QR

por Berrones-vera Berrones-vera

Fecha de entrega: 01-oct-2019 09:42a.m. (UTC-0500)
Identificador de la entrega: 1183860963
Nombre del archivo: DOCUMENTO_ACTIVIDADES_QR_FINAL_TURNITIN.docx (3.43M)
Total de palabras: 7054
Total de caracteres: 38818

INTRODUCCIÓN

Por medio de la técnica de la observación es fácil deducir que tanto para los estudiantes, como para los docentes que, las actividades complementarias en la Universidad Estatal de Milagro, generan malestar entre la comunidad, especialmente cuando se debe controlar la asistencia de los estudiantes a estas actividades.

Este proceso, es ejecutado mediante listas impresas, las cuales provocan pérdida de tiempo, desperdicio de materiales como papel, tóner de impresora y demás elementos asociados a procesos manuales de control y administración.

En virtud de lo expuesto, hemos planteado un proyecto que tiene como finalidad reducir el tiempo y mejorar los procesos de control y asistencia de los estudiantes, para incrementar la fluidez de la información durante la realización de las actividades complementarias.

El objetivo de este proyecto, es la implementación de un sistema que automatice el proceso de control de asistencia de las actividades complementarias con el fin de reducir tiempo y optimizar el trabajo. Todo esto en virtud que este proceso consume mucho tiempo. Nuestra propuesta consiste en generar una aplicación que asigne un código de identificación del estudiante en función de la actividad complementaria que realiza. Este código es generado d manera individual y utiliza tecnología QR portable en medios digitales portátiles (Teléfonos y tablets). Los códigos QR son únicos y están ligados al sistema de gestión académica de la universidad, para facilitar la recolección de datos y generar información, permitiendo que con solo mostrar el código generado y el docente sea capaz de gestionarlo a través de una cámara digital en su teléfono celular, automatizando el proceso de asistencia de los estudiantes a las actividades complementarias.

CAPÍTULO 1.

Planteamiento del problema

En la Universidad Estatal de Milagro (UNEMI), como parte de la formación integral de profesionales, se realizan un conjunto de actividades extracurriculares relacionadas con el deporte, la ciencia y el arte. Estas actividades se denominan Actividades Complementarias. En este sentido, el reglamento interno de la UNEMI, exige que todos los estudiantes realicen estas actividades como requisito previo a su graduación.

Es por ello que, mediante la observación y deducción de los procesos asociados a las actividades complementarias que se realizan en la UNEMI, logramos identificar que existe un problema asociado a la pérdida de tiempo de los docentes, durante el registro de asistencia de los estudiantes a las actividades complementarias, todo esto, porque este proceso se lo realiza de a través de medios manuales. En consecuencia de ello, consideramos oportuno que en la Universidad Estatal de Milagro se implemente un sistema automatizado que mejore la eficiencia del proceso de control de las actividades complementarias que realizan los estudiantes.

Objetivos

Objetivo General

Desarrollar una tecnología de códigos QR para gestión de procesos asociados a las actividades complementarias de los estudiantes de la Universidad Estatal de Milagro.

Objetivos Específicos

- Aprovechar el crecimiento de la tecnología QR para agilizar procesos engorrosos.
- Definir una estructura eficiente de generación y lectura de códigos QR.
- Adaptar el desarrollo al funcionamiento del sistema de gestión académico.

Justificación

La inconsistencia que presentan las listas impresas firmadas por los asistentes a las actividades complementarias supone una pérdida de tiempo para docentes y secretarias encargados de registrar la asistencia en el sistema, en muchas ocasiones por la ilegibilidad de la lista, es poco confiable creer que dicha firma pertenece a un alumno y se vuelve un problema verificar la validez. Por otra parte, los alumnos como beneficiarios de las actividades complementarias presentan muchas quejas debido a la intransigencia al firmar la lista y el tiempo que se pierde mientras una fila de aproximadamente cuarenta alumnos intenta registrar su asistencia.

Con el avance acelerado de la tecnología y la mejora de las condiciones de vida que su implementación propone, la presente investigación tiene como objetivo el desarrollo de una tecnología que aproveche las bondades de los códigos QR en procesos asociados con las actividades complementarias de la Universidad Estatal de Milagro.

El crecimiento en el uso de esta tecnología se debe a la valiosa información que puede contener dentro de sus píxeles y a su velocidad de respuesta, por ello el nombre QR (quick response o respuesta rápida).

La tecnología de códigos QR es usada en campos como el marketing para proponer una comunicación asertiva con el cliente porque permite guardar información de éste para luego ser usada en servicio post venta, en la gestión de inventarios para agilizar la localización en percha de los productos o como sucesión del código de barras debido a que puede contener más información, en los inicios de sesión rápidos para evitar el usuario y contraseña que resultan ser volátiles en nuestra memoria. Actualmente la educación está incursionando en el uso de esta tecnología para realizar eventos y dictar conferencias con una aplicación de terceros denominada Eventbrite.

El desarrollo del presente proyecto también pretende sentar precedentes del uso de los códigos QR en demás áreas de la universidad y la educación. Con miras a agilizar los procesos que se llevan a cabo para realizar dicha actividad y aumentar la fiabilidad de la información.

CAPÍTULO 2

Antecedentes

El avance tecnológico ha permitido que tecnología usada desde los 50's como los códigos de barra que consistían en información resumida en una estructura unidimensional y que estaban implementadas en sistemas de boleterías dentro de los terminales de transporte o en la parte posterior de algunos productos, evolucionen a una tecnología más rápida y que permita el almacenamiento de más información denominada QR (quick response).

Los QR almacenan la información en dos dimensiones, permitiendo la integración de un mayor número de caracteres (hasta 7.089) los cuales sus antecesores no podían manejarlos y además la mejora de algoritmo de generación y lectura. Lo que permitió que grandes compañías con complejos sistemas transaccionales manejan esta nueva tecnología por su simple manejo, mayor capacidad de almacenamiento, fácil traducción y un sinnúmero de librerías que permiten acoplar esta tecnología con diferentes lenguajes (Huidobro, 2009).

