_cutting_laser	!weapon_cutting_laser	!weapon_cutting_laser			
16	weapon_sonic_beam	ranged open-terrain	gun_type_laser	FALSE	!weapon_sonic_beam	!weapon_sonic_beam	!weapon_sonic_beam			
17	weapon_laser	ranged open-terrain	gun_type_laser	FALSE	!weapon_laser	!weapon_laser	!weapon_laser			
18	weapon_corrosive_jet	ranged open-terrain	gun_type_laser	FALSE	!weapon_corrosive_jet	!weapon_corrosive_jet	!weapon_corrosive_jet			
19	weapon_lightning_thrower	ranged open-terrain	gun_type_laser	FALSE	!weapon_lightning_thrower	!weapon_lightning_thrower	!weapon_lightning_thrower			
20	weapon_water_jet	ranged open-terrain	gun_type_laser	FALSE	!weapon_water_jet	!weapon_water_jet	!weapon_water_jet			
21	weapon_suicide_bomb_launcher	ranged open-terrain	gun_type_launcher	FALSE	!suicide_bomb	!suicide_bomb	!suicide_bomb			
22	weapon_baddie_rifle	ranged open-terrain	gun_type_rifle	FALSE	!rifle_lvl1	#equipment_rifle_description	#equipment_rifle_shop_purpose	ranged_rifle_icon	player_rifle_lvl1	
23										
24										

