Zoom rooms evaluation and webcast projects.

By Samuel Guillemet

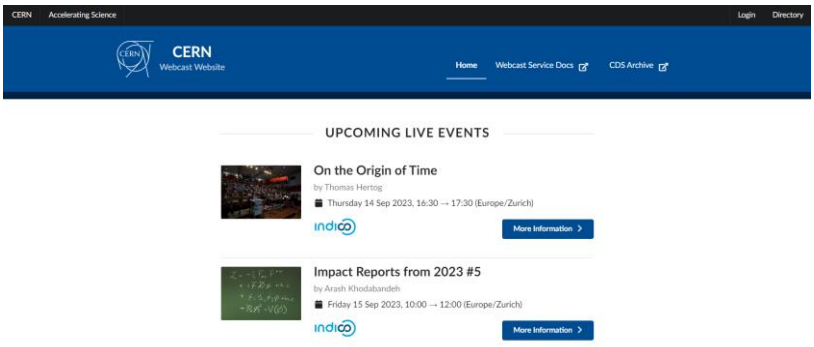
Supervised by René Fernandez Sanchez and Ruben Domingo Gaspar Aparicio

Introduction

The project titled "An Assessment of Zoom Rooms as Codec Replacements" endeavors to evaluate the efficacy of Zoom video conferencing technology within the meeting rooms of CERN. Zoom, a widely recognized online communication platform, empowers individuals with the capabilities of video conferencing, webinars, and virtual meetings. Its significance has been particularly highlighted during the pandemic as a crucial tool for remote communication. This project aims to ascertain its compatibility with the unique requirements of CERN's scientists and researchers.

The second project involves establishing a fallback solution for the conference website's broadcasting functionality (Webcast Website). Its primary objective is to develop an alternative website that can effectively replicate the main website's functions while adhering to a minimalist approach, utilizing solely static website generation.

1- The full device for the Zoom Room



2- The Webcast Website

Zoom Room

Objective

The primary objective of this project is to establish seamless integration with the calendar system of the Zoom Room device, thereby enabling the creation of events within the Zoom environment. By achieving this integration, we aim to enhance the functionality of the Logitech device positioned beneath the television.

In practical terms, this means that when users schedule events on their calendar that include Zoom meetings, the Logitech device will display these events prominently. This display will include essential details such as the event name, time, and a clickable Zoom meeting URL. This approach simplifies the process for users, as they can effortlessly initiate their scheduled Zoom meetings with just a few clicks on the Logitech device.

In essence, this project's objective is to create a more user-friendly and efficient experience for individuals utilizing the Zoom Room setup, ensuring that they can easily access and manage their meetings through the Logitech device without the need for manual input or cumbersome steps.

Different research

During the initial phases of this project, a substantial portion of our efforts was dedicated to conducting an in-depth exploration of the diverse Application Programming Interfaces (APIs) offered by Zoom and Microsoft Graph. This investigation aimed to establish a comprehensive understanding of the capabilities and functionalities available within the Zoom and Microsoft ecosystems.

Within the Zoom API ecosystem, we identified two distinct APIs that held the potential to be of value: the Base Zoom API and the Zoom Room API. Our research yielded the following key insights:

1. **Zoom Rooms API Limitations**: It was discovered that the Zoom Rooms API lacked the capability to facilitate the creation of events directly within a calendar. This limitation meant that the Zoom Rooms API couldn't directly manage calendar-related functions.
2. **Zoom API Meeting Endpoints**: We also found that, although the Zoom API offered various meeting endpoints, these endpoints were not designed to create events on a calendar. In essence, the Zoom API focused on facilitating meetings but did not encompass calendar event creation.
3. **Email Notifications**: One noteworthy observation was that meetings created through the Zoom API did not automatically trigger email notifications. This insight highlighted the need for additional mechanisms to interact with the calendars, indeed an Email Event notification creates an event on a calendar.
4. **Zoom Workspaces**: Our research also led us to the existence of Zoom Workspaces as another product in the Zoom ecosystem. Importantly, it was determined that Zoom Workspaces did not inherently display pending meetings within the Zoom Room environment, potentially necessitating custom solutions for meeting visibility.
5. **Unique Microsoft Account Integration**: Further investigation revealed that each individual Zoom Room was intricately linked to a unique Microsoft account. This linkage suggested that any interactions or integrations with Microsoft services could lead to an ability to create events in an easy way.