Múltiples aplicaciones y uso de la tecnología QR han sido llevadas a cabo en el continente asiático, lugar donde fue desarrollada. Su uso no es solo comercial, pero es el campo donde más destaca. Una de las implementaciones más admirables que este continente ha logrado dar a la tecnología QR son las compras por internet, los anuncios o vallas publicitarias con productos novedosos o de primera necesidad que cuentan con un código QR en la descripción, permiten comprarlos solo con la necesidad de escanear el código. Estos anuncios pueden ser vistos dentro de centros comerciales hasta en vagones del tren y estaciones de buses, lo que permite la compra rápida y desde casi cualquier lugar. Esta tecnología combinada con un sistema de entregas eficiente nos demuestra el potencial que tiene la tecnología en la mejora de nuestras vidas (García & Okazaki, 2012).

Una investigación demostró que la tecnología de códigos QR permite incrustar imágenes (logos e incluso fotografías de mayores dimensiones) sin perder legibilidad. La investigación desarrolló una serie de pruebas que dieron resultados positivos. El código a pesar de estar parcialmente dañado retorna el mismo valor con el que fue creado, es decir, la misma cantidad de caracteres y el mismo orden. Esto se debe a que los píxeles más grandes: localización, alineamiento y sincronización, facilitan la lectura y aseguran la integridad de la información. El procedimiento para crear códigos QR con imágenes, consiste en colocar

de fondo una imagen y con un leve degradado superponer el código QR, lo que permitirá la lectura íntegra y un diseño agradable. (Villarrea & Villamizar, 2013).

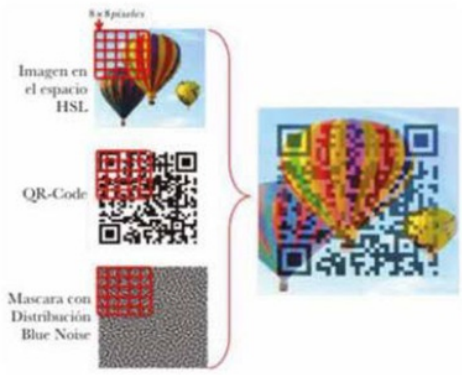

Ilustración 1. Imagen incrustada en código QR (Villarrea & Villamizar, 2013).

La tecnología de códigos QR ha permitido agilizar procesos de toma de asistencia a través de un código impreso en un carnet, esta investigación se llevó a cabo con el objetivo de reducir los datos impresos en la tarjeta dejando al descubierto solo información relevante como el nombre para que personas alejadas a la institución donde se desarrolló la investigación puedan reconocer al portador de la tarjeta. (Gonzalez & Garcia, 2016).

Ilustración 2. Propuesta de reducción de información usando código QR (Gonzalez & Garcia, 2016).

El uso de códigos QR dentro de múltiples áreas es bastante aceptado. Un estudio realizado en una institución dentro del área de educación física, consiste en la implementación de una aplicación que permite leer un código QR que tiene como resultado una URL que dirige a una página web con el ejercicio indicado. Esta página web contiene una serie de vídeos con diferentes rutinas como calentamiento, entrenamiento, entre otras. El desarrollo del proyecto contó con la intervención de estudiantes y personal docente. La metodología aplicada para el desarrollo en equipo se basó en tres ejes:

- Científico: a través de algoritmos de programación se logró el desarrollo de la página web que contenía los vídeos, así como la generación y lectura de códigos QR.
- Colaborativo: al ser un proyecto de gran alcance para una institución, se manejaban a través de roles, cada grupo estaba encargado de distintos tópicos, lo que permitía avanzar rápido con el desarrollo y obtener retroalimentación del aprendizaje grupal.
- Integración tecnológica: en el transcurso del desarrollo, tanto alumnos como docentes aprendían múltiples herramientas que permitieron enriquecer su conocimiento (Izquierdo, 2009).

El desarrollo de múltiples herramientas tecnológicas que tienen compatibilidad entre sí, es lo mejor y más cuando los desarrolladores fueron estudiantes, esto significa que la elaboración de sistemas se puede llevar a cabo a través de un seguimiento.

Una investigación realizada por estudiantes de la universidad de Sevilla estaba encargada de la generación de códigos QR para que los docentes, por medio de cursos de adaptación tecnológica sean capaz de guiarse hacia las diferentes ubicaciones que proporcionan los códigos. En puntos estratégicos se colocaban pancartas donde cada facultad contaba con una dirección web (link), docentes y estudiantes ingresaban a la sección que pertenecían, el sitio web contaba con una interfaz de cámara que permitía registrar la asistencia. Esta propuesta agilizó el proceso de la asistencia y dejar de lado los medios manuales. Permitiendo a la universidad a futuro realizar mejores implementaciones con la finalidad de optimizar procesos múltiples (Graván & Gutiérrez, 2013).

Estos códigos no solo permiten la optimización de tiempo dentro de las organizaciones y las compras por internet, también se pueden implementar en la enseñanza, permitiendo que así los libros físicos y apuntes queden como segunda opción o se fomente una clase colaborativa. Esta propuesta consistía en detallar la solución de ejercicios de matemáticas por medio de un código QR, al ser leído dirigía a una sección del libro que contenía la respuesta. De esta forma, el personal encargado podía hacer un seguimiento a los estudiantes cuando ingresaban por la respuesta (Cassanova & Molina, 2013).