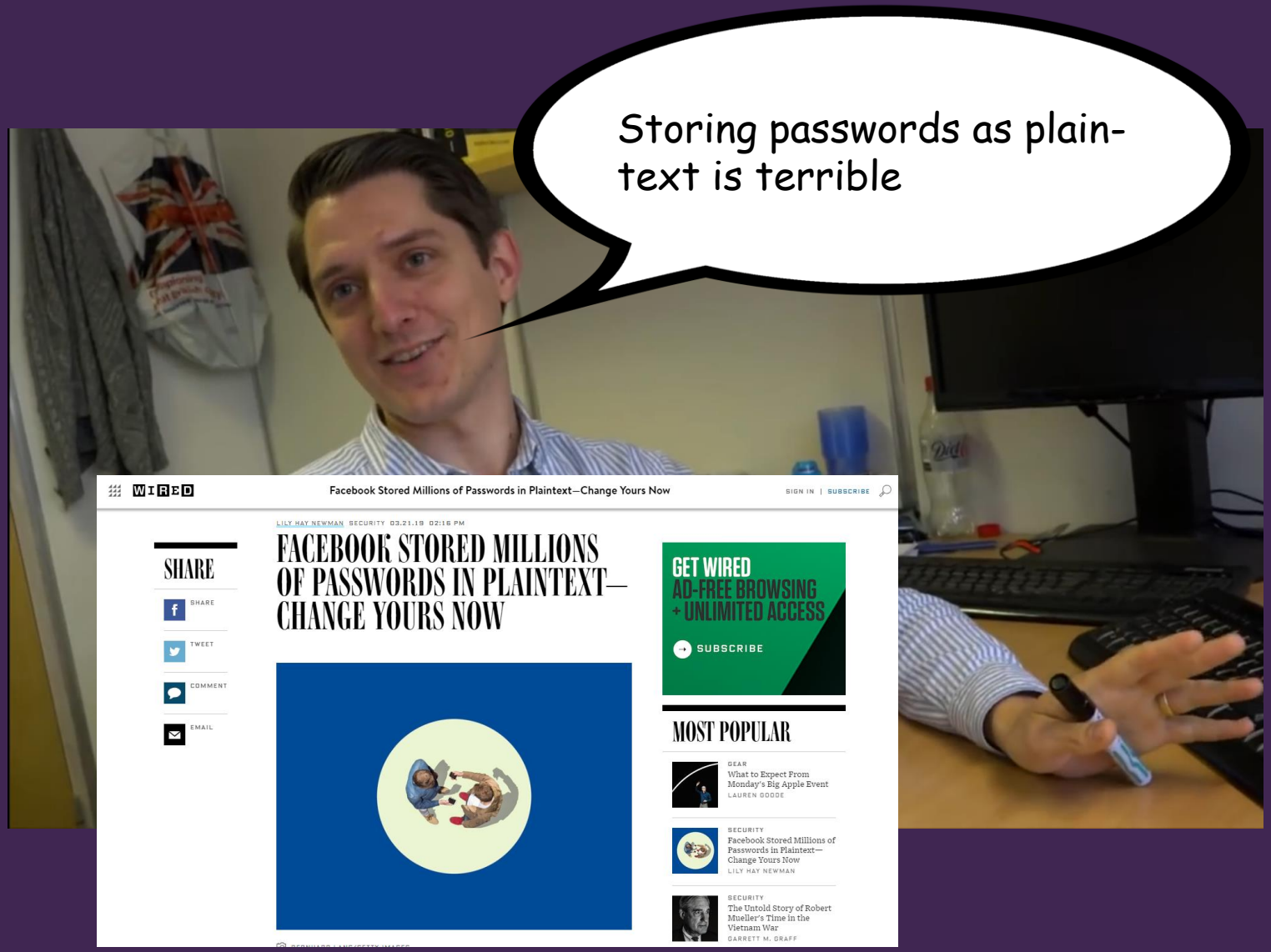
» 21 weapons x 30 attributes

- SQL
  - Data oriented programming
    - With DOA / DOP
      - 3. End up with fatter base classes (weapon & player / baddie weapon)
        - » But data modelling is divorced from code (which is good)
        - » Makes it very easy to share functionality, stats and fx between player and baddie (or make them different)
        - » Gives designers control over game content
          - If they want 2 dozen more weapons, go for it
        - » Gives programmers control over systems
          - Main concern is the way entities work and the fundamental differences between different entities

- Password-based User Accounts

- Password-based User Accounts
  - Adding passwords to accounts is a dictionary
    - <username, password>
    - Which we can store in a database

- Password-based User Accounts



- Password-based User Accounts
  - To store passwords, they should be hashed
    - i.e. we store the result of a transformation and not the raw data
      - Then, we can compare a transformed password with the known hash
      - Hash should be unique for the data provide
  - In Python, `hashlib` provides a library of hashing functions

```
import hashlib

password = 'test'

simpleHash = hashlib.md5()
simpleHash.update(bytes(password,'utf-8'))
print("simple hash:"+simpleHash.hexdigest())
```

