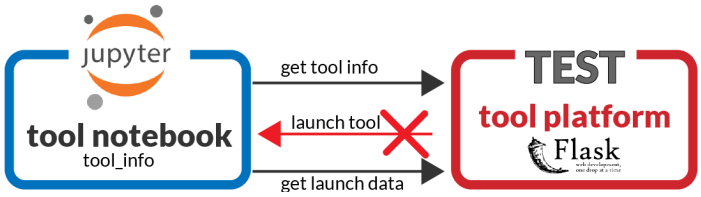
# LTI™ Advantage bootcamp notebook for Tool

claude.vervoort@gmail.com

*This bootcamp is still under development!*

## Introduction

The notebook shows how to interact with the LTI Advantage ecosystem from a tool implementer viewpoint. It interacts with an actual test server which has been built as a platform simulator to support this notebook.




## Limitation

The test tool platform cannot launch into the bootcamp. As a workaround, the test tool has APIs to get the launch data that would have been included in an actual launch.

In [1]:
```
# This notebook queries an actual tool platform test server. It needs its location:
platform_url='http://localhost:5000'
```

## Setup

### Import the python libraries needed by the tool

Here we just import the libraries that will be needed in this notebook, define some utility functions and constants.

In [2]:
```
import requests
import json
import jwt
import base64
import re
from time import time, sleep
from datetime import datetime
from cryptography.hazmat.primitives.asymmetric.rsa import RSAPublicNumbers
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.serialization import Encoding, PublicFormat
from IPython.display import display, Markdown, HTML, Javascript

def decode_int(b64value):
    return int.from_bytes(base64.urlsafe_b64decode(b64value), byteorder='big')

# for concise code, return full claim prefixed by ims
def fc(claim):
    return "http://imsglobal.org/lti/{0}".format(claim)

def md(mdt):
    display(Markdown(mdt))

md_buffer = ''

def mdb(mdt=None):
    global md_buffer
    if mdt:
        md_buffer = md_buffer + '\n' + mdt
    else:
        md(md_buffer)
        md_buffer = ''
```

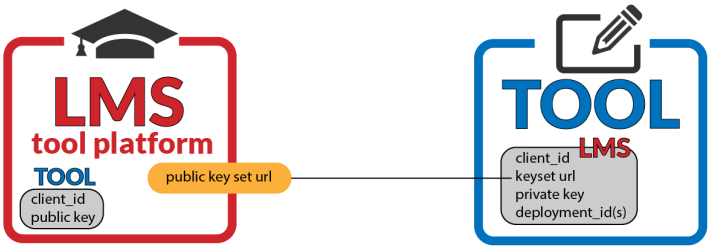
# Deploying tool and establishing keys

## get the tool deployment info to use in this notebook

First we need to get a new tool deployment from the server for this notebook instance to use. Each tool must have a `client_id` and a private key that will be used to interact with the platform services and send messages back to the platform. The `client_id` is used to for security purposes. A trust may be shared across multiple deployments of the same tool in a platform, so a `deployment_id` is also communicated to identify the actual deployment of that tool.

It also needs the keyset URL that exposes the platform public keys needed to validate the incoming messages.

While this information is required for each tool, how it is obtained by the tool is NOT currently part of the LTI specifications.



In [3]:
```
r = requests.get(platform_url + '/newtool')
tool_info = json.loads(r.text)

md('### Tool information')
md('Here is the tool information generated for this notebook. It is stored in ```tool
md('```json\n'+ json.dumps(tool_info, indent=4)+'````')
```