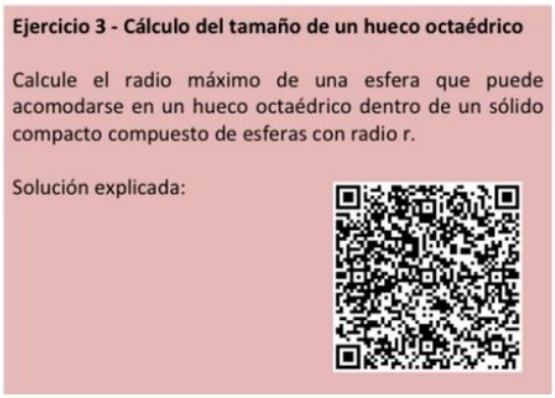

Ilustración 3. Aplicación de códigos QR en ejercicios prácticos para estudiantes (Cassanova & Molina, 2013).

Marco Teórico

Python

Lenguaje de programación de alto nivel, permite a los desarrolladores una mayor independencia y adaptabilidad debido a su curva de aprendizaje corta. Se ejecuta a través de un intérprete y su estructura es de scripting. Por ser de estructura sencilla y curva de aprendizaje corta, combinado con librerías permite agilizar el desarrollo de aplicaciones que no empleen el uso de grandes operaciones. Al ser un lenguaje de código abierto, le ha permitido poseer una extensa gama de librerías que brindan la posibilidad de crear trabajos excepcionales. (Marzal Varó & Gracia Luengo, 2014).

Múltiples proyectos se han llevado a cabo de la mano de este lenguaje, algunos han marcado la historia de internet, como es el caso de Google que hasta la actualidad desarrolla bajo esta plataforma y ha creado librerías importantes como Tensorflow. Un uso icónico fue en la producción de la película "Star Wars: Episodio II" el cual requería el manejo rápido de lotes de información que se ven reflejadas en composiciones de escenas.

Django

Es un framework de desarrollo web, que funciona bajo la estructura de Python. Es un optimizador de tiempo y trabajo, debido a que permite el desarrollo de aplicaciones bajo una estructura simple, organizada e independiente. Su abstracción ofrece al desarrollador múltiples patrones de desarrollo y funcionamiento básico para que éste se centre en la funcionalidad núcleo de la aplicación. Al estar basado en Python, la consola es el núcleo de su funcionamiento, lo que da paso a configuraciones que en un futuro serán implementadas

en un servidor, por lo que su adaptabilidad a varios ámbitos es amplia (Holovaty & Kaplan, 2008).

Aunque sea un framework que preserve el principio de independencia, Django cuenta con archivos de configuraciones necesarios para su funcionamiento, los cuales son:

- **Settings.py:** Su objetivo es contener las configuraciones para el funcionamiento del proyecto, tales como acceso a base de datos (PostgresQL), librerías, dirección de templates de diseño, templates de manejos de errores (404, 500, 403), dirección de archivos estáticos y multimedia. (Django Software Foundation, 2019).

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'templates'),
            os.path.join(BASE_DIR, 'templates', 'base'),
            os.path.join(BASE_DIR, 'templates', 'rrhh'),
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Ilustración 4. Configuración de templates en el archivo Settings

- **Models.py:** Archivo que permite la creación de la estructura de una base de datos en donde cada clase representa a una tabla dentro de la misma, realizan funciones extras las cuales son el retorno de datos o validaciones antes de una inserción/modificación/eliminación, cada aplicación dentro del proyecto contiene un archivo Models.py. El manejo DLL de estos modelos se los realiza con un lenguaje de manejo de base de datos basado en Python denominado ORM.

```
class Empleado(models.Model):
    nombre = models.CharField(max_length=50)
    apellido = models.CharField(max_length=50)
    edad = models.IntegerField()
    telefono  = models.CharField(max_length=12)
    email = models.EmailField()
    domicilio = models.TextField()
    estado = models.BooleanField()

    def __str__(self):
        if self.estado == True:
            return '{} {}'.format(self.nombre, self.apellido)
        else:
            return ''
```

Ilustración 5. Ejemplo de creación de modelo

- **Urls.py:** Define la estructura de interacción entre el usuario y la aplicación, donde por cada URL recibida Django responderá con la vista correcta. Con vista nos referimos al controlador que responderá una petición HTTP.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('rrhh/', include('apps.rrhh.urls'), name='rrhh'),
]
```

Ilustración 6. Ejemplo de llamados de URLs a vistas

- **Vistas:** Archivos que funciona tras la llamada de las URLs, se lo conoce como el controlador porque recibe una petición HTTP y retorna una respuesta del mismo tipo tras realizar sus operaciones, trabaja en medio del cliente y el servidor (Holovaty & Kaplan Mosss, 2015).

```
class ContratoCreate(CreateView):
    model = Contrato
    template_name = 'contrato_form.html'
    form_class = ContratoForm
    second_form_class = EmpleadoForm
    success_url = reverse_lazy('rrhh:empleado_listar')

    def get_context_data(self, **kwargs):
        context = super(ContratoCreate, self).get_context_data(**kwargs)
        if 'form' not in context:
            context['form'] = self.form_class(self.request.GET)
        if 'form2' not in context:
            context['form2'] = self.second_form_class(self.request.GET)
        return context