Other hashes are available as we all know that md5 is rubbish

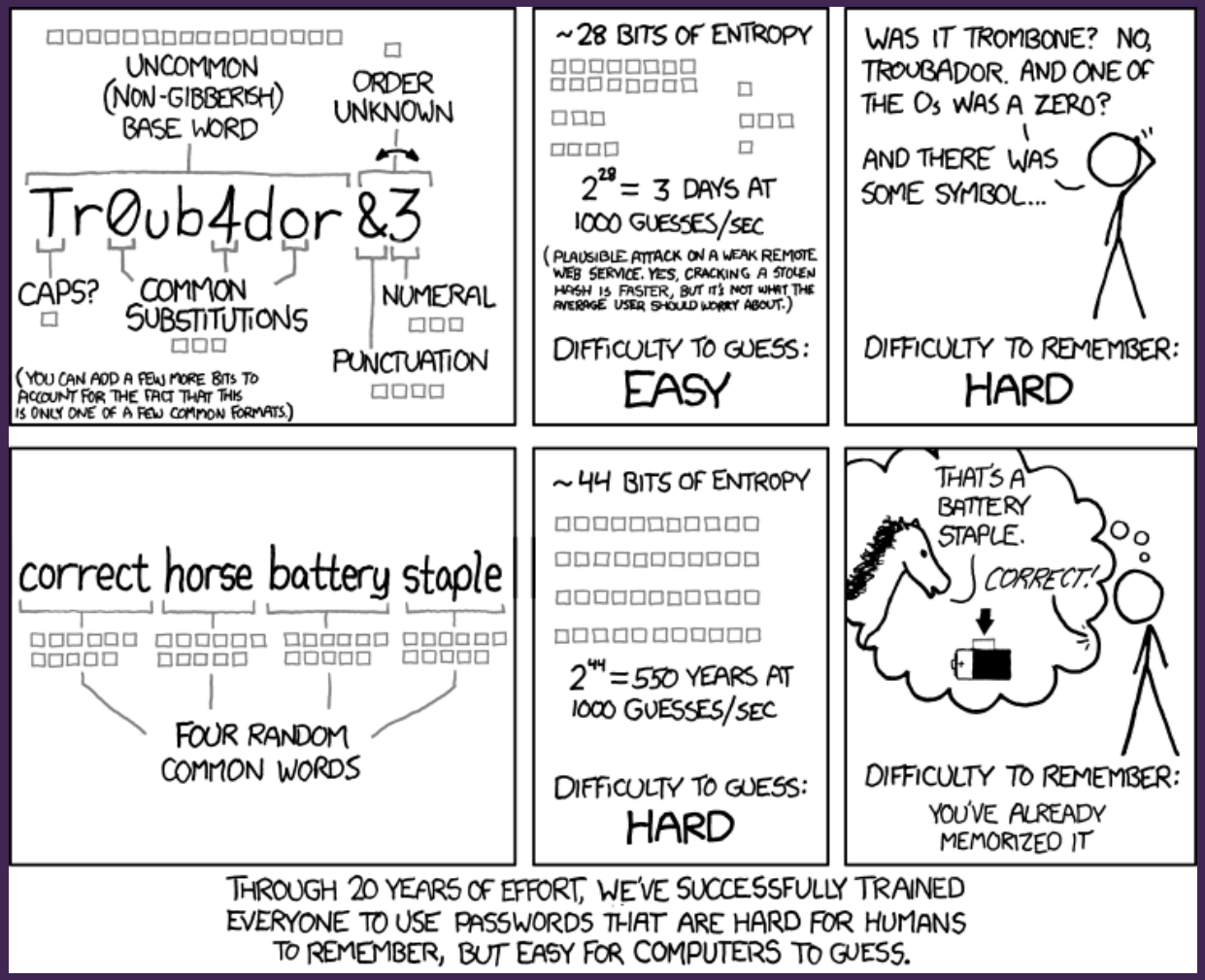
- Password-based User Accounts
  - NB
    - Hashes are calculated on the bytes that they have consumed, so use a new has every time

```
password = 'test'

simpleHash = hashlib.md5()
simpleHash.update(bytes(password,'utf-8'))
print("simple hash:"+simpleHash.hexdigest())

simpleHash.update(bytes(password,'utf-8'))
print("simple hash2:"+simpleHash.hexdigest())

simpleHash = hashlib.md5()
simpleHash.update(bytes(password,'utf-8'))
print("simple hash3:"+simpleHash.hexdigest())
```

```
Connected to pydev debugger (build 182.4505.26)
simple hash:098f6bcd4621d373cade4e832627b4f6
simple hash2:05a671c66aefea124cc08b76ea6d30bb
simple hash3:098f6bcd4621d373cade4e832627b4f6
```

    - Also, md5 is normally used to validate data integrity as a checksum rather than password hashing

- Password-based User Accounts
  - Making harder to crack passwords – approach 1



- Password-based User Accounts
  - Making harder to crack passwords – approach 2
    - Adding more data to the hash makes it harder to crack, but harder to remember
    - Let's help users by adding gibberish to their password and remembering the gibberish part
    - Salting

```
password = 'password'

simpleHash = hashlib.md5()
simpleHash.update(bytes(password,'utf-8'))
print("simple password hash:"+simpleHash.hexdigest())


salt = hashlib.md5()
salt.update(bytes('salty mcsalt-salt','utf-8'))
print("salt:"+salt.hexdigest())

s = hashlib.md5()
s.update(bytes(password,'utf-8')+salt.digest())
print("salted password hash:"+s.hexdigest())
```

```
simple password hash:5f4dcc3b5aa765d61d8327deb882cf99
salt:a4fee45f0a5af55c6007def3fb36ab0a
salted password hash:0602193f5656f39c18b29e24f42f2b4a
```