### Tool information

Here is the tool information generated for this notebook. It is stored in `tool_info` variable.


```
{
    "accesstoken_endpoint": "http://localhost:5000/auth/token",
    "client_id": "2",
    "keyset_url": "http://localhost:5000/.well-known/jwks.json",
    "webkey": {
        "alg": "RS256",
        "d": "BWC3cQhP0azuqU3-bwEliEqOS17dzFxd5Kvz_0EjuBqW4l9bufbny1s0g97SkZq9xTeT
IrkaXt4X4yd6tO-_vj-gx6JTpEHHg3TR_SGWLcheXexSBAC-eahPaATw4YMqOH7aUqEcX9p56pVTtab2Es
Dg1jPxeg-ymePQxIGual3hg02Nez_BL-cLXJYSLcQI8zw1JI0eJVr9QPedOeuX6HRA4C06pYArjJzvgsv5
qvouLEY-NDj2_clsWkfQhks-UhQhAHwIQdBEs6Raq8zoEVsrLBKT2P_CkiwI79iuw-F7nVFegnd9YYHttL
jDE6H8YS_KOTi-F0Z1wZZoxYLbAQ==",
        "e": "AQAB",
        "kty": "RSA",
        "n": "35vdvi4aXTA2SVuI8kLzKKVwI3zYwiwOkeG8d53y1FUNOyIK_RkN-8bqcAQT4m-Wvtg8
yxAfLgNRHVJ1muMCoBj2ioHDgyr6DksAc8pKH4tMhRQ2FYIz4IKKmOSduovMuoYFRpZmpKtCyoW3_Z5XY7
DH85D2Hsec23EtgK6WkhIKrIKeuBQnnsZyAsHL_vHQy0PDgFPiRpodVq3s6dPksCsUV5kK9UQMm-XGabJC
hqNpzM6_s7E6EtB4WdBii_hXrMWtXew5HCEeCE18JuA_59LwQILo2y7DVIOGmbjmvp3Haj393rBXSkMGAj
5OKSfhplpYSti7gHm7y7c6j5OApQ==",
        "use": "sig"
    },
    "webkeyPem": "-----BEGIN RSA PRIVATE KEY-----\nMIIEogIBAAKCAQEA35vdvi4aXTA2SVu
I8kLzKKVwI3zYwiwOkeG8d53y1FUNOyIK\n/RkN+8bqcAQT4m+Wvtg8yxAfLgNRHVJ1muMCoBj2ioHDgyr
6DksAc8pKH4tMhRQ2\nFYIz4IKKmOSduovMuoYFRpZmpKtCyoW3/Z5XY7DH85D2Hsec23EtgK6WkhIKrIK
e\nuBQnnsZyAsHL/vHQy0PDgFPiRpodVq3s6dPksCsUV5kK9UQMm+XGabJChqNpzM6/\ns7E6EtB4WdBii
/hXrMWtXew5HCEeCE18JuA/59LwQILo2y7DVIOGmbjmvp3Haj39\n3rBXSkMGAj5OKSfhplpYSti7gHm7y
7c6j5OApQIDAQABAoIBAAVgt3EIT9Gs7qlN\n/m8BJYhKjkte3cxcXeSr8/9BI7galuJfW7n258tbNIPe0
pGavcU3kyK5Gl7eF+Mn\nerTvv74/oMeiU6RBx4N00f0hli3IXl3sUgQAvnmoT2gE8OGDKjh+2lKhHF/ae
eqV\nU7Wm9hLA4NYz8XoPspnj0MSBrmpd4YNNjXs/wS/nC1yWEi3ECPM8NSSNHiVa/UD3\nnTnrl+h0QOA
tOqWAK4yc74LL+ar6LixGPjQ49v3JbFpH0IZLPlIUIQB8CEHQRLOk\nWqvM6BFbKywSk9j/wpIsCO/YrsP
he51RXoJ3fWGB7bS4wxOh/GEvyjk4vhdGdcGW\naMWC2wECgYEA36Zyf+9q2Cq8SAY5/hefPUqxKJhZaBY
LIsThJzjlGsn62R+GIbAs\nLU3tyPnu5UKbT+rq7qbz9MgOHrX8YVqfycvmRY0A4Bnp1ojzsaedwagFFV1
ZPU9Z\nVfI0Zjc6mrO64UV7QqwKHC4QJ7V+ISB0NYc7mAbjil6pCMF96h9arIECgYEA//Pj\nb5DHH+uK8
l0ZiFxIsAtOrfLKiBS1bWg3BGgQFeBsYZajmfjQIQLoBpNV0XUbYKaD\nIF7Cwhj25bGNcCJ31BKN26E3d
MDWh2yOHtJg99vS273R5ThAXnrDWNFTPN+B073K\nGF87yizL4wop+IIngJ6JozezXH7D1giZkhGXkiUCg
YBZdtPGqZcr8axvg04ffoOM\nmRxVCNx67pZ1yhvm/LNpC/L0d8/IGhkR5mKASrptqz9FsMtZvB9Kg9xiH
Jpgt0fE\nCXSLzePTwYnw78fMASjyFqwx562TAPLTlpeva9hdmhfflVPBT+CbdTGxMMDLr5fM\n8dO89MS
AzW0Au6YKyZAUAQKBgAH65giH6zJxBQso8zid1iYegEZSOYTh/CFTjBFp\nqK4ypUQAAVYAmcOXnSnn2+M
Z79NIln0anpEX1lkijYCPk7O9TQGk9qPdlgtIKLVO\nVVe1iKUQi94lGEJi2r4GKImxBPUZY8XafsDqpc0
k1/xHLX0OPOxZUvblOCigumdp\n4W1NAoGAabfPRr5kpB5VT1A2siNGo1RgZax11LrNq9kQVmJLV51wtx9
M2knr/QE7\ngy9JKcIs2hIZdup4AP/O20d0NM8Y8Ha6niq1C8k0ComZdX2Qp3Dq6+ZMl4KiHFGb\nSCF9o
eeJ+1IRnUpGqbgzZvkE3t9oxYZBzKuEkFTQYALlNkEZp9I=\n-----END RSA PRIVATE KEY-----"
}
```

### Getting the public keys from the platform

In order to validate the various messages we will receive from the platform, we need to get the key sets. We'll also transform the keys to the `PEM` format that is used by the `jwt` module to decode the messages.