    def post(self, request, *args, **kwargs):
        self.object = self.get_object
        form = self.form_class(request.POST)
        form2 = self.second_form_class(request.POST)
        print(form.errors)
        if form.is_valid() and form2.is_valid():
            rrhh = form.save(commit=False)
            rrhh.empleado = form2.save()
            form.save()
            return HttpResponseRedirect(self.get_success_url())
        else:
            return self.render_to_response(self.get_context_data(form=form, form2=form2))
```

Ilustración 7. Ejemplo de vista para creación de registro

HTML

Es un lenguaje de maquetado de etiquetas, define el desarrollo de la interfaz web, se ejecuta a nivel de cliente, es decir, en los navegadores (Chrome, Opera, Firefox, entre otros). Permite la interacción con hojas de estilo en cascada denominada CSS que brinda un diseño atractivo a la página. En combinación con HTML y CSS trabaja JavaScript, un lenguaje de cliente que permite mejorar la interacción del usuario con el sitio. (Anibarro Z, 2001).

CAPÍTULO 3

DESCRIPCIÓN DEL PROYECTO TÉCNICO

Para cumplir con la finalidad del proyecto, se procedió a recrear el **módulo de actividades complementarias**. Existen dos roles de usuarios dentro de este módulo.

El **docente** es el encargado de controlar las actividades que le fueron asignadas a través de la pantalla principal del módulo, ingresando a través del icono Actividades Complementarias en el menú. Donde podrá registrar la asistencia (normal y a través de la lectura del código QR) y calificar las actividades. O ingresando al módulo Actividades QR como un acceso directo al lector de códigos.

El **alumno** tendrá por deber presentarse a la actividad en el horario definido por el cronograma y mostrar el código QR al tutor para registrar su asistencia.

Ilustración 8. Listado de actividades complementarias del docente

Navegando por las acciones de cada actividad (Calificación / Asistencia), el **docente** podrá controlar el cronograma de asistencias, el cual consta de lugar, fecha, observación, número de asistentes y acciones.

Ilustración 9. Cronograma de horarios de la actividad.

Cabe recalcar que la generación de las actividades se las realizo a través del administrador de Django el cual permite realizar manipulación de datos directas sin necesidad de hacerlo con la aplicación, esto nos permite centrarnos en el desarrollo del módulo, la generación y lectura de código QR's. Al tener que desarrollarse sobre el Sistema de Gestión Académico, se presentó la necesidad de emular algunos modelos como el cronograma y la inscripción de estudiantes a las actividades.

Ilustración 10. Modelos recreados para el funcionamiento del proyecto

A través de Django se procedió a la creación de la base de datos, a este proceso se lo denomina "migración de modelos", consiste en aplicar los cambios hechos en los modelos a la base de datos especificada en el archivo Settings.py.

Este proceso permite al programador centrarse en el desarrollo del funcionamiento de la aplicación.

Una vez realizada la migración, Django crea tablas de configuraciones en la base de datos, las cuales serán usadas para el control de usuarios, sesiones, migraciones y permisos.

```python
class Actividad(models.Model):
    area = models.ForeignKey(Area, on_delete=models.PROTECT)
    facultad = models.ForeignKey(Facultad, on_delete=models.PROTECT)
    actividad = models.CharField(max_length=200)
    induccion = models.BooleanField(default=False)
    detalle = models.CharField(max_length=400)
    fechaIniInscripcion = models.DateField(verbose_name='Fecha de Inicio de Inscripción')
    fechaFinInscripcion = models.DateField(verbose_name='Fecha de Fin de Inscripción')
    cupo = models.IntegerField(default=0)
    tutorPrincipal = models.ForeignKey(Profesor, on_delete=models.PROTECT)
    calificar = models.BooleanField(default=False)
    calificacionSobre = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    calificacionAprobacion = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    minAsistencia = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    seccion = models.CharField(choices=SECCION, max_length=20)

    def __str__(self):
        return "{} : {} : {} | {} a {}".format(self.cupo, self.actividad, self.tutorPrincipal.persona.datos(),
                                               self.fechaIniInscripcion,
                                               self.fechaFinInscripcion)

    class Meta:
        verbose_name = 'Actividad'
        verbose_name_plural = "Actividades"
        ordering = ('area', 'actividad', 'fechaIniInscripcion', 'fechaFinInscripcion',)
```

Ilustración 11. Creación del modelo Actividad

Dicho anteriormente, al realizar el proceso de migración, Django crea 10 tablas necesarias para configuraciones. El resto de tablas son modelos para gestión del módulo de actividades complementarias, tales como actividades, profesor, estudiante, asistencia.

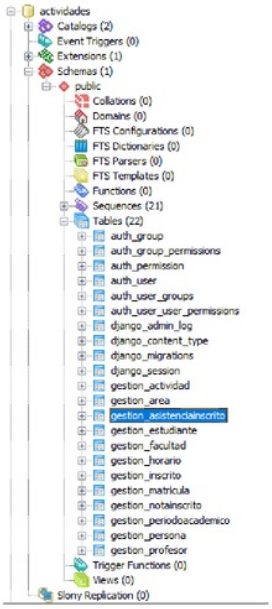

Ilustración 12.Listado de tablas en la base de datos

Adaptación

Al recrear el módulo "Actividades Complementarias" del Sistema de Gestión Académica, se presentó la necesidad de crear los modelos maestros, es decir, el nivel de abstracción superior, en una normalización de base de datos. Estos modelos permitirán el funcionamiento de la aplicación, entre los cuales están:

- PeriodoAcademico: Modelo que contiene el periodo académico. Necesario para la creación de actividades y alumnos.
- Facultad: Contiene el nombre de la facultad en la que se encuentra el estudiante para la asignación de las actividades complementarias.
- Persona: Usada como tabla principal para los datos de docente y alumno, al compartir características semejantes se usa para evitar entradas redundantes.
- Profesor: Contendrá la clave primaria para referencias y los datos que se encuentran ubicados en la tabla Persona.
- Estudiante: Contendrá su clave primaria en conjunto con los datos de Persona.
- Matricula: Es una tabla intermediaria la cual contiene claves foráneas que funcionan para crear la conexión matricula, los nombres a tablas que se refieren son a PeriodoAcademico y Estudiante.

Sistema

Los modelos mencionados anteriormente eran necesarios para el funcionamiento de la aplicación. Los modelos a continuación han sido rediseñados para adaptarse al nuevo proceso de registro de asistencias. Estos son:

- Actividad: Contiene información como los cupos, docente encargado, calificación mínima, fecha, detalle, facultad, entre otros. Es uno de pilares en la generación de códigos.
- Horario: Cronograma de las actividades, es decir, por cada actividad existirán uno o más registros de horarios. Otro de los pilares en la generación de códigos.
- Inscrito: Se realiza la conexión entre la matricula, la cual contiene al alumno y la actividad, es decir, cada registro se convierte en un alumno registrado en la actividad.
- NotaInscrito: Modelo que contendrá información de la nota al final de la actividad del alumno.
- AsistenciaInscrito: Por cada alumno inscrito en la actividad asigna el cronograma de la actividad, es decir, si la actividad tiene tres horarios, este modelo contendrá tres registros, uno por cada horario.

Settings.py

Archivo de configuraciones del proyecto, a continuación el detalle de su funcionamiento:

- La sección INSTALLED_APPS permite definir las aplicaciones instaladas en el proyecto, las que incluyen el nombre django_contrib son necesarias para acceder al panel de administrador, la autorización, sesiones, archivos estáticos entre otras. La que tiene por nombre gestión, contiene el funcionamiento del proyecto, como es la generación y lectura de códigos. Mientras que crispy_forms es una librería que brinda facilidad en el manejo de los formularios de Django.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'gestion',
    'crispy_forms',
    # 'django_extensions',
]
```