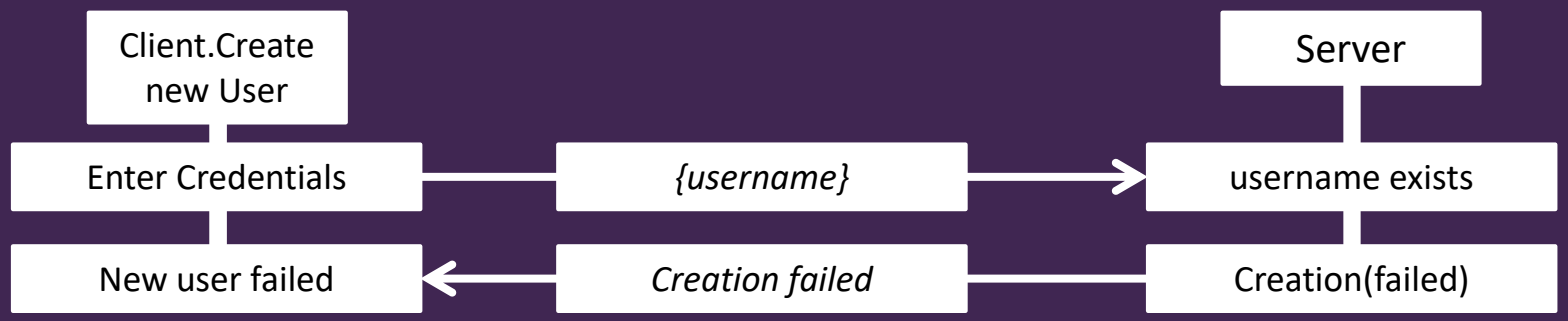
- Password-based User Accounts
  - Making harder to crack passwords – approach 2
  - We store
    - <username, (password hash, salt)>
    - Good practice to make salts individual
      - Not based on user name ;)
  - Can send hashes and salts around
    - They shouldn’t make it easier to reverse engineer the original password