The test platform exposes its keys in a keyset format at a well-know location (.well-known/jwks.json). Other platform might just communicate the keyset url as part of the tool deployment information.

In [4]:
```
keyset = json.loads(requests.get(tool_info['keyset_url']).text)

md('#### Platform keyset')
md('```json\n'+ json.dumps(keyset, indent=4)+'````')

platform_keys = {}

# let's transform as a map for ease of use, and just the PEM because this is what is
for key in keyset['keys']:
    public_key = RSAPublicNumbers(decode_int(key['e']), decode_int(key['n'])).public_
    pem = public_key.public_bytes(Encoding.PEM, PublicFormat.SubjectPublicKeyInfo)
    platform_keys[key['kid']] = pem
```

**Platform keyset**


```
{
    "keys": [
        {
            "alg": "RS256",
            "e": "AQAB",
            "kid": "1519880184_0",
            "kty": "rsa",
            "n": "pUZ5hP10NYgpRpW9ziOkcSh0HhaGdpuEKbyvryUx8-yMBv2FjYH2HoVQQ_aerVBV
o8xDzf7UHylzkAE3it_zNWRTaez3_KripRvviE9DtC_6Ah_lSbq_-nRKCiYmNQPjOQOHwOrGojPxeK95UE
Ey6oBHSzdixgoHdDbkyRgbK6rvnEH-4-cC4jmU1gfVb7SRG8Dxop7fLvyO7VFaGjeC1hFrMuvt6MikIuZ6
eR-ueczmEs9Rt0ZU2IcLGkT3hQ0np991_Qe5zgtXdQ58GoPejQHbTxYSDv8eJiAljrZ1lQkqCduXclygOk
BN3GzyAz1KddVJEYvOeUSdX-Qf0miAWQ==",
            "use": "sig"
        },
        {
            "alg": "RS256",
            "e": "AQAB",
            "kid": "1519880184_1",
            "kty": "rsa",
            "n": "lJPW0l-x0t1HlSDMN5TOsRGfTl4FXfo-T-_BQPgx9EHwNVVG8f1j3pzbtkWLtyHD
5o8JQ7JJPc6XLv5NtMlNGEggshI9m0kIiA021EoJm2UttJ8yMziVd50HEqZgXOAiJsRdsp-CZzLfgkIjDm
hz67BBfZIpuUwlHDQC5AscLqWRRsyBRTKwX-HM1p4zGGruuXoSgaQkiyZRiWycr62_2Q3mpBi4mHeurilz
HYZVAJyiBnneb-hrm8YZzveFZ2mdk-jQ_xAEy6I8U0rPyyzfXbnvmZHI_oNKVVQexLOBSAtq1Wh-fVxHoC
uBWz0KUTKWNJ6vZj_txmOebRB7N2WYow==",
            "use": "sig"
        },
        {
            "alg": "RS256",
            "e": "AQAB",
            "kid": "1519880184_2",
            "kty": "rsa",
            "n": "wwwfkjS2V6hRx7cJhQT558ZBAx0zQzhlK9Gv3YjW96pNo0uZ2tOw5zuhq_HyeHGc
73T3ZhiANpOrq4GEg3uU2W3VKsywO21fwYiXRZrhl5a7yDywtKs29jG4xXnZLsLg6nFeqdx0dY5di00rfk
ThO1Ros6xUm6U7P2dNFseXNc-3WLAHoLzHg95r4Wnp-yVOspFhqYtOJ2DpZ3VVgjvxJMHhFpzMP53PVpMI
baM_8sfI3xl3EGQnMPGJabziSg5Fct_hTijCKeIAXy1tydBp0lSo3EhP18ew_UCtXxEF-DLupsoC3BWAT8
ThF2XnJsy44vhjPLcJ1xB5oondIIaxTQ==",
            "use": "sig"
        },
        {
```

# Deep Linking - Creating a Link

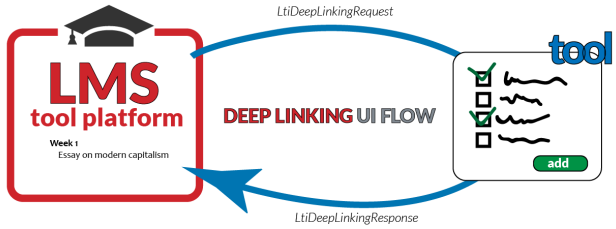
This section will use the deep linking specification to create a Resource Link to the platform. That resource link will be gradable and used in the following sections of that notebook.