Ilustración 13. Sección de aplicaciones instaladas en el proyecto

- Al ser un archivo de configuraciones, dentro de su estructura se definen las variables globales para el proyecto, en esta sección las variables de sesión que permiten definir el tiempo máximo de sesión a través de la cookie, la URL del inicio de sesión, entre otras.

```
LOGIN_URL = '/login/'
LOGOUT_REDIRECT_URL = '/login/'
LOGIN_REDIRECT_URL = '/'
SESSION_EXPIRE_AT_BROWSER_CLOSE = True
SESSION_COOKIE_AGE = 17000
```

Ilustración 14. Variables globales relacionadas a la sesión de usuarios.

- La librería de crispy_forms trabaja bajo el esquema de Bootstrap, por ello es necesario definir la versión a utilizar. Mientras que la variable ROOT_URLCONF define el directorio raíz del archivo de configuración de URLs.

```
ROOT_URLCONF = 'actividades.urls'
CRISPY_TEMPLATE_PACK = 'bootstrap3'
```

Ilustración 15. Variables de configuración globales

- Para establecer una conexión con la base de datos en el archivo se crear un diccionario con cada uno de los parámetros que necesita el proyecto para la conexión,

estos van desde el motor que necesitará para conexión el cuales el psycopg2, se trata de un driver que permite la conexión entre la base de datos y Python – Django, el nombre de la base de datos a emplear, usuario, contraseña, host y puerto.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'actividades',
        'USER': 'postgres',
        'PASSWORD': '1234',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Ilustración 16. Diccionario con parámetros de conexión a base de datos PostgresQL

- Una de las últimas configuraciones a en el archivo Settings.py son las direcciones raíz de los archivos estáticos (templates HTML, JavaScript, CSS) y multimedia (fotos, vídeos, sonidos)

```
STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
STATIC_URL = '/static/'

MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'
```

Ilustración 17. Variables de dirección raíz de archivos estáticos y multimedia

Views.py

Las vistas más conocidas en un MVC (modelo/vista/controlador) como el controlador, permiten el manejo de la petición HTTP, es decir, la interacción entre el cliente (navegador) y el servidor (de aplicaciones, base de datos). Tiene por tarea responder a las peticiones HTTP, el esquema usado para el manejo de vistas es BASADAS EN CLASE las cuales permiten manipulación de métodos como POST y GET de una forma sutil y elegante, a través de sus TemplateView que son vistas con métodos predefinidos que pueden ser sobrescritos (concepto de la programación orientada a objetos), agilizando el desarrollo.
Las librerías a implementar dentro del archivo son de Django:

- **from django.contrib.auth import login:** Función de la aplicación de autorización de Django para inicios de sesión, recibe por parámetros un formulario de sesión y la petición HTTP. Dando como respuesta la autorización del login.
- **from django.contrib.auth.forms import AuthenticationForm:** Formulario heredado de la aplicación de autorización de Django.
- **from django.contrib.auth.mixins import LoginRequiredMixin:** Parámetro de la vista basada en clase que permite acceder al contenido solo si en la petición HTTP

existe un usuario con sesión iniciada. Caso contrario redireccionará a la URL de inicio de sesión.

- **from django.contrib.auth.views import LoginView, LogoutView:** Vistas basadas en clase de la aplicación de autorización de Django, permiten un inicio y cierre de sesión rápido sin necesidad de programar nuevamente la funcionalidad.
- **from django.core import signing:** Función de criptografía de Django, es capaz de recibir cadenas de texto simples y retornar una cadena encriptada y también de recibir estructuras complejas como un diccionario en formato JSON.
- **from django.db import transaction:** Gestiona las transacciones realizadas en la base de datos, es de suma importancia para resguardar la consistencia de la información.
- **from django.http import HttpResponseRedirect, HttpResponse:** Clases que permiten respuestas HTTP de redirección y con retorno de un contexto, definimos a contexto como un resultado de la petición (modelo extraído de la base de datos, un objeto JSON, una lista de objetos, entre otros). Mientras que la respuesta de redirección, reenvía al usuario a una URL.
- **from django.urls import reverse_lazy:** Función que permite obtener la URL especificando su nombre, es decir, si una URL tiene por nombre listado, al usar esta función retornará la dirección "/gestión/listado".
- **from django.views import View:** La clase genérica de las vistas basadas en clase, a través de una herencia permite acceder a los atributos y métodos para sobrescribirlos o sobrecargarlos.
- **from django.views.generic import TemplateView:** Una clase genérica de las vistas basadas en clases que permiten manipular una template HTML, sus métodos son muy básicos.
- **import json:** Permite la creación de objetos con estructura JSON.

```python
import json
from datetime import datetime

from django.contrib.auth import login
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth.views import LoginView, LogoutView
from django.core import signing
from django.db import transaction
from django.http import HttpResponseRedirect, HttpResponse
# Create your views here.
from django.urls import reverse_lazy
from django.views import View
from django.views.generic import TemplateView

from gestion.models import *
```

Ilustración 18. Librerías utilizadas para la generación de vistas

A continuación las vistas usadas en el proyecto, dentro del archivo Views.py:

