# Fast Path Notes

## 1. Cartridges and Controllers

- Storefront-reference-arhitecture is github starting point for SFRA implantation ( app_storefront_base cartridge and modules folder)
- There are already predefined cartridges for custom features such as: apple pay, wish list, compare products, in store pickup, gift registry, data download. You can add them in a cartridge path if you want to use some of those features.
- Cartridge path can be replaced by using require() function; (./, ../../, */, ~/, dw/catalog/CatalogMgr, 'server' ⇒ all of these are the ways how something can be included with the require() function )
- require('server') is refers to server.js file in module folder
- In order to extend controller we need to use `server.extend(module.superModule);`
- In order to extend model `var parentModel = module.superModule;`
- You can extend any module with the `module.superModule`
- next() executes next function in the middleware chain and then next append, prepend, replace, get, post etc.. in the controller.

## 2. Models, ISML templates, Server Side Scripts

- Model is the object that represents the data the controller sends to the view
- Model is serializable JSON object
- Model decorator is subset of the model that makes it easier to extend the model
- In order to extend the model we need to use module.superModule to identify the model we want to extend and we can use base.call() passing the same parameters that base model needs

- ISML - Internet Store Markup Language
- ISML tags that we can replace with the new B2C API or vanilla JS methods: iscookie(getCookie in cookie.js), iscontent (setContent()), isredirect (window.location.href = ..), isstatus (res.setStatus(404))
- Isactivedatahead, isactivedatacontext, isobject => tags that are used to collect active ( for instance isobject on pdp is used to collect active data about what products are viewed in the search results, product detail pages, and recommendation.

## 3. Forms, Transaction and Middleware events

- server.forms.getForm('formname') - this is how we get form data in controller
- Custom Object must be persisted in the database, you must use a Transaction. There are two flavours:
    1. Explicit transactions, you decide when to commit or rollback (Transaction.begin(), Transaction.commit(), Transaction.rollback())
    2. Implicit transactions, commits and rollbacks are automatically ( Transactio.wrap(function() {}) )

- Middleware events: route:BeforeComplete and route:AfterComplete. This is used to define when something needs to be executed, before or after everything in the middleware chain.

## 4. OCAPI

OCAPI documentation

Three API types: Shop, Data and Meta.

1. Https request and response in OCAPI (Requests => GET, POST, PUT, DELETE, PATCH, HEAD. OPTIONS; Responses=> so many, check documentation)
2. Get Client ID => client ID can be obtained from Account Manager. Include client ID in every request. It can be passed in several ways: Bearer Token, Request Parameter, HTTP Header (in the `x-dw-client-id` ). Client ID must be enabled in Account Manager.

3. URL Schema => Base URL which is this pattern

   `http(s)://public_domain/dw/api_type/ - for production instance`
   `http(s)://sub_domain.demandware.net/s/site_id/dw/api_type/ - non prod. Instance`

   Extended URLS which have more patterns

   *base_url/version_id/resource_type*

   *base_url/version_id/resource_type/identifier*

   base_url/*version_id/resource_type/action*

   *base_url/version_id/resource_type/identifier/relationship_type*

   *base_url/version_id/resource_type/identifier/relationship_type/relationship_type_identifier*

   *base_url/version_id/resource_type/identifier/relationship_type/action*

4. Resource State Framework => A resource state represents the server-side state of a specific resource e.g. a basket or a customer. It is a string token, baked on all the resource property information.

   OCAPI exposes resource state information via the *_resource_state* property in the response payload (body). The response contains either a single resource state or multiple resource states in case calling collection or search resources. In case of body-less responses (e.g. HEAD) the resource state is exposed via *x-dw-resource-state* response header.

   To use the resource state for optimistic locking you have to include the resource state from the last server response in your next state changing (DELETE, PATCH, POST, PUT) request. You should always prefer to use the *_resource_state* property in the body. In case the requested API doesn't have a body use the *x-dw-resource-state* header. Whenever a resource state is part of a client's request, OCAPI verifies it by comparing the given value with the server resource state. If both resource states are equal the operation is executed, otherwise an HTTP 409 *ResourceStateConflictException* fault is returned. In case of a create request

(e.g. with PUT) where the resource is not expected to be existing (with *not_exists* state), the returned fault is an HTTP 409 *ResourceAlreadyExistsException* instead.

5. Open Commerce API Settings => To configure OCAPI settings, perform the following steps:

1. In Business Manager: **Administration > Site Development > Open Commerce API Settings**.
2. In the **Select type** field, select the API type for the configuration.
3. In the **Select context** field, select the context for the configuration: Global for the configuration to affect all sites in the organization, or the name of a site for the configuration to affect only that site.
4. In the text field, edit the JSON document.
5. Click **Save**.

**Note:** You can override global settings for specific sites. Your settings are cached for up to three minutes until they become effective.

OCAPI JSON settings ex:


```
{
  "_v" : "20.4",
  "clients":
  [
    {

"allowed_origins":["http://www.sitegenesis.com","https://secure.sitegenesis.com"],
      "client_id":"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
      "response_headers":{"x-foo":"bar","P3P":"CP=\"NOI ADM DEV PSAi COM
NAV OUR OTR STP IND DEM\""},
      "resources":
      [
        {
          "resource_id":"/product_search",
          "methods":["get"],
          "read_attributes":"(**)",
          "write_attributes":"(**)",
          "cache_time":900,
          "version_range":{"from":"20.4"}
        },
        {
          "resource_id":"/products/*/bundled_products",
          "methods":["get"],
          "read_attributes":"(c_name,c_street)",
          "write_attributes":"(**)"
        },
        {
          "resource_id":"/baskets/*/items",
          "methods":["post"],
          "read_attributes":"(**)",
          "write_attributes":"(product_id, quantity)"
        }
      ]
    }
  ]
}
```