Deep Linking (https://www.imsglobal.org/specs/lticiv1p0) is a **UI flow**, it is an important piece that is sometimes missed on 1st reading. The user is redirected from the platform to the tool to pick or create one or multiple piece of content (often, LTI links), and the the tool redirects the UI back to the tool with the actual selection (or an empty payload if nothing was picked or created).

So there are 2 messages:

1. `LtiDeepLinkingRequest` from the platform to the tool to start the picking/create session. This is a typical platform launch that contains the context and the user information, and what kind of content items may be created in this flow (for example, this flow might indicate it only wants LTI links).
2. `LtiDeepLinkingResponse` from the tool back to the plaform using the `content_item_return_url` provided in the request.

Once a tool is added to a course, usually the first launch from the platform will be a Deep Linking request.




## Setup: Getting a Deep Linking Request

This notebook is not a tool actually launched by the platform, so the test platform as way to give us the token that it would include in an actual HTTP POST request, so we can build a mock POST request including the parameter `post_data`.

In [5]:
```
r = requests.get("{}/tool/{}/cisr".format(platform_url, tool_info['client_id']))

post_data = {
    'id_token':r.text
}

md('`id_token='+ r.text+'`')
```

`id_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjE1MTk4ODAxODRfMSJ9.eyJodHRwO JixwysMcpbLjsuL_dHHw5PVa1k3pn_E8_02PXz9ukYQW3yAYgJnnf8QZyDU_Bt4bzJI4VbvXYMEDH7r8_VLI o_FPwfjbWn06sG_j9- nrKkmb68zyEDADzBTR74yfL5h3yRlcAcWAabz9xkTABVHA4WvIokfh79VaFHCwEEg_8gQpnvb4_bbdG09LkQ`

## Task 1: Verify the JWT is properly signed

The first thing before to display to the user the picker/authoring interface to create the link is to validate this request is properly signed. This is done by decoding the JWT using public key from the platform.

In [6]:
```
# Let's get the kid so we can get the proper public key

# should check ISS first to lookup keyset

encoded_jwt = post_data['id_token']
jwt_headers = jwt.get_unverified_header(encoded_jwt)

content_item_message = jwt.decode(encoded_jwt,
                                  platform_keys[jwt_headers['kid']],
                                  jwt_headers['alg'],
                                  audience = tool_info['client_id'])

md('#### Message properly signed! Decoded ContentItemSelectionRequest message:')
md('```json\n'+ json.dumps(content_item_message, indent=4)+'```')
```

**Message properly signed! Decoded ContentItemSelectionRequest message:**


```
{
    "http://imsglobal.org/lti/deep_linking_request": {
        "accept_media_types": [
            "application/vnd.ims.lti.v1.ltilink"
        ],
        "accept_presentation_document_targets": [
            "iframe",
            "window"
        ],
        "accept_multiple": true,
        "auto_create": true,
        "data": "op=321&v=44"
    },
    "iat": 1519912812,
    "exp": 1519912872,
    "nonce": "dc604fc8-1d58-11e8-8172-f40f243530c8",
    "iss": "http://localhost:5000",
    "aud": "2",
    "http://imsglobal.org/lti/deployment_id": "deployment_2",
    "http://imsglobal.org/lti/message_type": "LTIDeepLinkingRequest",
    "http://imsglobal.org/lti/version": "1.3.0",
    "http://imsglobal.org/lti/launch_presentation": {
        "document_target": "iframe",
        "return_url": "http://localhost:5000/tool/dc4e7fbe-1d58-11e8-b0c7-f40f2435
30c8/cir"
    },
    "sub": "LTIBCU_1",
    "given_name": "Peggie",
    "family_name": "Nikole",
    "name": "Peggie Nikole",
    "email": "Peggie.Nikole@example.com",
    "http://imsglobal.org/lti/roles": [
        "http://purl.imsglobal.org/vocab/lis/v2/membership#Instructor"
    ],
    "http://imsglobal.org/lti/context": {
        "id": "dc4e7fbe-1d58-11e8-b0c7-f40f243530c8",
        "label": "LTI Bootcamp Course",
        "title": "LTI Bootcamp Course",
```

### Task 2: extract the information needed to render the selector/authoring UI

If this is the first launch for the user or the course, as a tool you may prompt the user for setup information, including account linking or course setup. Ultimately the user will see the authoring or picking interface that will allow her to create or select the content items to be added to the course.

