
2U
edX
Commerce Coordinator: Connecting Open edX with Commerce
Open edX Conference 2023
Lightning Talks: Developing & Operating
tinyurl.com/oec-2023-ecomm-lightning

Hello!

Phil Shiu
Software Engineer
Purchase Squad
@pshiu

Glenn Martin
Principal Software Engineer
Purchase Squad
@gmartin
@grmartin

tinyurl.com/oec-2023-ecomm-lightning

[Glenn]

- Hello everybody, good afternoon!
- (Glenn, Phil introduce themselves)
- This is our Ecommerce lightning talk.

Alright, what have we been up to?

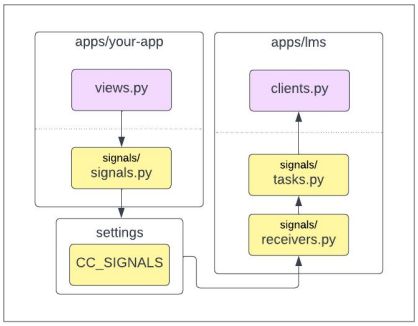
- **For background** and rationale for the move to Commerce Coordinator, please feel free to review our talk from Open edX 2022: Ecomm for edX.
- Ecommerce is in maintenance mode**, Stripe will be available** as a processor in the **Palm** release.
- And while Ecommerce (in its current form) is still **going away**... the community feedback on Coordinator's complexity, has encouraged us to think critically about how to simplify it!

tinyurl.com/oec-2023-ecomm-lightning

[Glenn]

- E-commerce has been on a **long journey** in the Open edX project
- As a community, we need to figure out **where to go next**.
- Currently, we are working on a solution called **Commerce Coordinator**.
  - It's meant to be a **request router and transformer**
  - See our **slides** at the conference last year!
- So what have we been up to?
  - We actually got **sidetracked** because there was an pressing need to migrate to another payment processor at our company, 2U
  - So we had to finish that up, and the advantage is now **Stripe is available** as a payment processor on master and we hope the community will be able to see it in Palm.
  - Please check out those release notes if you haven't already.
- But right now, we are back on the **Ecommerce Deprecation & Replacement Project**.
  - While Ecommerce and its current form is still going away…
  - We've received **good feedback** that we need to simplify the Coordinator and are thinking critically how to simplify it.
  - To be completely honest, we don't have an exact answer right now in terms of how the Coordinator is going to be simpler and how it could meet everyone's needs.
  - But we wanted to give a shot at showing briefly what we have so far, what a **typical flow** in the Coordinator might look like

An example: Fulfillment



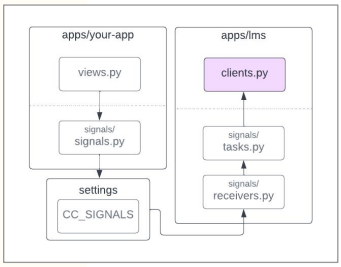
tinyurl.com/oec-2023-ecomm-lightning

[Glenn]

- So let's start with an example, fulfillment.
- The idea behind **fulfillment** is that after a user pays for a course, the operator needs to give the user access to the course they have paid for.
- We call that action "fulfillment". Other may call it "enrollment".

apps/lms/clients.py

```
class LMSAPIClient(BaseEdxOAuthClient):
    """
    API client for calls to the edX LMS service.
    """

    def enroll_user_in_course(self, enrollment_data):
        """
        Send a POST request to LMS Enrollment API endpoint
        Arguments:
            enrollment_data: dictionary to send to the API resource.
        Returns:
            dict: Dictionary representation of JSON returned from API.
        """
        return self.post(
            url=self.api_enrollment_base_url,
            json=enrollment_data,
            timeout=settings.FULFILLMENT_TIMEOUT
        )
```