1. **CerrarSesionView (LogoutView):** Al ser una template de cierre de sesión, permite redefinir sus métodos cumplir su objetivo, sin necesidad de volver a programar.
2. **IniciarSesionView (LoginView):** Template de inicio de sesión, al sobrescribir el método get_context_data se puede parametrizar al contexto HTTP con más información. Cuando el usuario inicia sesión, se llama al método form_valid y procede a redefinirlo, para agregar información de los roles de usuario. Como es una recreación del SGA, los roles están definidos por un número, 1 para profesor, 2 para estudiante, 3 para el administrador. Si el inicio de sesión es correcto y se ha guardado información del rol de usuario en la variable de sesión de la aplicación, se procede a retornar una respuesta HTTP de redirección con la URL "home".

```python
class IniciarSesionView(LoginView):
    form_class = AuthenticationForm
    template_name = 'login.html'
    success_url = reverse_lazy('home')
    redirect_field_name = 'next'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['h'] = datetime.now()
        return context

    def form_valid(self, form):
        """Security check complete. Log the user in."""
        login(self.request, form.get_user())
        persona = None
        tipo = 0  # 1 -> profesor, 2 -> estudiante, 3 -> admin
        if Estudiante.objects.filter(usuario=form.get_user()).exists():
            self.request.session['tipo'] = 2
            self.request.session['spk'] = Estudiante.objects.filter(usuario=form.get_user())[0].id
        elif Profesor.objects.filter(usuario=form.get_user()).exists():
            self.request.session['tipo'] = 1
            self.request.session['spk'] = Profesor.objects.filter(usuario=form.get_user())[0].id
        else:
            self.request.session['tipo'] = 3
            self.request.session['spk'] = Persona.objects.filter(correo='cverac2@unemi.edu.ec')[0].id
        return HttpResponseRedirect(self.get_success_url())
```

Ilustración 19. Vista para el inicio de sesión

3. **HomeView (LoginRequiredMixin, TemplateView):** Contiene dos parámetros y el primero tiene la finalidad que comprobar que hay un usuario con sesión iniciada, en caso de no estarlo, lo redirecciona al inicio de sesión. El segundo parámetro define a la vista como una vista de template, que contiene parámetros básicos como el nombre del template HTML y un método de contexto GET, el cual es sobrescrito y se le añade el listado de periodos académicos junto con la hora del sistema, ambos para efectos de presentación del menú.

```
class HomeView(LoginRequiredMixin, TemplateView):
    template_name = 'menu.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['periodos'] = PeriodoAcademico.objects.all()
        context['h'] = datetime.now()
        return context
```

Ilustración 20. Vista del menú

4. **ActividadList (LoginRequiredMixin, TemplateView):** Al igual que la vista anterior, contiene los mismos parámetros y funcionamiento. La diferencia se centra en el contexto get retornado y a la template HTML. Si el tipo de sesión es 1, es decir, docente. Se procede a cambiar el parámetro de la template HTML por "list_profesor.html", y añadir al contexto una lista de actividades filtradas por el docente. En caso de ser el tipo de sesión 2, la template HTML cambia a "list_alumno.html" y se añade el periodo académico y las actividades filtradas por el alumno al contexto.

```
class ActividadList(LoginRequiredMixin, TemplateView):
    template_name = 'list_alumno.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['periodos'] = PeriodoAcademico.objects.all()
        context['h'] = datetime.now()
        tipo = self.request.session['tipo']
        spk = self.request.session['spk']
        if tipo == 1:
            self.template_name = 'list_profesor.html'
            actividades = Actividad.objects.filter(tutorPrincipal__id=spk)
            context['actividades'] = actividades
        elif tipo == 2:
            self.template_name = 'list_alumno.html'
            actividades = []
            for a in Inscrito.objects.filter(matricula__estudiante__id=spk):
                actividades.append(a.actividad)
            context['periodo'] = Matricula.objects.filter(estudiante__id=spk)[0].periodoAcademico
            context['actividades'] = actividades
        return context
```

Ilustración 21. Vista del módulo de actividades

5. **CronogramaList (LoginRequiredMixin, TemplateView):** Vista dedicada al docente, verificamos que sea la sesión de un docente revisando si el rol es igual a 1, obtenemos la clave primaria de la actividad que fue enviado en los parámetros de la petición GET, añadimos la actividad y los horarios de la misma al contexto. Si el rol no es de docente se procede a retornar el contexto vacío.

```
class CronogramaList(LoginRequiredMixin, TemplateView):
    template_name = 'cronograma_asistencia.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['periodos'] = PeriodoAcademico.objects.all()
        context['h'] = datetime.now()
        tipo = self.request.session['tipo']
        spk = self.request.session['spk']
        if tipo == 1:
            pk = kwargs['pk']
            actividad = Actividad.objects.get(pk=pk)
            context['actividad'] = actividad
            context['horarios'] = actividad.horario_set.all()
        return context
```

Ilustración 22. Vista del cronograma de la actividad

6. **CalificarList (LoginRequiredMixin, TemplateView):** Funciona Vista dedicada al docente, el nombre de la template HTML es "calificar_actividades.html" y añade al contexto la activad y el listado de notas de los inscritos filtrado por actividad.

```
class CalificarList(LoginRequiredMixin, TemplateView):
    template_name = 'calificar_actividades.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['periodos'] = PeriodoAcademico.objects.all()
        context['h'] = datetime.now()
        tipo = self.request.session['tipo']
        spk = self.request.session['spk']
        if tipo == 1:
            pk = kwargs['pk']
            actividad = Actividad.objects.get(pk=pk)
            context['actividad'] = actividad
            notas = NotaInscrito.objects.filter(inscrito__actividad=actividad)
            context['notas'] = notas
        return context
```

Ilustración 23. Vista para visualizar el listado de calificaciones de los inscritos

