Users

Algorithm Generate Salted, Hashed Password

```
Require: $password$ (string)
  Generate random 16-byte salt as hexadecimal string
  Compute SHA-256 hash of $password$ concatenated with $salt$
  return $(salt, hashed\_password)$
```

Algorithm Create a New User

```
Require: $username$ (string), $password$ (string)
  $(salt, hashed\_password) \leftarrow$ Generate Salted, Hashed Password($password$)
  Create new User with $(username, salt, hashed\_password)$
  try
    Add user to database
    Commit transaction
  except:
    Rollback transaction
```

Algorithm Check User Password

```
Require: $username$ (string), $password$ (string)
  Retrieve User by $username$ from database
  if User not found then
    throw
  end if
  $(salt, stored\_hashed\_password) \leftarrow$ User's salt and hashed password
  Compute SHA-256 hash of $password$ concatenated with $salt$
  return computed_hash == stored_hashed_password
```

BattleModel Client-Side Caching

```
**Algorithm** Prep combatant
Require: $combatant\_data$ ({string: Any})
  Append $combatant$ id to $combatants$
  Add / update $combatant\_data$ to $meals\_cache$
  Add / update $combattant\_ttls$ with $time + TTL$
```

```
Algorithm Battle
Ensure: Two valid combatant ids in $combatants$
  for $combatant$ in $combatants$ do
    if $combatant$ id not in cache or has expired then
      Get $combatant$ data
      Add / update $combattant\_ttls$ with $time + TTL$
      Add / update $combatant\_data$ to $meals\_cache$
    end if
  end for
  Get combatant data from cache
  ...
```

KitchenModel Server-Side Caching

```
Algorithm Get meal by id
Require: meal_id (int), meal_name (string || None)
  Create cache_key from meal_id
  Lookup cache_key
  if cache_key found then
    meal_data ← Redis hash entry
         (note: decoded from binary to strings)
    Cast price to float
    Cast deleted cast to bool
    return $meal_data ata$
  end if
  Query db for meal_id
  meal_data ← resulting $Meals$ object cast to a dictionary
  Cache as Redis hash entry
       (note: we cast the values to strings, and redis will
        encode both keys and values in binary)
  return meal_data
```

```
Algorithm Get meal by name
Require: meal_name (string)
  Create cache_key from meal_name
  Lookup cache_key
  if cache_key found then
    meal_id ← Redis entry
    return Get meal by id
  end if
  Query db for meal_name
  Cache cache_key, meal_id pair in Redis (note: as strings)
  return Get meal by id
```

DB to Redis Write-Through Caching

This depends on SqlAlchemy sending events when the table is changed that Redis is looking for. I'm calling these "algorithms" for consistency. Is that appropriate? Shrug emoji

Algorithm Enable change tracking in SqlAlchemy

```
  Enable change tracking in SqlAlchemy
```

Algorithm Attach listeners to events

```
  Attach listener to after_update and after_delete events
```

Algorithm Update cache on change event

```
Require: target (Meal)
  Create cache_key from meal_id
  if target is now deleted then
    Delete cache_key from Redis
  else
    Update Redis hset for cache_key
         (note: we cast the Meal object to a dictionary and
          the values to strings. Redis will encode both keys
          and values in binary)
  end if
```

“Session” “Management”

Algorithm “Log in”

```
Require: user_id (int), battle_model (BattleModel)
  Lookup user_id in mongo
  if user_id is found then
    Clear current combatants from battle_model
    for combatant in db record do
      prep combatant
    end for
  else
    create record in mongo with empty combatants
  end if
```

Algorithm “Log out”

```
Require: user_id (int), battle_model (BattleModel)
  combatants_data ← combatants in battle_model
  Update record for user_id in mongo
  if user_id not found then
    throw
  end if
  Clear combatants in battle_model
```

Environment

Similarly I'm calling these "algorithms" for consistency.

Algorithm Container Dependency

```
Set app container to depend on Redis and Mongo containers
Set hostname and ports to match between containers
```

Algorithm Initialize Redis client

```
Get hostname / port / db from environment
Initialize Redis client
```

Algorithm Initialize Mongo client

```
Get hostname / port environment
Initialize Mongo client
Initialize db
```

Algorithm Initialize SqlAlchemy

```
Create db object
Initialize db and create tables
```