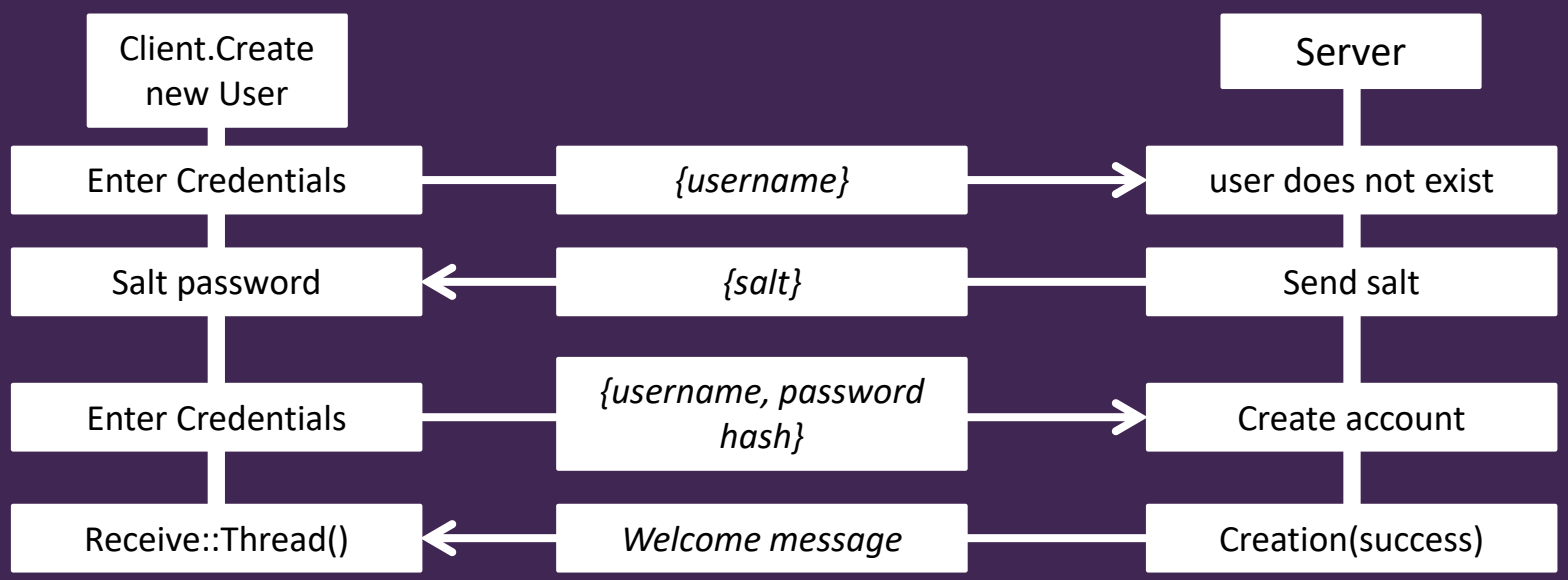
- Adding User Accounts to GaaS

- Adding User Accounts to GaaS
  - Need to think about the application in terms of:
    - Form, function and datacoms (like assignment 1)
    - db model & ui/ux

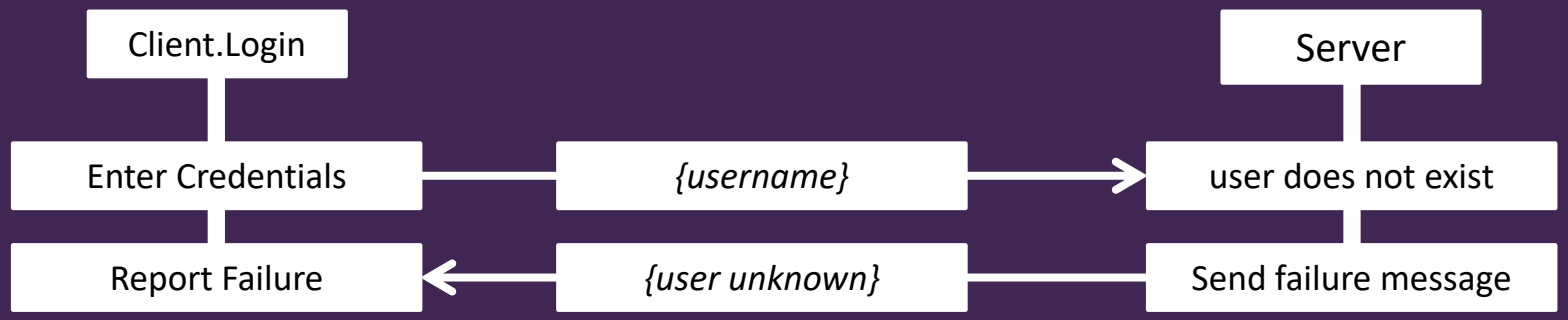
- Adding User Accounts to GaaS
  - Form
    - Should be no real changes
  - Function
    - States of the server
      - Client connected, but not logged in
      - Client connected and logged in
      - Client lost
    - States of the client
      - Not connected to server
      - Connected, but not logged in
      - Connected and logged in
      - Disconnected and logged in
      - Not connected and no account with server

