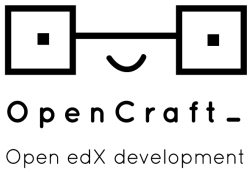
(Revisited) Technical Discovery & Scoping

Instructor Dashboard MFE conversion



Prepared for edX by OpenCraft

March 2022

Context

This discovery investigates and estimates converting the Instructor Dashboard to a Micro Frontend. The goal of the project is to match parity/features on the current dashboard in a new micro front end. Additional details (mostly from the *learner dashboard* discovery, which apply here as well):

- A new MFE using the updated front-end application template, which includes a shift to using react and paragon components, that replaces the courses.edx.org/courses/{course_id}/instructor page. (OEP-11 related)
- The toggles/switches necessary to switch over to using this MFE instead of the current Django view (OEP-17 related)
- The tabs should be done via some form of plugin boundary (extension points guide)
- As part of these, we'd like to set aside some time to improve usage documentation for developers (OEP-19 related)
- Additionally, some APIs may need to be written to support the features this page uses, the details here edX can help provide but hopefully, the code can help convey much of what is necessary here. (No existing/active OEP, but can collaborate on how to make incremental progress here as is necessary to complete this project.)
- Once the MFE exists we will want to mark the current dashboard as deprecated and do the appropriate notifications (OEP-21 related)

The current state of the Instructor Dashboard

Instructor Dashboard is a separate Django app with the main view instructor_dashboard_2. As it's accessible from the course view via the course navigation tabs, we propose to extract it into a separate MFE (in a separate repository). The new MFE will be embedded in an iframe in frontend-app-learning, like the Discussions MFE, which are embedded in the LMS. You can see how it's working in this sandbox (username: "staff@example.com", password: "edx").

By creating the smaller endpoints, fragmenting the frontend parts to separate components, and making them own their loading state (as done in frontend-app-learning) we could separate different concerns here. As the sections of the current dashboard depend on various parts of the platform and external plugins, the new dashboard must be plugin-based - it should not determine whether the specific sections should be displayed or not; rather, each plugin will determine whether it is displayed or not.

Required functionalities related to the Instructor Dashboard

"Instructor" course tab

The tab should not be shown when the user doesn't have *VIEW_DASHBOARD* permissions. This is already handled by *frontend-app-learning*, as the tab is not sent with */api/courseware/course/{course_id}* when the user does not have these permissions.

Permissions

There are multiple access levels related to the instructor dashboard:
- *'admin',*
- *'instructor',*
- *'finance_admin',*
- *'sales_admin',*
- *'staff',*
- *'forum_admin',*
- *'data_researcher'.*

Some dashboard sections, or their parts, are gated by these permissions. While updating the current Instructor Dashboard APIs (mentioned below, under “APIs”), we can reuse that approach from Discussions - the API is returning a list of `editable_fields`. We could also add a field, called e.g. `allowed_actions`. It will be used by the MFE to determine which parts of the UI should be displayed for the particular user.

The "View Course in Studio" button

The current version of the dashboard displays a button leading to the course outline page in Studio. The Learning MFE adds the Instructor Toolbar, which already contains this button. Therefore, we can remove this feature from the dashboard.

Plugins

We can introduce a new type of tab, which will work similarly to the Course Tabs extension point. This tab type will automatically detect new entries and display them in the instructor tabs list. Each tab will be responsible for performing permissions checks and deciding if it should be shown for the current user. It will also return a list of actions that are available for a user. Each of these tabs will also have its API and a frontend component that is rendered for this tab. The frontend component can call all APIs it needs to do its work.
The first-party (built-in) instructor tabs will be built into the instructor dashboard MFE and for third-party ones (registered via plugins), we will render their UI in an iframe. This aligns with the current draft of the frontend pluggability OEP, which recommends iframes.

APIs

47 of the existing API endpoints used by sections do not support OAuth2 and need CSRF tokens. We can create new REST API endpoints and mark the current ones for deprecation. We will reuse the code from existing views and split the new APIs logically between Django apps - i.e. `enrollments`, `grades`, etc. They will benefit from some adjustments, like the native DRF permissions and proper documentation with `drf-yasg`. This way, we will have no dependencies with the old views (like we would have after proxying the APIs) once it comes to removing the legacy Instructor Dashboard. We will mark the old views as deprecated, so they can be removed with the legacy version of the Instructor Dashboard.

Sections

The Instructor Dashboard is divided into the following sections. For making the development process iterative, we will be creating the sections one by one. The new dashboard will have a button linking to “existing experience” for having full backward compatibility and consistency with the *frontend-app-learning* MFE.

The following points list the sections that exist in the current version of the Instructor Dashboard.