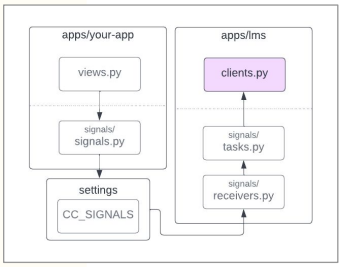
tinyurl.com/oec-2023-ecomm-lightning

[Glenn]

- So let's **work backwards**, starting with our goal.
- In our LMS we have API endpoints that allow a user or an external system to place a user into a course.
- You can see this code is reaching out to that endpoint and sending a JSON object called **enrollment_data**

enrollment_data JSON

```
enrollment_data = {
      'user': user.username,
      'mode': course_mode,
      'is_active': True,
      'course_details': {
          'course_id': course_id
      },
      'email_opt_in': email_opt_in,
      'enrollment_attributes': [
          {
              'namespace': 'order',
              'name': 'order_number',
              'value': order_number,
          },
          {
              'namespace': 'order',
              'name': 'order_placed',
              'value': date_placed,
          }
      ]
}
```



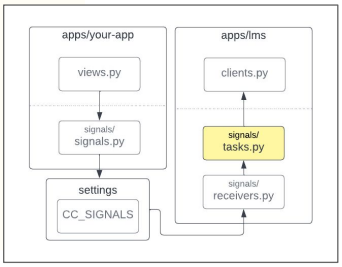
tinyurl.com/oec-2023-ecomm-lightning

[Glenn]

- Here's what enrollment_data looks like.
- You can see it's just a **dictionary** with information about how we want to do the fulfillment.

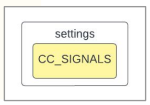
apps/lms/signals/tasks.py

```
@shared_task()
def fulfill_order_placed_send_enroll_in_course_task(course_id, course_mode,
date_placed, edx_lms_user_id, email_opt_in, order_number, provider_id):

    user = User.objects.get(lms_user_id=edx_lms_user_id)

    enrollment_data = {...}

    return LMSAPIClient().enroll_user_in_course(enrollment_data)
```



settings/base.py

```
CC_SIGNALS = {
    'commerce_coordinator.apps.your-app.signals.fulfill_order_placed_signal': [
        'commerce_coordinator.apps.lms.signals.fulfill_order_placed_send_enroll_in_course',
    ],
}
```



tinyurl.com/oec-2023-ecomm-lightning

7

[Glenn]

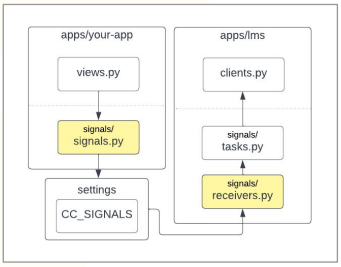
- We send enrollment_data from a **Celery task**.
- It's very **simple**, this is literally a copy and paste of what we have in our code right now
- This Celery task allows us to queue up the requests for fulfillment.
    - **Why**: We needed to add this because as a larger operator we have a lot of fulfillment requests and historically this fulfillment endpoint has been a very slow operation for us
    - **How we solve**: So we have had need of some sort of asynchronous queue to do this fulfillment so our e-commerce servers do not stall other requests while waiting for fulfillment
- And this **configuration** in **settings/base.py** configures Coordinator to **create a task to fulfill a course** every time our backend systems at 2U requests a fulfillment.
- I think this is the heart of the **versatility** of the Coordinator: as long as you fire a signal with the right parameters, **any app in the Coordinator can trigger fulfillment**.
- (There's a bit of **code we're not presenting**, but it's mainly a signal–receiver pair that is a pass through of parameters. Nothing fancy there, but you can see the missing slide if you check out our presentation online.)