7. **AsistenciaNormalList (LoginRequiredMixin, TemplateView):** Contiene la funcionalidad básica para registrar la asistencia, es decir, a través de una tabla HTML se presentan los inscritos y procede a chequear a los asistentes. Como es del proceso de asistencia, se verifica que el rol sea docente.

```python
class AsistenciaNormalList(LoginRequiredMixin, TemplateView):
    template_name = 'registrar_asistencia.html'
    model = AsistenciaInscrito
    success_url = reverse_lazy('cronograma')

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['periodos'] = PeriodoAcademico.objects.all()
        context['h'] = datetime.now()
        tipo = self.request.session['tipo']
        spk = self.request.session['spk']
        if tipo == 1:
            pass
            pkactividad = kwargs['actividad']
            pkhorario = kwargs['horario']
            actividad = Actividad.objects.get(pk=pkactividad)
            horario = Horario.objects.get(pk=pkhorario)
            inscritos = AsistenciaInscrito.objects.filter(inscrito__actividad=actividad, horario=horario)
            context['actividad'] = actividad
            context['horario'] = horario
            context['inscritos'] = inscritos
        return context
```

Ilustración 24. Vista para registrar la asistencia normal

8. **AsistenciaQrList (LoginRequiredMixin, TemplateView):** A diferencia de la vista anterior, esta vista al presentar la template "registrar_qr.html" ya contará con la funcionalidad de registrar la asistencia a través de la lectura del código QR.

```python
class AsistenciaQrList(LoginRequiredMixin, TemplateView):
    template_name = 'registrar_qr.html'
    model = AsistenciaInscrito
    success_url = reverse_lazy('cronograma')

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['periodos'] = PeriodoAcademico.objects.all()
        context['h'] = datetime.now()
        tipo = self.request.session['tipo']
        spk = self.request.session['spk']
        if tipo == 1:
            pkactividad = kwargs['actividad']
            pkhorario = kwargs['horario']
            actividad = Actividad.objects.get(pk=pkactividad)
            horario = Horario.objects.get(pk=pkhorario)
            inscritos = AsistenciaInscrito.objects.filter(inscrito__actividad=actividad, horario=horario)
            context['actividad'] = actividad
            context['horario'] = horario
        return context
```

Ilustración 25. Vista para registrar la asistencia QR

9. **AsistenciaQr (LoginRequiredMixin, TemplateView):** El funcionamiento es similar a la anterior, con la única diferencia que el docente no necesitará entrar al módulo Actividades Complementarias para registrar la asistencia, para este caso debe entrar al enlace directo "Asistencia QR".

```python
class AsistenciaQr(LoginRequiredMixin, TemplateView):
    template_name = 'registrar_qr_menu.html'
    model = AsistenciaInscrito
    success_url = reverse_lazy('cronograma')

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        tipo = self.request.session['tipo']
        spk = self.request.session['spk']
        context['periodos'] = PeriodoAcademico.objects.all()
        context['h'] = datetime.now()
        if tipo != 1:
            self.template_name = 'menu.html'
        return context
```

Ilustración 26. Vista para registrar asistencia a través del enlace directo Asistencia QR

10. **AjaxRequestView (LoginRequiredMixin, View):** Vista definida para ser usada con peticiones Ajax desde el cliente (navegador), con la primera acción denominada "addasistencia" procede a registrar la asistencia de los alumnos de forma normal, recorrer un listado de valores seleccionados de la tabla HTML y procede a cambiar el estado de asistencia de los inscritos a True en caso de haber sido chequeado y False en caso contrario. La acción "addqrasistencia" toma el valor leído por el lector QR y lo descifra la información enviada. La función signing.loads permite descifrar una cadena de texto y validar que haya sido un objeto JSON.

```python
class AjaxRequestView(LoginRequiredMixin, View):
    def get(self, request, *args, **kwargs):...

    def post(self, request, *args, **kwargs):
        try:
            accion = kwargs['accion']
            if accion == 'addasistencia':
                cadenaselect = clear_str(request.POST['cadenaselect'], ',')
                cadenanoselect = clear_str(request.POST['cadenanoselect'], ',')
                horariopk = request.POST['fechaactividadesid']
                with transaction.atomic():
                    if Horario.objects.filter(pk=horariopk).exists():
                        horario = Horario.objects.filter(pk=horariopk)[0]
                        for select in cadenaselect:
                            if AsistenciaInscrito.objects.filter(pk=int(select), horario=horario).exists():
                                asistencia = AsistenciaInscrito.objects.filter(pk=int(select), horario=horario)[0]
                                asistencia.calificado = True
                                asistencia.asistio = True
                                asistencia.save()
                        for notselect in cadenanoselect:
                            if AsistenciaInscrito.objects.filter(pk=int(notselect), horario=horario).exists():
                                asistencia = AsistenciaInscrito.objects.filter(pk=int(notselect), horario=horario)[0]
                                asistencia.calificado = True
                                asistencia.asistio = False
                                asistencia.save()
                        return HttpResponse(json.dumps({'result': 'ok', 'mensaje': 'Asistencia registrada correctamente'}))
            elif accion == 'addqrasistencia':
                pk = signing.loads(request.POST['value'])
                with transaction.atomic():
                    tipo = self.request.session['tipo']
```

