CLASS 5 - WEBSERVICES BASICS

- global data like `$_SESSION` and `$_FILE` should be handled in the service layers rather than CoreServiceLayer
- create a class in core, extend in, e.g., TeacherWebServiceLayer
  - but it's limiting as it requires the same arguments
  - so instead so can instantiate the core class in the presentation layer class and call its init function with the correct arguments

- WebServices Conventions

```
//Including a CoreServiceLayer class from a CoreServiceLayer class.
//Same for AbcMouseMarketingServiceLayer
include_once __DIR__ . '/../Namespace/Class.php';

//Including a CoreServiceLayer class from an extending repository.
include_once CORE_SERVICE_LAYER . 'Namespace/Class.php';

//Including a class from within your own repository (Excluding CoreServiceLayer see first case).
//(also Excluding AbcMouseMarketingServiceLayer see first case)
include_once 'Namespace/Class.php';

//Including a data access layer from CoreServiceLayer.
//First define the $data_access_library class member from within the constructor.
include_once __DIR__ . '/../Config/DataAccess.php';
$this->data_access_library = __DIR__ . '/../' . DataAccess::getDefault();
//Then include like the following.
include_once $this->data_access_library . 'Namespace/ClassDal.php';

//Including a data access layer from a non CoreServiceLayer repository.
include_once DATA_ACCESS_LIBRARY . 'Namespace/ClassDal.php';
```

Structure

- Namespace: Object
- Class: Behavior
- Interface: init()
- Examples:
  - `/User/GetAccountInfo/init`
  - `/Teacher/GetLessons/init`

Business objects and DALs

- each business object can have one data access layer (DAL) class
- DALs should be very dumb
- a business object can call another business object to retrieve its data
- a business object cannot call another DAL directly to retrieve its data. instead, instantiate the business object first.

Content Cache

- public content cache
  - any user facing data
  - CDN users can hit directly
  - images
  - assets
- private content cache
  - only interact with the data via the API
  - database values
  - config files
  - content that can be displayed in an admin tool

Platform Store

- contains data specific to "presentation" layer that would not be managed by a tool
- json file

- can be accessed by an endpoint like `/abc/Store/Get/init`
- each client has its own platform Store
- mounted drive whenever a server is spun up
- not platform agnostic (mobile vs desktop)
- NOT for store assets (clothes, items), only for things like "you've seen this popup that only occurs on desktop"

`/Resource/Enumerate`

- white pages for the front end, tells you where to find things
- to get things into /Resource/Enumerate, you need to edit `service_whitelist.php` and `api_endpoints.php`

WebServices

- each class should have one public function, called `init`
- each function should do one thing
- other than `init`, function names should start with `get`, `set`, `is` `do`, or `validate`, for example
  - `getAccountTypes`
  - `doCheckAuthReturn`
  - `isSubscribed`

Postman

- in BODY, set key `arguments` with value an array of the arguments
  - you must have the array brackets
  - you must use double quotes
- HOST needs https

Unit Testing

- `CoreServiceLayerTests` repo
- no support from TAPS or IT to tie the unit tests into anything
- write unit tests for everything new piece of code written