apps/your-app/signals/signals.py

```
from commerce_coordinator.apps.core.signal_helpers import CoordinatorSignal

fulfill_order_placed_signal = CoordinatorSignal()
```

apps/lms/signals/receivers.py

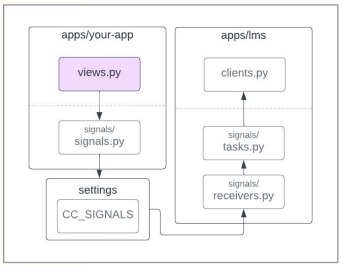
```
@log_receiver(logger)
def fulfill_order_placed_send_enroll_in_course(**kwargs):
    """
    Fulfill the order with a Celery task to LMS to enroll a user in a single
course.
    """
    fulfill_order_placed_send_enroll_in_course_task.delay(
        course_id=kwargs['course_id'],
        course_mode=kwargs['course_mode'],
        date_placed=kwargs['date_placed'],
        edx_lms_user_id=kwargs['edx_lms_user_id'],
        email_opt_in=kwargs['email_opt_in'],
        order_number=kwargs['order_number'],
        provider_id=kwargs['provider_id'],
    )
```



(There's a bit of code we're not presenting, but it's mainly a signal–receiver pair that is a pass through of parameters. Nothing very fancy here.)

apps/your-app/views.py

```python
from commerce_coordinator.apps.core.signal_helpers import format_signal_results

from .serializers import OrderFulfillViewInputSerializer
from .signals import fulfill_order_placed_signal

class OrderFulfillView(APIView):
    def post(self, request):
        params = {
            'course_id': request.data.get('course_id'),
            'course_mode': request.data.get('course_mode'),
            'date_placed': request.data.get('order_placed'),
            'edx_lms_user_id': request.data.get('edx_lms_user_id'),
            'email_opt_in': request.data.get('email_opt_in'),
            'order_number': request.data.get('order_number'),
            'provider_id': request.data.get('provider'),
        }

        serializer = OrderFulfillViewInputSerializer(data=params)

        if serializer.is_valid(raise_exception=True):
            results = fulfill_order_placed_signal.send_robust(
                sender=self.__class__,
                **serializer.validated_data
            )
            return Response(format_signal_results(results))
```



9

[MIDDLE: Glenn & Phil]

[Glenn]

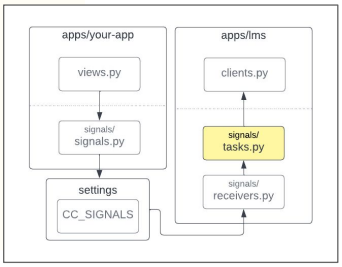
- Finally, we have a **view**. This is where our backend systems at 2U reach out to request fulfillment of the course.
- All we do is:
  - Get all those **parameters** that we plug into the enrollment_data dictionary
  - **Validate** those are sane values
  - And send the **signal**
- And we don't mean to be mysterious about our backend systems: it's basically a software like what our ecommerce repo uses, Django Oscar, but instead it's in **Ruby** and it's called **Spree**. Let's not worry about that here; just imagine **your backend system** is able to reach out to this endpoint here.

[Phil]

- So one more time, let's review what happens, this time **working forwards** instead of backwards.
- Our financial backend systems hit this OrderFulfillView endpoint you're seeing on screen.

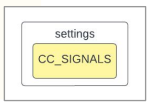
apps/lms/signals/tasks.py

```
@shared_task()
def fulfill_order_placed_send_enroll_in_course_task(course_id, course_mode,
date_placed, edx_lms_user_id, email_opt_in, order_number, provider_id):

    user = User.objects.get(lms_user_id=edx_lms_user_id)

    enrollment_data = {...}

    return LMSAPIClient().enroll_user_in_course(enrollment_data)
```



settings/base.py

```
CC_SIGNALS = {
    'commerce_coordinator.apps.your-app.signals.fulfill_order_placed_signal': [
        'commerce_coordinator.apps.lms.signals.fulfill_order_placed_send_enroll_in_course',
    ],
}
```



10

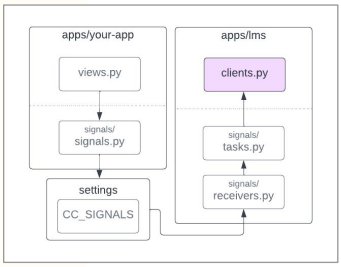
[Phil]

In our **settings**, we've configured that signal to be **sent to a signal receiver** in the part of Coordinator's **code that talks to LMS**.