Course Info section

This section displays a breakdown of how many learners are enrolled for each enrollment track.
We could use */enrollment/v1/enrollments/?course=id=<course_id>* for retrieving the enrollments and counting them on the frontend, but this would be a heavy operation for bigger courses, so we will add a new endpoint in the Enrollments API for retrieving the aggregated data. We could introduce a new `?mode=summary` query param instead, but the response will be different, so we’re proposing the former approach.

Membership section

This section displays functionalities for batch (un)enrollments (via plaintext or CSV), batch beta tester addition/removal, and course team management.

We will move the existing APIs related to:

1. Enrollments - to `openedx/core/djangoapps/enrollments/views.py` or `lms/djangoapps/bulk_enroll/views.py`.
2. Discussions - to `lms/djangoapps/discussion/rest_api/views.py`.
   These could also be moved to the Discussions MFE - either as a part of this project, or another one.

Cohorts section

This section allows enabling/disabling cohorts for a course and cohort management. It consists of the following templates:

- `cohort-editor.underscore`
- `cohort-form.underscore`
- `cohort-group-header.underscore`
- `cohort-selector.underscore`
- `cohort-state.underscore`
- `cohort_management.html`
- `cohorts.underscore`

We can move the existing APIs to `openedx/core/djangoapps/course_groups/views.py`.

Extensions section

This section adds an option to manage extensions on specific subsections to individual students.
It is displayed when edx-when is *enabled* for the course. We can either leave this inside *edx-platform*, move it to a separate plugin or move it to edx-when because the newly developed plugin API will allow us to inject it into the new dashboard.

We can move the APIs to the edx-when repository.
Additionally, we can move helper functions from edx-platform directly to edx-when. They are not using any platform-specific dependencies, except the `Schedule`, which is imported directly by edx-when anyway.

Discussions section

This section allows dividing discussions between enrollment tracks or cohorts.
Checks whether the course is cohorted and the number of the enrollment track division schemes for the course.

As the Discussions can now be configured entirely from the Course Authoring MFE, we can remove this section.

Student Admin section

This section allows viewing the gradebook for enrolled learners, checking the user's enrollment status, checking the learner's progress (directs to the *courses/{course_id}/progress/{userId}* page) and adjusting grades.
The current version displays the gradebook only for *small* courses. We could mark the old gradebook view as deprecated and only use the frontend-app-gradebook MFE, embedded in an iframe.

We can move the APIs to `lms/djangoapps/grades/rest_api/v1/views.py`.

Data Download section

This section allows downloading data related to the course (e.g. student anonymized IDs, profile information, grade reports, certificates, etc.).
Displays additional “Click to generate a CSV file of all proctored exam results in this course.” button for special exams.

With the plans of improving data reporting APIs, we could keep the existing APIs in the current place. However, we could consider moving “get_proctored_exam_results” to the Special Exams section.

Analytics section

This section displays the URL to the analytics page (if enabled). This button is already present in the Instructor Toolbar, so we can remove this section to reduce redundancy.

Email section

This section allows sending bulk emails to enrolled learners or cohorts.
It currently uses an artificial HtmlBlock for providing an editor. We could embed the block in React or rewrite this to use a third-party editor (e.g. TinyMCE).
This also retrieves a list of cohorts for a course. There is */api/cohorts/v1/courses/{course_id}/cohorts* endpoint, but at the moment of writing it's not working: “*'CohortHandler' should either include a `queryset` attribute or override the `get_queryset()` method.*”. We should fix it and reuse it.

We can move other API endpoints to `lms/djangoapps/bulk_email/views.py`

Special Exams section

This section allows managing student special exam attempts when special exams are enabled for a course.

We can move APIs to edx-proctoring.

Certificates section

This section allows managing the certificates and viewing the certificate generation history if a course has certificates enabled. We should introduce new APIs to retrieve *CertificateGenerationHistory* and count the number of certificates with each status.

We can use `lms/djangoapps/certificates/apis/v0/views.py` for the APIs. We could also create a v1 API for this Django app.

Open Responses section

Displayed when there are *openassessment* blocks in the course. We could embed the *ora_blocks_listing_view* in React for now and look into rewriting it to React later.

Implementation plan

Convert Instructor Dashboard views to RESTful APIs

This will allow calling them from the MFE. They will be properly documented, as described in the OEP-19.

Prepare the Instructor Dashboard MFE

We will start converting existing sections to React. They will be built into the new MFE.

Implement plugin support

This will include an option to dynamically load tabs from plugins. It will allow third-party applications to add new instructor tabs, which will contain MFEs that will be displayed in iframes.

Deprecate and remove the existing Instructor Dashboard

The deprecation and removal process is defined in OEP-21. As the timeline for the deprecation of the current Instructor Dashboard is not known, and the amount of the removed codebase will depend on the previous steps, it would be reasonable to do a small discovery after implementing this MFE to determine the actual effort needed for deprecating and removing the current Instructor Dashboard.