Some key attributes of the `ContentItemSelectionRequest` will drive the experience:

In [7]:
```
# fc(claim) prefix the claim with http://imsglobal.org/lti/
mdb('1. What is the current course id? {0}'.format(content_item_message[fc('context')
mdb('1. What is the current user id? {0}'.format(content_item_message['sub']))
is_instructor = len(list(filter(lambda x: 'instructor' in x.lower(), content_item_mes
mdb('1. Is this user an instructor? {0}'.format(is_instructor))
deep_linking_claim = content_item_message[fc('deep_linking_request')]
mdb('1. What kind of content item can be created? {0}'.format(deep_linking_claim['acc
mdb('1. Can I return more than one items to be added? {0}'.format(deep_linking_claim[
mdb('1. Will the user be prompted before to actually save the items? {0}'.format(not
deep_linking_return_url = content_item_message[fc('launch_presentation')]['return_url
mdb('1. Where should I redirect the browser too when done? {0}'.format(deep_linking_r
mdb('1. Is there any data I must pass back to platform when I return? {0}'.format('da
mdb()
```

1. What is the current course id? dc4e7fbe-1d58-11e8-b0c7-f40f243530c8
2. What is the current user id? LTIBCU_1
3. Is this user an instructor? True
4. What kind of content item can be created? ['application/vnd.ims.lti.v1.ltilink']
5. Can I return more than one items to be added? True
6. Will the user be prompted before to actually save the items? False
7. Where should I redirect the browser too when done? http://localhost:5000/tool/dc4e7fbe-1d58-11e8-b0c7-f40f243530c8/cir (http://localhost:5000/tool/dc4e7fbe-1d58-11e8-b0c7-f40f243530c8/cir)
8. Is there any data I must pass back to platform when I return? True

### Task 3: building the response token

After the end of the interaction, so user is sent back to the platform throught a browser redirection using an HTTP POST containing the JWT `ContentItemResponse` message. In this case, we will return 2 LTI links, one being graded (since the request supports multiple content items).

Here we're creating the actual response token.

In [8]:
```
## First let's create our 2 content items
## Note that the URLs are phony as for now there is now way to launch in the notebook
simple_link =  {
      "mediaType": "application/vnd.ims.lti.v1.ltilink",
      "url": "http://lti.bootcamp/item111",
      "presentationDocumentTarget": "iframe",
      "title": "A simple content item",
      "text": "Some long text",
      "icon": {
        "url": "http://lti.example.com/image.jpg",
        "width": 100,
        "height": 100
      },
      "custom": {
        "lab": "sim4e"
      }
}

assignment_link =  {
      "mediaType": "application/vnd.ims.lti.v1.ltilink",
      "url": "http://lti.bootcamp/item111",
      "presentationDocumentTarget": "iframe",
      "title": "An assignment",
      "text": "Chemical lab sim",
      "icon": {
        "url": "http://lti.example.com/image.jpg",
        "width": 100,
        "height": 100
      },
      "custom": {
        "lab": "sim3a",
        "level": "easy"
      },
      "lineItem": {
        "scoreMaximum": 34,
        "label": "Chemical lab sim",
        "resourceId": "sim3a",
        "tag": "final_grade"
      }
}

now = int(time())

deep_linking_response = {
    "iss": tool_info['client_id'],
    "aud": content_item_message['iss'],
    "exp": now + 60,
    "iat": now,
    "http://imsglobal.org/lti/message_type": "DeepLinkingResponse",
    "http://imsglobal.org/lti/version": "1.3.0",
    "http://imsglobal.org/lti/content_items": [
        simple_link, assignment_link
    ]

}
```

### Task 4: build the signed JWT

In [9]:
```
deep_linking_response_token = jwt.encode(deep_linking_response, tool_info['webkeyPem'

print(deep_linking_response_token)
```