This **fires off a Celery task**, which puts the **parameters** we've sent into a **queue**.

apps/lms/clients.py

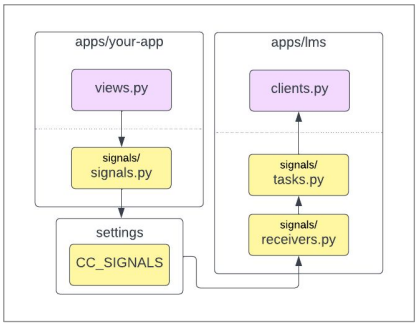
```
class LMSAPIClient(BaseEdxOAuthClient):
    """
    API client for calls to the edX LMS service.
    """

    def enroll_user_in_course(self, enrollment_data):
        """
        Send a POST request to LMS Enrollment API endpoint
        Arguments:
            enrollment_data: dictionary to send to the API resource.
        Returns:
            dict: Dictionary representation of JSON returned from API.
        """
        return self.post(
            url=self.api_enrollment_base_url,
            json=enrollment_data,
            timeout=settings.FULFILLMENT_TIMEOUT
        )
```



[Phil]

And when the task in the queue's turn arrives, the **task reaches out to LMS** and sends that **enrollment_data** dictionary we've built using the **parameters** our **backend system** originally sent.

An example: Fulfillment



tinyurl.com/oec-2023-ecomm-lightning

[Phil]

And so **this is the general pattern** of what the Coordinator is doing.

Why might this be a good idea?

- The architecture lets different companies drop in plugins or IDAs and provides a **common framework** for working together on #ecommerce.
- We are trying to **migrate** the parts of Ecommerce we use to Coordinator, bit by bit, and will probably be doing this through mid-2024.
    - We are hoping our work will be by nature generalizable enough to be useful to others in the community.
- A lot of us are writing **custom code** for ecommerce. Coordinator is a place to run that code within the Open edX ecosystem.

tinyurl.com/oec-2023-ecomm-lightning

[Phil]

Ok. Why might this be a good idea? Three reasons.

1. We have a place to **work together** on e-commerce using a single framework and language
2. We, 2U, are migrating the parts of Ecommerce we use to the Coordinator, and maybe the community can use that work in some form, whether it's running Coordinator in production, or using Coordinator as a reference implementation for their own code.
    a. We are hoping that our work will be able to be **generalized to others** in the community.
3. A lot of us have the same problem: where do we **run the custom code** we need for selling our courses? Coordinator provides a place for that code to run.

But our solution is overly complicated...

- We've received clear feedback from the community that there are a lot of operators that a solution like Coordinator is more overhead than it's worth.
- We are trying to think of ways to simplify the Coordinator—simplification helps both us and the community! Ideas welcome.

tinyurl.com/oec-2023-ecomm-lightning

[Phil]

But we know our solution is overly complicated.

We met back in July and got a **clear message** that the Coordinator is **too much overhead** for most operators.

We're trying to continue to think of ways to simplify this architecture.

Internally, we need the Coordinator at 2U because:

- **Can't put it in LMS**: Open Source wouldn't accept code that only applies to our systems in LMS
- **Don't want to reinvent in Ruby**: We could call the LMS directly from Ruby, but this requires reinventing a lot of Python libraries we've already made in Ruby.