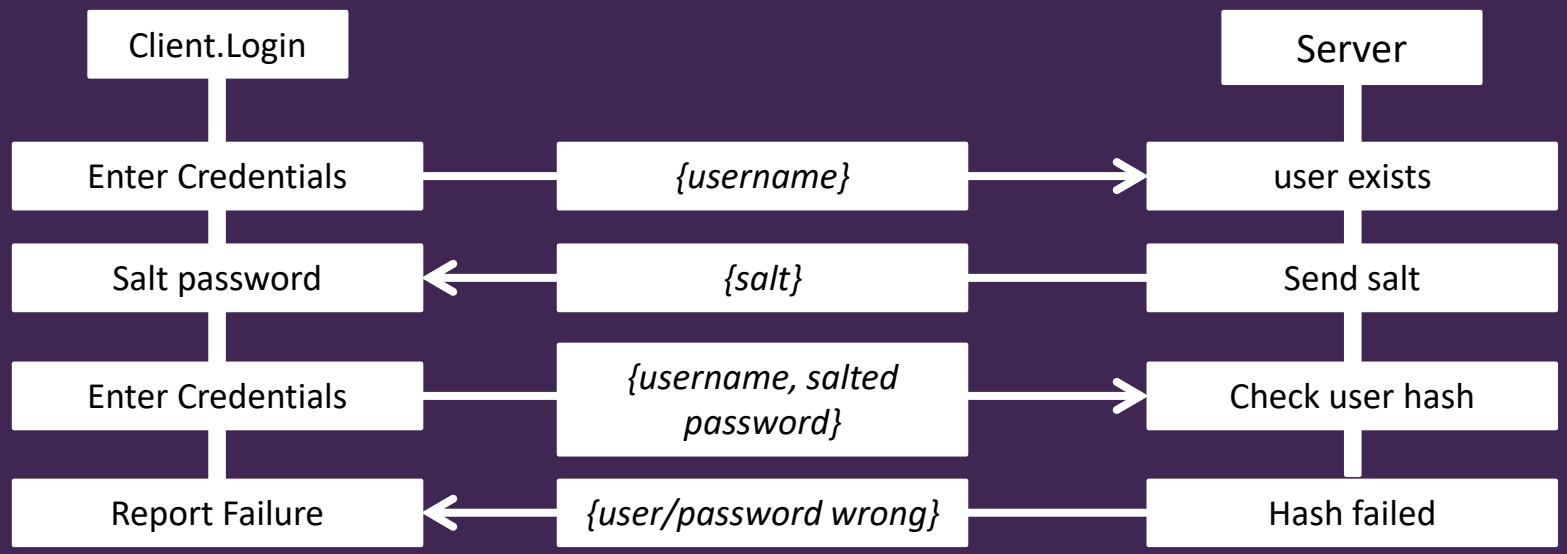
- Adding User Accounts to GaaS
  - Datacomms: New User