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiIyIiwiYXVkIjoiaHR0cDovL2xvY2FsaG9zd
Do1MDAwIiwiZXhwIjoxNTE5OTEyODcyLCJpYXQiOjE1MTk5MTI4MTIsImh0dHA6Ly9pbXNnbG9iYWwub3J
nL2x0aS9tZXNzYWdlX3R5cGUiOiJEZWVwTGlua2luZ1Jlc3BvbnNlIiwiaHR0cDovL2ltc2dsb2JhbC5vc
mcvbHRpL3ZlcnNpb24iOiIxLjMuMCIsImh0dHA6Ly9pbXNnbG9iYWwub3JnL2x0aS9jb250ZW50X2l0ZW1
zIjpbeyJtZWRpYVR5cGUiOiJhcHBsaWNhdGlvbi92bmQuaW1zLmx0aS52MS5sdGlsaW5rIiwidXJsIjoia
HR0cDovL2x0aS5ib290Y2FtcC9pdGVtMTExIiwicHJlc2VudGF0aW9uRG9jdW1lbnRUYXJnZXQiOiJpZnJ
hbWUiLCJ0aXRsZSI6IkEgc2ltcGxlIGNvbnRlbnQgaXRlbSIsInRleHQiOiJTb21lIGxvbmcgdGV4dCIsI
mljb24iOnsidXJsIjoiaHR0cDovL2x0aS5leGFtcGxlLmNvbS9pbWFnZS5qcGciLCJ3aWR0aCI6MTAwLCJ
oZWlnaHQiOjEwMH0sImN1c3RvbSI6eyJsYWIiOiJzaW00ZSJ9fSx7ImllZGlhVHlwZSI6ImFwcGxpY2F0a
W9uL3ZuZC5pbXMubHRpLnYxLmx0aWxpbmsiLCJ1cmwiOiJodHRwOi8vbHRpLmJvb3RjYW1wL2l0ZW0xMTE
iLCJwcmVzZW50YXRpb25Eb2N1bWVudFRhcmdldCI6ImlmcmFtZSIsInRpdGxlIjoiQW4gYXNzaWdubWVud
CIsInRleHQiOiJDaGVtaWNhbCBsYWIgc2ltIiwiaWNvbiI6eyJ1cmwiOiJodHRwOi8vbHRpLmV4YW1wbGU
uY29tL2ltYWdlLmpwZyIsIndpZHRoIjoxMDAsImhlaWdodCI6MTAwfSwiY3VzdG9tIjp7ImxhYiI6InNpb
TNhIiwibGV2ZWwiOiJlYXN5In0sImxpbmVJdGVtIjp7InNjb3JlTWF4aW11bSI6MzQsImxhYmVsIjoiQ2h
lbWljYWwgbGFiIHNpbSIsInJlc291cmNlSWQiOiJzaW0zYSIsInRhZyI6ImZpbmFsX2dyYWRlIn19XX0.T
zttHGAqgU3H1xPTUa8cBV_9l1mcWV7iCuD6Pauqp-G5iVFajIFwl-AvEefHdGhHaTvxDACgbt8A3x3RQxX
SYZr0UsFwHsayyopoG3GuXkQhN8HIO5tw-GAUu9k-z6j3sfbnX8VZhfNw8-JG9YH3TmtmMMhh0St-Pw0rt
Bqc79zYkVyYsAuKPghcGz6Wd5baiLf0W1SBpfTQjjl4HNxPUtGUM6G7pyw1aq7x6ONcn43AO3Xb-ZfhPMi
FpemgCjSDWcBGjYKqefrtWlsKWa-tuouT7tmq0snHdeayasNl8iacTKhLQ-xJennj_h_eBSdSURkIL6Ogh
PlyPODb9-COPA

### Task 5: redirect the user back to the platform with the content item selection

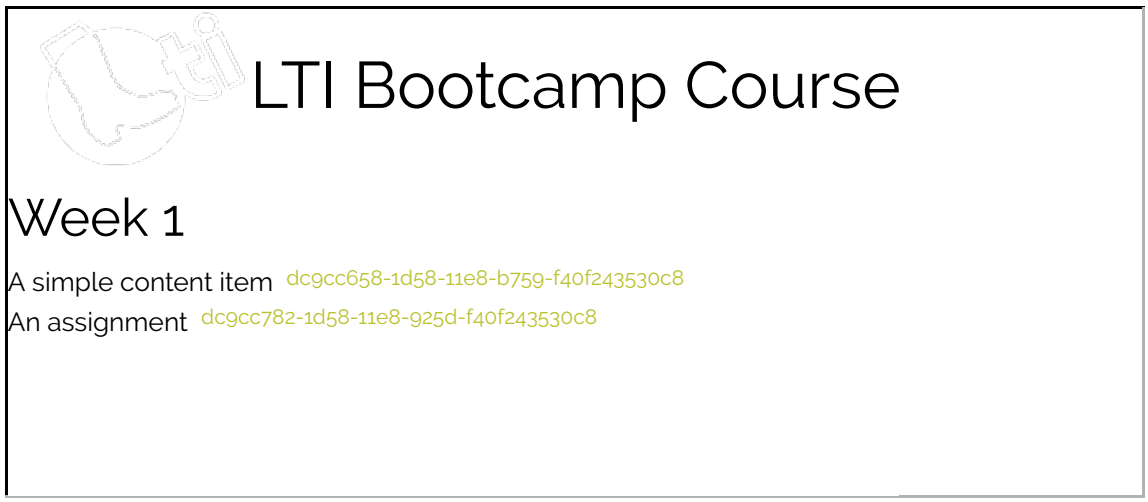
Now that we the response token, let's do the actual HTML POST redirection to the platform. Note that because the platform supports `autocreate` there will be no prompt. The 2 items will be added directly to the course.