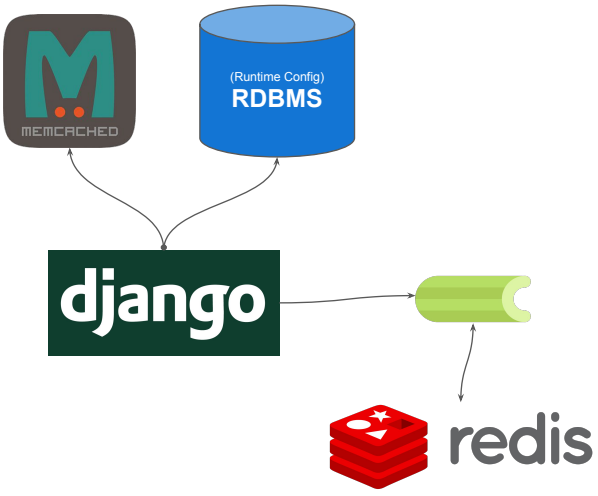
So Coordinator lets us put this logic in Python and transform the incoming requests into the right shape so that we can fire the request into the open source world.

It sort of becomes the **transition zone** between **closed source** that is very particular to us and only us as an operator, and **open source**, which is generalizable to everybody.

But not everyone will need what we are going to need.

A small illustration: Infrastructure

- Coordinator requires:
  - A database
    For site configuration, as we do not store business objection
  - A Celery broker like Redis
  - Memcached



All of these items are used by other Open edX services (incl. old Ecommerce), but not all operators need or want a setup like this.

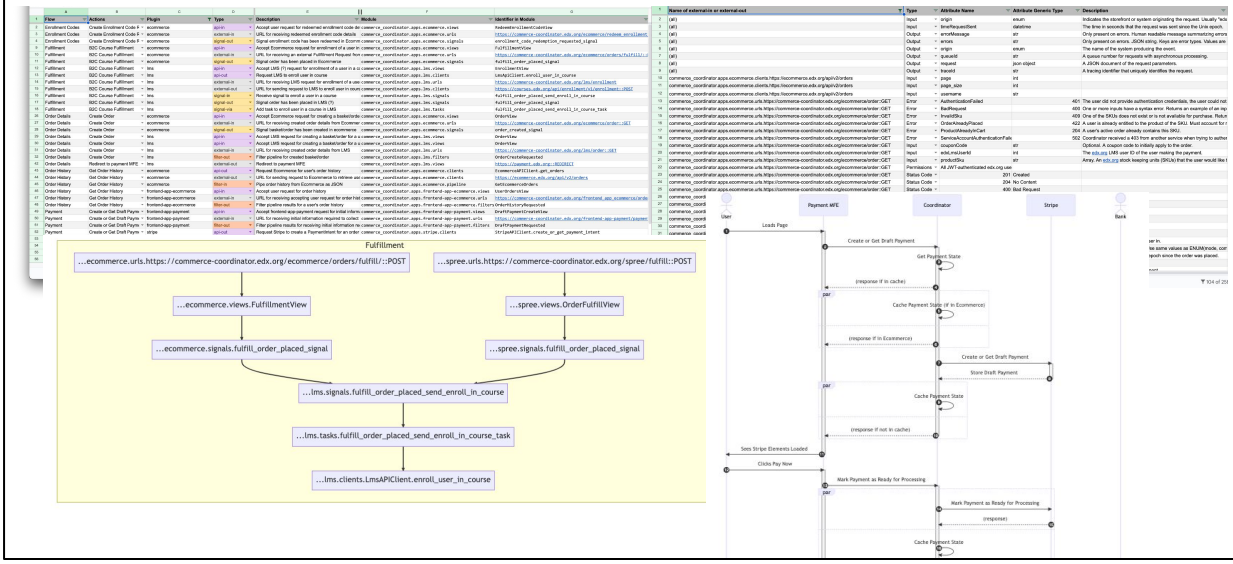
15

[Phil]

(Will probably not have time to go over this slide, but:)

There are **a lot of reasons why** it might **not work** for everybody, but one of them is infrastructure. Coordinator needs a **Celery backend** like Redis. Many folks don't need or want a Celery backend or Redis.

What we're up to now...



[Phil]

So what are we up to?

Lots of **documentation**.