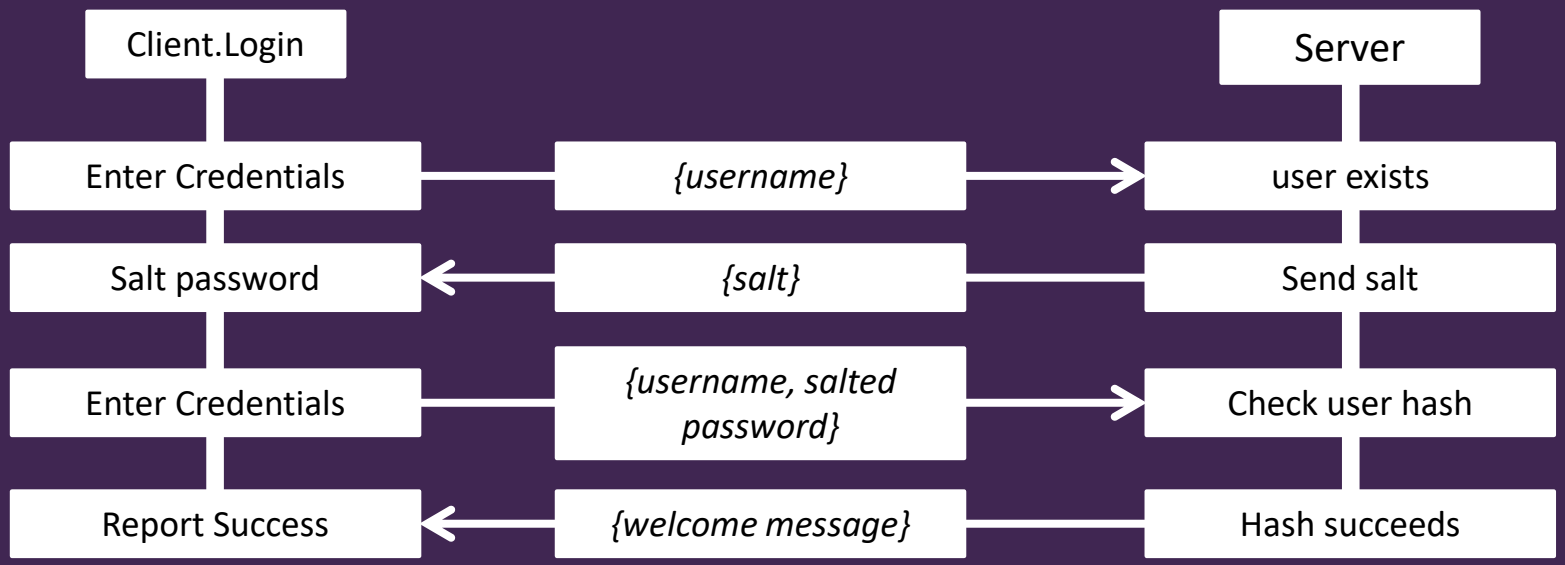


- Adding User Accounts to GaaS
  - Datacomms: Returning User





- Adding User Accounts to GaaS
  - Datacomms: Returning User

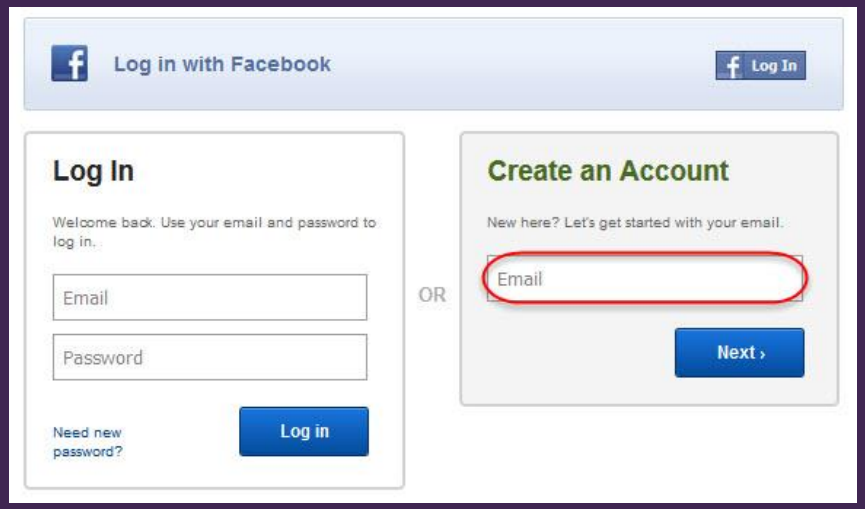


- Adding User Accounts to GaaS
  - Database model
    - Table for rooms
    - Table for users
      - User name (unique)
      - Hashed password
      - Salt
      - <other data>

- Adding User Accounts to GaaS
  - UI / UX
    - How do you manage the UX for creating accounts and logging into your service?
      - Use the existing command line interface of your MUD?

```
login as: username
Sent username "username"
username@server.it.uts.edu.au's password:
```

      - Add a modal dialog?



- Workshop preview
  - Add secure accounts to your MUD
  - Do in stages:
    - 1. Command line password tool
    - 2. Command line password too with SQL support
    - 3. Retrofit into SUD with appropriate UI
    - 4. Move functionality into MUD

- Questions