In [10]:
```
# Let's start by adding the JWS security claims
content_item_response = {
    'iss': tool_info['client_id'] ,
    'aud': content_item_message['iss']
}

autosubmit_js = """
                var ltiForm = $('<form>');
                ltiForm.attr('action', '{url}');
                ltiForm.attr('method', 'POST');
                ltiForm.attr('target', 'deeplinking_frame');
                $('<input>').attr({{
                    type: 'hidden',
                    name: 'jws_token',
                    value: '{token}'
                }}).appendTo(ltiForm);
                $('#deeplinking_frame').before(ltiForm);
                ltiForm.submit();
                ltiForm.remove();
                """

autosubmit_js = autosubmit_js.format(url=deep_linking_return_url, token=deep_linking_

display(HTML('<iframe id="deeplinking_frame" name="deeplinking_frame" style="height:
display(Javascript(data=autosubmit_js,
                   lib="https://code.jquery.com/jquery-3.3.1.min.js"))
```




## Student Resource Link launch

Now that we have created resource links, let's handle a student launch from one of them. We're going to use a resource link with a **coupled** line item, so that we can use it to send a score back to the platform.

### Setup

The first thing we need, as with deep linking, is to get from the test platform the launch token which an actual tool would get in an actual HTML Form Post.

In [11]:
```
# in automated run, force a sleep for the response above to have been processed
sleep(5)

# select an id from the ones displayed in the course platform in the IFrame above
# if not specified the platform will pick a resource to use

resource_link_id = ''
context_id = content_item_message['http://imsglobal.org/lti/context']['id']

r = requests.get("{}/tool/{}/context/{}/studentlaunch?rlid={}".format(platform_url,
                                                                      tool_info['client_id'],
                                                                      context_id,
                                                                      resource_link_id))

post_data = {
    'id_token':r.text
}

md('`id_token='+ r.text+'`')
```

`id_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjE1MTk4ODAxODRfMCJ9.eyJpYXQiO0AVdKc1n-gT-B1vVOaLcUMfOayPxWpFqPE4axh8FdxoB3Li-f3K7oxgG2u_uqgiGp8HOQ`

### Task 1: Decode the launch

Now, same as with the Deep Linking request, we decode the token:

In [12]:
```
encoded_jwt = post_data['id_token']
jwt_headers = jwt.get_unverified_header(encoded_jwt)

student_launch = jwt.decode(encoded_jwt,
                            platform_keys[jwt_headers['kid']],
                            jwt_headers['alg'],
                            audience = tool_info['client_id'])

md('```json\n'+ json.dumps(student_launch, indent=4)+'```')
```


```
{
    "iat": 1519912818,
    "exp": 1519912878,
    "nonce": "dff7ea58-1d58-11e8-bca0-f40f243530c8",
    "iss": "http://localhost:5000",
    "aud": "2",
    "http://imsglobal.org/lti/deployment_id": "deployment_2",
    "http://imsglobal.org/lti/message_type": "LTIResourceLinkLaunch",
    "http://imsglobal.org/lti/version": "1.3.0",
    "http://imsglobal.org/lti/launch_presentation": {
        "document_target": "iframe",
        "return_url": "http://localhost:5000http://localhost:5000/"
    },
    "sub": "LTIBCU_15",
    "given_name": "Gaius",
    "family_name": "Baltar",
    "name": "Gaius Baltar"
```

## Task 2: extract information to show the correct activity

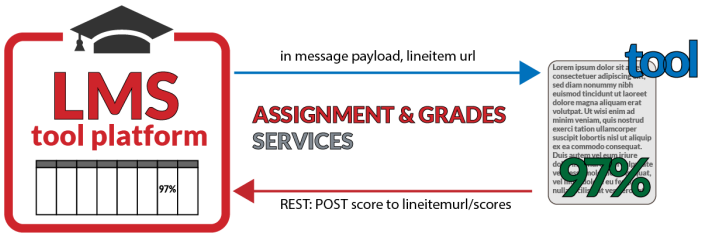
The launch gives information about the user, her role, the course, but also which actual resource we want to launch into.