[Click]

We're doing a lot of work to try to **understand** what exactly Ecommerce does and **re-organize** the parts we use into the Coordinator so that it's more organized and systematic.

Because we're **drowning in documentation**, we also hope to make **tooling** to **auto-gen**erate **runtime** diagrams like the one you see **in yellow** so it's really easy to for folks to see, oh, ok, my setup of the Coordinator allows either the legacy Ecommerce service or another service called Spree to call the fulfillment endpoint.

And we've been **sharing** all this information with tCRIL so that it can help inform their projects as we together try to advance the state of Open edX ecommerce with the community.

Divining: The future of #ecommerce

- For **smaller** operators, **tCRIL** is funding a contribution to help advance the Open edX platform. This project aims to do some discovery and build a reference implementation for integration the Open edX platform with **WooCommerce** as the commerce system.
- For **larger** operators or those requiring more custom implementations, Commerce Coordinator is an option. We are still looking for **partners** to help us make this more useful for these more complex operational needs.
- We have **common pain points**. Examples: adding payment processors, support for multiple currencies. How can we tackle these subjects together?

17

[Phil]

Alright. So where do we go from here? We don't know the answer to that. But if I were to perform a little bit of divination, here's what I can see from my vantage point. And others might have additional thoughts as well.

For **smaller operators**—tCRIL is funding a really cool discovery project that is looking into wiring WooCommerce with Open edX and make that really easy to setup. So that's really exciting work. It's a kind of blended development model so if you are a smaller operator and you are interested in WooCommerce, definitely take a look at that project.

We're trying our best to **help** with that project—like we mentioned earlier, we are sharing our **R&D** with tCRIL and actually already met a team looking into that work and were **talking shop** just last night.

And if you **operate** any sort of system running **Django Oscar ecommerce** with Open edX, or if you're a **larger operator** interested in selling courses using Open edX software, please talk to us, because the **message** we've gotten so far is that there are **not** a lot of folks out there interested in the scale we're working with right now.

And this leads to a very important question: how do we work together on this?

Vision casting needed:

How are we as a community together going to collaborate on #ecommerce in open source?

What would a world in which we collaborated even more closely together on Open edX #ecommerce look like?

[Phil]

Vision casting is needed.

(read the slide)

We don't know the answer on this. This is an **open question**, but I would definitely challenge everyone in the room to **think about these things**.

Ecommerce is **naturally** a very **difficult** thing to collaborate on because it **touches the revenue model** of all of our organizations, and revenue is often where the rubber hits the road. But I think it's possible. I have a glimmer of hope that there are awesome things in the e-commerce world out there. And we need to figure out how we as a community want to **work together** to do that.

For more information:

- NEW: **Join us at the Ecommerce Next Step Update on April 26**
- NEW: tCRIL Funded Contribution - WooCommerce Discovery
- NEW: Ecommerce: Changes to Stripe payment processor
- NEW: Previous Ecommerce Next Steps Update - Info Session (22 Dec 2022)
- NEW: Ecommerce Dep & Rep Project Confluence Pages
- NEW: Racoon Gang E-commerce Pain Points
- Ecommerce deprecation announcement
- Commerce Coordinator ADRs

19

[Phil]

Alright, that's it! Lots of links here. Wanted to quickly highlight two things.

1. None of our code is a secret right now. Check out the **2u/project-theseus** branch in GitHub commerce-coordinator, links on the Confluence page to that repo.

2. And please **join us** at the Ecommerce Next Steps update on **April 26**. We'd love to hear your thoughts.

Thank you.

#ecommerce

Phil Shiu
Software Engineer
Purchase Squad
@pshiu

Glenn Martin
Principal Software Engineer
Purchase Squad
@gmartin
@grmartin

tinyurl.com/oec-2023-ecomm-lightning

[Phil]

If you have any follow up questions or thoughts, there's the April 26 **meeting**, or please come **say hi** or reach out on the **#ecommerce** channel in the Open edX Slack.

Thank you so much!