In summary, the project's early phases involved a comprehensive exploration of Zoom and Microsoft Graph APIs to discern their capabilities and limitations. These findings served as a foundational basis for the subsequent development and implementation stages, guiding our approach to create efficient solutions and integrations within this context.

The selected solution

The chosen approach for interacting with the Outlook calendar of the Zoom Room involves direct integration with the Microsoft Graph API. To streamline this process and enhance usability, a dedicated proxy account has been established. This proxy account possesses access to all the Zoom Room's calendars as shared calendars, simplifying the authentication and data retrieval process. Here's a breakdown of how this integration works:

1. **Proxy Account Setup**: A proxy account has been configured specifically for this purpose. This account is equipped with permissions to access all the calendars associated with the Zoom Rooms, functioning as a central point of authentication and data management.
2. **Authentication**: To initiate the interaction programmatically, the chosen authentication method is the device flow. In this approach, during the initial setup, user authentication is required via a human interaction process with the proxy account. This initial authentication grants the necessary permissions for subsequent interactions.
3. **Python Package**: To facilitate interaction with these calendars programmatically, a Python package has been developed. This package includes user-friendly interfaces and tools that simplify the process of adding, modifying, or deleting events on the Zoom Room calendars. This python package could be found here: https://github.com/cern-vc/ms-python-client

4. **Token Management**: Following the initial authentication, a set of refresh tokens is generated. These tokens are crucial for maintaining authentication without requiring repeated human interactions. They ensure that the system remains authenticated with the proxy account, allowing seamless and ongoing interaction with the Zoom Room calendars through the Microsoft Graph API.

Additional work on this project

One of the significant aspects of the Zoom Room integration project involved seamlessly integrating Zoom Rooms into the CERN ecosystem, particularly within the context of the RAVEM website. The RAVEM website plays a crucial role in managing the IP2CC On Air device, responsible for controlling the meeting rooms' on/off status.

To achieve this integration, several key steps and updates were made, primarily centered around enhancing the existing Python package's capabilities for interacting with the Zoom API. Here's a breakdown of the specific enhancements made:

Enhancements to Python Package for Zoom API Interaction, which could be found here: https://github.com/cern-vc/zoom-python-client

1. **Zoom Rooms**:
   - *Get All Zoom Rooms*: This endpoint was added to retrieve a comprehensive list of all Zoom Rooms, facilitating better management and monitoring.
   - *Get Zoom Room Details*: Another endpoint was implemented to retrieve detailed information about specific Zoom Rooms, offering insights into their configurations and settings.
   - *Get Zoom Room Sensor Data*: This endpoint was designed to fetch sensor data from Zoom Rooms, which could be valuable for tracking room usage and occupancy.
2. **Calendars**:
   - *Get Calendar Services List*: This endpoint provides a list of available calendar services, which can be useful for identifying the integration points.
   - *Get Calendar Resources List*: It became possible to fetch a list of calendar resources, allowing for efficient resource allocation and management.
   - *Get Calendar Resources by Service ID List*: This endpoint enhances resource retrieval by allowing filtering based on service ID.
   - *Get Calendar Resource Details by ID*: This endpoint offers detailed information about a specific calendar resource, aiding in precise resource utilization.

Integration with RAVEM Website:

These expanded API capabilities were utilized to enable a Celery task that regularly checks the status of each Zoom Room. The task updates the IP2CC On Air device, ensuring that it reflects the real-time status of the Zoom Room (whether it's in use or not). Additionally, the task leverages the calendar endpoints to extract essential details about ongoing meetings. Specifically:

- The task retrieves the room's email associated with the resource in the Outlook calendar.
- The Microsoft Python client is employed to gather information about the current event in the calendar, including its title.

In summary, these enhancements and integrations ensure a seamless connection between Zoom Rooms and the CERN ecosystem, particularly the RAVEM website. The enriched Python package with expanded API endpoints empowers efficient management of Zoom Rooms and enables real-time

updates of the IP2CC On Air device status, along with detailed information about ongoing meetings for improved room utilization and scheduling.