In [13]:
```
# fc(claim) prefix the claim with http://imsglobal.org/lti/
mdb('1. Is this a resource link launch? {0}'.format(student_launch[fc('message_type')
mdb('1. What is the id of the resource link that is launched? {0}'.format(student_lau
mdb('1. What is the name of the resource that is launched? {0}'.format(student_launch
mdb('1. What is the current course id? {0}'.format(student_launch[fc('context')]['id'
mdb('1. What is the current user id? {0}'.format(student_launch['sub']))
is_learner = len(list(filter(lambda x: 'learner' in x.lower(), student_launch[fc('rol
mdb('1. Is this user a student? {0}'.format(is_learner))
return_url = student_launch[fc('launch_presentation')]['return_url']
mdb('1. Where should I redirect the browser too when done? {0}'.format(return_url))
mdb('1. Which lab should be displayed? {0}'.format(student_launch[fc('custom')]['lab'
ags_claim = student_launch[fc('ags')]
mdb('1. Is there a gradebook column for this resource? {0}'.format('lineitem' in ags_
mdb()
```

1. Is this a resource link launch? True
2. What is the id of the resource link that is launched? dc9cc782-1d58-11e8-925d-f40f243530c8
3. What is the name of the resource that is launched? An assignment
4. What is the current course id? dc4e7fbe-1d58-11e8-b0c7-f40f243530c8
5. What is the current user id? LTIBCU_15
6. Is this user a student? True
7. Where should I redirect the browser too when done? http://localhost:5000http://localhost:5000/ (http://localhost:5000http://localhost:5000/)
8. Which lab should be displayed? sim3a
9. Is there a gradebook column for this resource? True

## Assignment and Grade Services

Now that the student has launched into a grading activity, eventually she will complete it. Let's assume this is an autograded quiz. At the end of the interaction, we're going to send a score.



### Step 1: Get an access token

To be able to send a grade, or call any service on that matter, we must first get an access token. This is done by using a JWT based client grant flow RFC-7523 (https://tools.ietf.org/html/rfc7523).

Here we will re-use the token for the rest of the notebook, so we don't specify scope. If you intend to use the token only for a given operation, it is a good practice to scope it accordingly.

The grant type is client_credentials (https://tools.ietf.org/html/rfc6749#section-1.3.4) as the trust is established between the tool and the platform. The current user and context are not considered.

In [14]:
```
## Let's define a function we can re-use for other calls

def get_token(scope):
    token_endpoint = tool_info['accesstoken_endpoint']

    now = int(time())

    assertion = {
        "iss": tool_info['client_id'],
        "aud": token_endpoint,
        "exp": now + 60,
        "iat": now,
        "jti": "{0}-{1}".format(tool_info['client_id'], now)
    }

    assertion_jwt = jwt.encode(assertion, tool_info['webkeyPem'], 'RS256').decode()

    return json.loads(requests.post(token_endpoint, data = {
        'client_assertion': assertion_jwt,
        'grant_type': 'client_credentials',
        'scope': scope,
        'client_assertion_type': 'urn:ietf:params:oauth:client-assertion-type:jwt-bea
    }).text)

token_info = get_token('https://imsglobal.org/lti/ags/score https://imsglobal.org/lti

md('#### Access token:')
md('```json\n'+ json.dumps(token_info, indent=4)+'```')

# We'll also need to create a proper header, so let's also create a function for that
def add_authorization(headers, access_token):
    b64token = base64.b64encode(access_token.encode('utf-8')).decode()
    headers.update({'Authorization': 'Bearer {0}'.format(b64token)})
```

**Access token:**


```
{
    "access_token": "tke005518c-1d58-11e8-b19d-f40f243530c8",
    "expires_in": 3600,
    "token_type": "Bearer"
}
```


### Step 2: Publish a score

In [15]:
```
# Scores in the subpath scores from lineitem.
def append_to_path(path, subpath):
    p = re.compile('lineitem($|\?|#)')
    return p.sub('lineitem/{0}\\1'.format(subpath), path)

scores_url = append_to_path(ags_claim['lineitem'], 'scores')

score = {
    'userId': student_launch['sub'],
    'scoreGiven': 9,
    'scoreMaximum': 10,
    'activityProgress': 'Completed',
    'gradingProgress': 'FullyGraded',
    'timestamp': datetime.utcnow().isoformat()
}

headers = {'Content-Type': 'application/vnd.ims.lis.v1.score+json'}
add_authorization(headers, token_info['access_token'])

r = requests.post(scores_url.encode(), headers=headers, data=json.dumps(score))

# let's check it was OK
r.raise_for_status()

md('The score was processed successfully be the back-end')
```

The score was processed successfully be the back-end

### Step 3: get the results

Let's not get the results to see our operation did actually succeed

In [16]:
```
results_url = append_to_path(ags_claim['lineitem'], 'results')

headers = {'Content-Type': 'application/vnd.ims.lis.v2.resultcontainer+json'}
add_authorization(headers, token_info['access_token'])

r = requests.get(results_url.encode(), headers=headers)

# let's check it was OK
r.raise_for_status()

md('#### Current results for item')
md('```json\n'+ json.dumps(r.json(), indent=4)+'```')
```

**Current results for item**


```
[
    {
        "resultMaximum": 34,
        "resultScore": 30.6,
        "userId": "LTIBCU_15"
    }
]
```


In [ ]:

In [ ]: