Proceso de Scraping

Portal Mobilise-Me — Consulta de Competencias DC3

Crónica del flujo de scraping descubierto

Cliente: Newmont Peñasquito

1. APIs expuestas

Para que el bot de WhatsApp (o cualquier otro consumidor) pueda obtener las competencias DC3 sin tener que conocer el detalle del scraping, se exponen tres endpoints REST. El scraping ocurre del lado del servidor de manera asíncrona; el cliente solo interactúa con estas APIs.

1.1 Autenticación

Todas las peticiones requieren un header de autorización con un token estático que se entrega aparte por canal privado:

TOKEN 43eff45fff1422d21d371e3d72d76ae56f435a2c

```
Authorization: Token <token>
```

Las peticiones sin este header o con un token inválido son rechazadas con HTTP 401.

1.2 Endpoint 1 — Crear solicitud de búsqueda

```
POST https://penasquito.wotbi.com/api/v1/dc3/competencies/passports/search/
Content-Type: application/json
Authorization: Token <token>
```

Body:

```
{
    "passport": "E000055821"
}
```

Comportamiento. La API valida el passport, encola la tarea de scraping y devuelve inmediatamente un identificador de ticket. El proceso de scraping corre en segundo plano y no bloquea la respuesta.

Respuesta cuando la tarea queda encolada (comportamiento estándar):

```
{
    "status": "pending",
    "ticket_id": "2cf364bebf16",
    "posicion": 1
}
```

El cliente debe guardar el ticket_id y consultarlo en el endpoint de resultado (1.3) hasta que el scraping termine.

Respuesta cuando ya hay datos en caché. Si en la última hora ya se consultó ese mismo passport, la API devuelve directamente el resultado sin volver a encolar nada. Esto evita re-scrapear información que no cambia con frecuencia y libera carga del portal externo:

```
{
    "status": "completed",
    "result": {
        "passport": "E000055821",
        "name": "ITZEL CARDENAS REYES",
        "company": "INGENIERIA NET COM SOCIEDAD ANONIMA",
        "competencies": [
            {
                "activity": "FRM - CONFINED SPACES",
                "request_date": "30/07/2025",
```

```
                "expiration_date": null,
                "status": "Received"
            }
        ],
        "error": null
    }
}
```

Recomendación de uso: el cliente siempre llama primero a este endpoint. Si recibe `status: completed` usa el resultado al instante; si recibe `status: pending` pasa a hacer polling con el ticket_id.

1.3 Endpoint 2 — Consultar estado / resultado del ticket

```
GET
https://penasquito.wotbi.com/api/v1/dc3/competencies/passports/result/<ticket_id>/
Authorization: Token <token>
```

Comportamiento. Devuelve el estado actual del ticket. Mientras el worker no haya terminado, el campo `status` puede ser `pending` o `processing`. Cuando el scraping termina, `status` pasa a `completed` y se incluye el campo `result` con todos los datos.

Respuesta de un ticket completado:

```
{
    "status": "completed",
    "result": {
        "passport": "E000055821",
        "name": "ITZEL CARDENAS REYES",
        "company": "INGENIERIA NET COM SOCIEDAD ANONIMA",
        "competencies": [
            {
                "activity": "FRM - CONFINED SPACES",
                "request_date": "30/07/2025",
                "expiration_date": null,
                "status": "Received"
            }
        ],
        "error": null
    }
}
```

Estructura del campo result:

Campo	Descripción
`passport`	Passport consultado.
`name`	Nombre del empleado, extraído del HTML del portal.
`company`	Empresa a la que pertenece el empleado.
`competencies`	Lista de competencias DC3. Cada elemento incluye activity, request_date, expiration_date (puede ser null si no expira) y status.

Campo	Descripción
`error`	null en caso exitoso. Si hubo un fallo (sesión expirada, passport no encontrado, etc.) trae el mensaje descriptivo y competencies viene vacío.

1.4 Endpoint 3 — Estado de la cola

```
GET https://penasquito.wotbi.com/api/v1/dc3/competencies/passports/status/
Authorization: Token <token>
```

Endpoint de monitoreo. Devuelve el estado actual de la cola en Redis y si el worker está procesando algo en este momento. Útil para health checks y para detectar acumulación de trabajos pendientes.

```
{
    "queue_length": 0,
    "processing": false
}
```

Campo	Significado
`queue_length`	Cantidad de tickets esperando ser procesados por el worker.
`processing`	true si el worker está ejecutando un scraping en este momento, false si está libre.

1.5 Caché de resultados

Cada resultado exitoso se almacena en caché (Redis) durante **1 hora** indexado por número de passport. Durante esa ventana, cualquier nueva solicitud para el mismo passport responde inmediatamente con el dato cacheado y no encola ningún trabajo.

La razón: las competencias DC3 no cambian con frecuencia, y evitar consultas redundantes al portal externo reduce carga sobre Mobilise-Me y mejora los tiempos de respuesta para el usuario.

1.6 Flujo recomendado del cliente

El consumidor de la API (en este caso el bot de WhatsApp) sigue este patrón:

- Llamar al endpoint 1 (POST /search/) con el passport.
- Si la respuesta trae status=completed, usar el resultado.
- Si trae status=pending, guardar el ticket_id y hacer polling al endpoint 2 (GET /result/<ticket_id>/) cada cierto intervalo (ej: cada 2 segundos) hasta recibir status=completed.
- Mostrar al usuario el contenido de result.competencies.

El tiempo total típico de respuesta cuando no hay caché es de entre 5 y 15 segundos, dependiendo de la carga del portal externo.

2. Punto de partida

El portal de Mobilise-Me utilizado por Newmont Peñasquito está disponible en:

```
https://testing.newmontpenasquito.mobilise-me.com
```

La meta es: dado un Passport (ID de empleado, por ejemplo E000055821), obtener el listado de sus competencias DC3 directamente desde el portal, navegando como si fuéramos un administrador.

Para descubrir el flujo se navegó manualmente con un navegador, observando con DevTools (pestaña Network) qué peticiones se hacen, qué cookies se generan, qué formularios se envían y a qué URL se redirige cada paso. El proceso se replicó con peticiones HTTP secuenciales, manteniendo una única sesión (cookies vivas) durante toda la operación.

Lo que sigue es la crónica de cómo se hace el scraping, de inicio a fin, con los nombres reales de campos, URLs, valores y códigos de respuesta observados.

3. Paso 1 — Abrir la página de Login

Lo primero es entrar a `/Login` con un GET. El servidor responde con HTTP 200 y entrega el HTML del formulario, y de paso establece las cookies iniciales de sesión.

```
GET https://testing.newmontpenasquito.mobilise-me.com/Login
```

De esta primera respuesta nos importan dos cosas:

- **Las cookies que el servidor establece.** Aparecen en los headers Set-Cookie e incluyen al menos: `ci_session` (sesión del backend), `csrf_cookie_name` (token CSRF) y `lang` (idioma de la sesión). Estas cookies hay que conservarlas y enviarlas en todas las peticiones siguientes.
- **El token CSRF dentro del HTML del formulario.** Inspeccionando el formulario de login encontramos un input oculto:

```
<input type="hidden" name="csrf_test_name" value="..."/>
```

Ese valor (el `value` del input `csrf_test_name`) debe leerse del HTML y reenviarse junto al usuario y la contraseña en el siguiente POST. Si no, el servidor rechaza la autenticación.

Detalle: hay dos cosas relacionadas con CSRF que es fácil confundir. Una es la `cookie` llamada `csrf_cookie_name` (la maneja el cliente HTTP automáticamente al guardar las cookies). Otra es el `input hidden` del formulario llamado `csrf_test_name`, cuyo valor se envía en el body. Ambos deben coincidir, y el servidor compara uno contra el otro.

4. Paso 2 — Enviar el formulario de Login

Con las cookies obtenidas y el CSRF leído del HTML, simulamos el envío del formulario haciendo un POST a la misma URL `/Login`. El body se envía como `application/x-www-form-urlencoded` con los siguientes campos:

Campo (name)	Valor
`csrf_test_name`	Token CSRF leído del HTML del paso 1.

Campo (name)	Valor
`USERNAME`	Usuario administrador (variable de entorno NEWMONT_DC3_USERNAME). En desarrollo se utilizó E000055821.
`PWD`	Contraseña del usuario (variable de entorno NEWMONT_DC3_PASSWORD).
`action`	Login (este valor corresponde al name/value del botón de envío del formulario).

Headers importantes en este POST:

- `Content-Type: application/x-www-form-urlencoded`
- `Referer: https://testing.newmontpenasquito.mobilise-me.com/Login`
- User-Agent estándar de navegador, y todas las cookies del paso 1.

Qué responde el servidor

Lo que se observó en el log del cliente HTTP fue:

```
POST /Login HTTP/1.1" 303 0
GET /Selection HTTP/1.1" 200 1957
```

El servidor responde 303 See Other y un header `Location` que apunta a `/Selection`. Como el cliente HTTP sigue redirecciones automáticamente, termina haciendo un GET a `/Selection` y obtiene HTTP 200 con el HTML de esa página.

Punto clave: aquí descubrimos que el portal NO te lleva directamente al panel de administración después del login. Te deja en una pantalla intermedia llamada `/Selection` donde tienes que elegir con qué "perfil" o "contexto" vas a operar. Sin pasar por esa pantalla cualquier intento posterior de entrar al área administrativa termina rechazado.

5. Paso 3 — La pantalla de Selección (el paso clave)

Inspeccionando el HTML de `/Selection` se encontraron dos botones, cada uno dentro de su propio contenedor. El que nos interesa es el de la base de datos administrativa:

```
<button type="submit" class="btn selection-btn"
        name="action"
        value="Administration Database">
    <span class="selection-text">Base de datos administrativa</span>
</button>
```

Y el otro:

```
<button type="submit" class="btn selection-btn"
        name="action"
        value="Induction portal">
    <span class="selection-text">Continua a la Inducción</span>
</button>
```

Diferencia importante: la opción `Induction portal` devuelve únicamente las competencias del usuario logueado. La opción `Administration Database` abre el panel de administración donde se pueden consultar los DC3 de cualquier empleado, que es justamente lo que necesitamos.

Por qué este paso fue el principal dolor de cabeza

La primera versión del scraper saltaba directo del POST de login al panel `/Admin/...`. El servidor respondía con HTTP 307 redirigiendo de regreso a `/Selection`, o entregaba HTML de la propia pantalla de selección. Tardamos en darnos cuenta de que el portal exige declarar explícitamente el contexto antes de acceder al área de administración. Una vez identificado, agregar esta petición destrabó todo el flujo.

La petición

Hacemos un POST a `/Selection` simulando el click del botón "Administration Database". El body se envía como `application/x-www-form-urlencoded` con:

Campo (name)	Valor
`csrf_test_name`	Token CSRF leído de la cookie csrf_cookie_name (que es el valor vigente en este punto del flujo).
`action`	Administration Database (idéntico al value del button, sensible a mayúsculas y al espacio).

Headers importantes:

- `Referer: https://testing.newmontpenasquito.mobilise-me.com/Selection`
- `Content-Type: application/x-www-form-urlencoded`

Qué responde el servidor

```
POST /Selection HTTP/1.1" 303 0
GET /Admin/index/Users HTTP/1.1" 200 28500
```

Otra vez 303, y otra vez seguimos la redirección. El servidor nos lleva a `/Admin/index/Users` (la página de inicio del panel administrativo). Ya estamos dentro: la sesión queda "contextualizada" como administrador y a partir de aquí podemos navegar a cualquier subsección del módulo `Admin`.

6. Paso 4 — Localizar el módulo DC3

Dentro del HTML de `/Admin/index/Users` hay un menú lateral con todas las secciones. La que nos interesa aparece como un enlace:

```
<li><a href="https://.../Admin/index/Dc3 Formats Records">Registros Formatos
Dc3</a></li>
```

La URL del módulo (con el espacio codificado como `%20`) es:

```
https://testing.newmontpenasquito.mobilise-me.com/Admin/index/Dc3%20Formats%20Records
```

Hacemos un GET a esa URL y obtenemos HTTP 200 con la página completa del módulo: una tabla vacía (porque aún no hemos buscado nada) y, sobre ella, un formulario de búsqueda con varios filtros.

Importante: hay que volver a leer el CSRF de este HTML, porque el portal lo refresca al cargar la página. El siguiente POST de búsqueda usará este token nuevo.

7. Paso 5 — Buscar las competencias por Passport

Ahora simulamos el envío del formulario de búsqueda. El detalle particular de este formulario es que se envía como `multipart/form-data` (no como urlencoded), que es lo que hace el portal por defecto cuando se envía desde el navegador. Replicamos ese mismo encoding.

Petición

```
POST /Admin/index/Dc3%20Formats%20Records
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryDC3
```

Campos del body:

Campo (name)	Valor
`csrf_test_name`	Token CSRF leído del HTML de la página DC3 que se cargó en el paso 4.
`search[data][FOLIO_NUMBER]`	Número de Passport del empleado a consultar (ej: E000055821). Este es el filtro principal.
`search[data][COMPANY]`	--Show All-- (literal, así viene por defecto en el form).
`search[data][STATUS]`	--Show All-- (literal).
`action[search]`	(vacío) — corresponde al botón "Search" del formulario.

Nota sobre el name de los inputs: los campos vienen con notación tipo array PHP/CodeIgniter: `search[data][FOLIO_NUMBER]`, `search[data][COMPANY]`, etc. Hay que respetarlos exactamente; si se envía solo `FOLIO_NUMBER` el backend no lo reconoce.

Detección de sesión perdida

Antes de parsear, verificamos que la respuesta no nos haya tirado al login. Si la URL final de la respuesta contiene `/Login`, la sesión expiró: hay que repetir todo el flujo desde el paso 1. El scraper marca al usuario como no autenticado y reintenta el login una vez antes de devolver error.

8. Paso 6 — Parsear la tabla de resultados

La respuesta del POST de búsqueda trae un HTML que contiene la tabla con los registros DC3 del empleado. La identificamos por su clase:

```
<table class="admin-table"> ... </table>
```

Iteramos por las filas del `<tbody>` y, de cada fila, extraemos las celdas `<td>` por su posición. La estructura es la siguiente:

Índice de td	Contenido	Uso en la API
0	Botón de acción	Contiene un <button data-fun="verify" data-id="..."> del que extraemos el record_id (data-id) que servirá para pedir el documento PDF.

Índice de td	Contenido	Uso en la API
6	Nombre de la actividad/competencia	Campo actividad (ej: "Newmont Peñasquito Visitor Induction I1").
7	Fecha de solicitud	Campo fecha_solicitud.
8	Fecha de expiración	Campo fecha_expiracion. Si la celda contiene "---" se interpreta como sin expiración (null).
10	Estado	Campo status (ej: vigente, vencida).

Filtro defensivo: se ignoran las filas que tienen menos de 11 celdas. Eso protege contra filas de cabecera, separadores o filas vacías cuando el portal no encuentra registros.

Si después de iterar la lista de competencias está vacía, devolvemos un resultado con error "Sin competencias". Esto puede significar que el Passport no existe, o que ese empleado no tiene DC3 registrados.

9. Paso 7 — Obtener la URL del documento PDF (opcional)

Cada fila de la tabla tiene asociado un PDF (la constancia DC3 firmada). En el portal, al hacer click en el botón `verify` de la columna 0, el frontend hace una petición AJAX a otro endpoint y abre el documento en un modal. Replicamos esa misma petición.

Petición

```
POST /DC3/index
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Referer: .../Admin/index/Dc3%20Formats%20Records
```

Campo	Valor
`id`	record_id extraído del data-id del botón verify de la fila.
`table`	`DC3_FORMATS_RECORDS`
`function`	`verify`
`induction`	`false`
`csrf_test_name`	Token CSRF vigente.

Respuesta

Esta petición devuelve **JSON** (no HTML). Dentro del JSON, en la ruta `data.FILE`, está el nombre del archivo PDF. La URL final del documento se construye así:

```
{BASE_URL}/DC3/showDoc/DC3_FORMATS_RECORDS/dc3/{nombre_archivo}
```

Esa URL puede compartirse con el empleado para que descargue su constancia.

10. Resumen del flujo completo

Visto en orden, el flujo es:

#	Método	URL	Qué hace
1	GET	`/Login`	Inicializa cookies y obtiene CSRF del form.
2	POST	`/Login`	Envía USERNAME, PWD, action=Login. Responde 303 → redirige a /Selection.
3	POST	`/Selection`	Envía action=Administration Database. Responde 303 → redirige a /Admin/index/Users.
4	GET	`/Admin/index/Dc3%20Formats%20Records`	Carga la página del módulo y refresca el CSRF.
5	POST	`/Admin/index/Dc3%20Formats%20Records`	Envía búsqueda con search[data][FOLIO_NUMBER]=passport. Devuelve HTML con la tabla.
6	(parseo)	`table.admin-table`	Extrae actividad, fechas, status y record_id de cada fila.
7	POST	`/DC3/index`	(opcional) por cada record_id, obtiene el JSON con el nombre del archivo PDF.

11. Puntos críticos

Estos son los detalles que es fácil pasar por alto y que rompen el scraping si se omiten:

- **Mantener una única sesión HTTP.** Las cookies (especialmente `ci_session` y `csrf_cookie_name`) deben persistir durante todo el flujo. Si el cliente las descarta entre peticiones, todo falla.
- **El paso /Selection no se puede saltar.** Sin él, cualquier acceso a `/Admin/...` termina redirigido (HTTP 307) o devuelto al login.
- **El valor exacto "Administration Database".** Es sensible al espacio y a las mayúsculas. Coincide con el atributo `value` del botón en el HTML de `/Selection`.
- **El CSRF se refresca por página.** Hay que leer el `csrf_test_name` del HTML de la página actual antes de cada POST, no reutilizar el del paso anterior. La cookie `csrf_cookie_name` también se actualiza.
- **El header Referer es obligatorio en los POST.** Cada POST debe declarar como `Referer` la URL desde la que viene el formulario. Sin él, el portal puede rechazar la petición.
- **La búsqueda DC3 va como multipart, no urlencoded.** Es el detalle más fácil de equivocar. Hay que usar `multipart/form-data` con boundary. En urlencoded el portal no procesa los filtros.
- **Los nombres de campo del filtro tienen notación de array.** Son `search[data][FOLIO_NUMBER]`, `search[data][COMPANY]`, `search[data][STATUS]`. Y los valores de COMPANY y STATUS, cuando no se filtran, van con el literal `--Show All--`.

- **Detectar sesión perdida.** Si después de un GET o un POST la URL final contiene `/Login`, la sesión expiró y hay que volver a empezar desde el paso 1.
- **Idioma fijo.** Forzar la cookie `lang` mantiene los textos del HTML predecibles para el parseo.