Webcast Website

Objective

The primary aim of this project was to develop a simplified version of the Webcast website, designed as a fully static website. The impetus for this project stemmed from a critical issue encountered with the existing webcast website. The problem lay in the fact that authentication could not be disabled in the event of any disruptions or problems with the Single Sign-On (SSO) system at CERN.

Considering this challenge, my overarching objective was to construct an alternative iteration of the website. This new version was meticulously crafted to precisely replicate the URL structure of the primary website. However, a pivotal enhancement was the implementation of a more flexible and versatile authentication mechanism that could be easily toggled on or off as needed.

The mechanism to switch between the two versions will be done using Nginx as a reverse proxy but this part will be in the hands of my supervisor.

Another thing to do was to deprecate the use of create-react-app as frontend stack and instead switch to Vite powered technology.

Different research

To address the authentication challenge in a WebEOS-hosted environment, we utilized an `.htaccess` file for authentication, which offers the flexibility to easily disable authentication when needed. The specific configuration included the use of OpenID Connect for authentication and access control based on claims associated with CERN roles. This configuration ensured that only users with the specified role identifier could access the webcast content.

```
AuthType openid-connect
Require claim cern_roles:<role-identifier>
```

Now, let's delve into the configuration of the webcast content. To define which webcast to display, we employed a configuration file structured as follows:

```
{
  "streams": "<camera_slides | camera | slides>",
  "cameraUrl": "<camera_url>",
  "slidesUrl": "<slides_url>",
  "title": "<event_title>"
}
```

At the outset of the project, we explored various potential solutions. The two primary options that underwent detailed examination were centered around creating a Single Page Application (SPA) using React, and this SPA would be powered by Vite.

1. **Two websites**: For the first approach, the idea was to separate the website from the server responsible for serving the webcast content. The website would attempt to load the configuration

and utilize the `.htaccess` file to display the webcast from a different server when users accessed a specific URL. However, this approach posed several challenges, particularly in terms of Cross-Origin Resource Sharing (CORS) issues arising from the redirection of the Single Sign-On (SSO) system. Consequently, we opted not to proceed with this solution due to its inherent complexities and potential security concerns.

2. **One Website**: The second option involved consolidating all configuration files and the `.htaccess` directives within a single URL. To achieve this, we employed an internal redirection mechanism with a complex file structure. Unfortunately, this approach was ultimately discarded because WebEOS's disabling of `.htaccess` support resulted in the internal redirection becoming non-functional, rendering the solution ineffective.

The selected solution

Following the setback encountered with the Single Page Application (SPA), we made the strategic decision to transition to a fully static website generated using Astro technology, which is powered by Vite. This shift in approach allowed for a more streamlined and reliable webcast platform. The website's build and deployment processes were automated using Continuous Integration and Continuous Deployment (CI/CD) pipelines within GitLab.

However, a significant challenge remained – integrating the webcast events from the existing webcast website into the new static platform. To overcome this hurdle, we harnessed the power of the GitLab Commit API. The process involved creating commits that contained the necessary configuration files. Optionally, an `.htaccess` file was included if authentication was required for a particular event.

Here's a simplified overview of the process:

1. **GitLab Commit**: We initiated a GitLab commit operation, during which we included the configuration file and, if necessary, the `.htaccess` file, tailored to the specific event's requirements.
2. **CI/CD Pipeline**: Subsequently, the CI/CD pipeline was triggered because of the commit. This pipeline handled the building and deployment of the new version of the static website.

Additional work on this project

The integration process into the core webcast website involved the implementation of a Celery task. This task was responsible for generating the necessary files using Jinja templating technology and initiating GitLab commits via the API. This approach proved to be a straightforward and efficient way to seamlessly integrate additional functionality and content.

Additionally, another noteworthy aspect of this project was the transition from Create React App (CRA) to Vite, a change that primarily pertained to the underlying module bundler used for the project. However, this transition introduced a few complexities, particularly about certain plugins utilized by the player, Paella player.

One significant challenge was that some of the plugins used by the player were initially compiled by Webpack (CRA's bundler) at runtime. Unfortunately, SWC, the compiler utilized by Vite, lacked support for this functionality. Consequently, I took the initiative to develop an NPM package tailored to this specific requirement. This package leveraged Webpack to bundle the necessary plugins for the player, enabling them to be seamlessly incorporated into the Vite project. This package can be found here: https://github.com/cern-vc/cern-paella-plugins .