Ilustración 27. Vista para peticiones Ajax, con registro de asistencia normal

```
    elif accion == 'addqrasistencia':
        pk = signing.loads(request.POST['value'])
        with transaction.atomic():
            tipo = self.request.session['tipo']
            spk = self.request.session['spk']
            if tipo == 1:
                if AsistenciaInscrito.objects.filter(pk=pk['asis'], inscrito__actividad__id=pk['act'],
                                                     inscrito__matricula__estudiante__id=pk['est'],
                                                     inscrito__actividad__tutorPrincipal=spk).exists():
                    asistencia = AsistenciaInscrito.objects.filter(pk=pk['asis'],
                                                                   inscrito__actividad__id=pk['act'],
                                                                   inscrito__matricula__estudiante__id=pk[
                                                                       'est'],
                                                                   inscrito__actividad__tutorPrincipal=spk)[0]
                    asistencia.calificado = True
                    asistencia.asistio = True
                    asistencia.save()
                    return HttpResponse(
                        json.dumps({'result': 'ok', 'mensaje': 'Asistencia registrada correctamente'}))
                else:
                    return HttpResponse(
                        json.dumps({'result': 'error', 'mensaje': 'No tiene acceso a esta actividad'}))
            else:
                return HttpResponse(
                    json.dumps({'result': 'error', 'mensaje': 'No tiene acceso a esta función'}))
    except Exception as ex:
        return HttpResponse(json.dumps({'result': 'error',
                                        'mensaje': 'Ocurrió un error al registrar, código incorrecto. Vuelva a '
                                                   'escanear.'}))
```

Ilustración 28. Vista para peticiones Ajax, con registro de asistencia QR

Gestion_tag.py

Las templates de Django permiten mostrar información enviada por el contexto de la vista, por ejemplo, presentar un formulario de usuario, presentar un listado de alumnos. Pero las limitaciones de ejecución de código Python y Django han llevado a crear las template_tags que son funciones que permiten la ejecución del código mencionado en las mismas template de Django.

Estas template_tags pueden recibir parámetros, retornar listas y diccionarios de una forma sutil y elegante. Basta con crear un directorio denominado template_tags en la raíz de nuestra aplicación y crear un archivo como "gestion_tags.py" para definir dentro de él las funciones a usar en nuestras templates, a continuación las funciones utilizadas en el proyecto:

- **get_username (tipo, spk):** Recibe por parámetro el rol del usuario y su clave primaria para retornar el nombre de usuario, se realizó esta función por la limitación de aplicar filtros de modelos en la template normal.
- **get_total_inscritos (actividad):** Enviando la actividad por parámetro retorna el número total de inscritos.
- **get_horarios (actividad):** Filtra los horarios por actividad.
- **get_horario_inscrito (actividad, spk):** Obtiene el horario de la actividad de un alumno inscrito.
- **get_nota_actividad (actividad, spk):** Obtiene la nota de la actividad del alumno inscrito.

- **get_asistencia (actividad, horario, spk):** Retorna el estado de asistencia a la actividad del alumno inscrito.
- **get_value_code (actividad, horario, spk):** Cifra los datos de la actividad, el horario y el alumno inscrito, retorna un diccionario JSON que sirve como pilar para la creación del código QR.
- **get_total_fechas (actividad):** Retorna un número de fechas de la actividad.
- **get_total_asistencias (inscrito):** Retorna el total de asistencias a los horarios de un alumno inscrito.

Urls.py

Son el intermediario entre las vistas y los templates. Por cada URL definida dentro de un archivo urls.py se debe asociar una vista, es decir, cada que el usuario se dirija a una dirección, el primer archivo en ser ejecutado será el urls.py, Django buscará la URL con mayor coincidencia y le asignará a la vista (controlador) tu petición HTTP. El nombre es usado para funciones de reverso, cuando se haga una petición por nombre de URL, este buscará la dirección asociada. Dentro del proyecto se usaron dos archivos URLs, el primero es el ROOT_URLCONF y el segundo está segmentado para mantener una estructura limpia de la aplicación.

- **actividades\urls.py:** Archivo base de URLs del proyecto, contiene direcciones importantes como el inicio y cierre de sesión, administrador, archivos estáticos y multimedia.

```python
from django.conf.urls import url
from django.contrib import admin
from django.urls import path, include
from django.views.static import serve
from actividades.settings import STATIC_ROOT, MEDIA_ROOT
from gestion.views import HomeView, IniciarSesionView, CerrarSesionView, AjaxRequestView, AsistenciaQr
urlpatterns = [
    path('', HomeView.as_view(), name='home'),
    path('admin/', admin.site.urls, name='admin'),
    path('login/', IniciarSesionView.as_view(), name='login'),
    path('logout/', CerrarSesionView.as_view(), name='logout_'),
    path('ajaxrequest/<slug:accion>', AjaxRequestView.as_view(), name='ajaxrequest'),
    path('aextracurriculares/', include('gestion.urls'), name='gestion'),
    path('asistencia_qr/', AsistenciaQr.as_view(), name='asistencia'),
    url(r'^static/(?P<path>.*)$', serve, {'document_root': STATIC_ROOT}),
    url(r'^media/(?P<path>.*)$', serve, {'document_root': MEDIA_ROOT})
]
```

Ilustración 29. URLs del proyecto.

- **gestión\urls.py:** Es el segmento de URLs que contiene las vistas de la aplicación gestión, para acceder a este archivo primero la aplicación debe pasar por el anterior, y tendrá como prefijo la URL "/gestion".

```python
from django.urls import path
from gestion.views import *

urlpatterns = [
    path('', ActividadList.as_view(), name='actividades'),
    path('cronograma/<int:pk>', CronogramaList.as_view(), name='cronograma'),
    path('calificar/<int:pk>', CalificarList.as_view(), name='calificar'),
    path('cronograma/<int:actividad>/asistencia/<int:horario>', AsistenciaNormalList.as_view(),
         name='asistencia'),
    path('cronograma/<int:actividad>/asistenciaqr/<int:horario>', AsistenciaQrList.as_view(), name='asistenciaqr'),
]
```

Ilustración 30. URLs de la aplicación gestión

Templates

Archivos HTML que contienen el diseño que interactuará con el usuario (funcionalidad JavaScript, estilos CSS, estructura HTML). A continuación cada una de ellas a un nivel de detalle más profundo:

Login.html

Este documento fue ofrecido por el personal de TICS de la Universidad Estatal de Milagro, no se procederá a explicar para mantener la confidencialidad o a menos que sean adaptados nuevos cambios por el proyecto.

```python
<link href="/static/css/jquery.dataTables.css?1.0.0" type="text/css" rel="stylesheet"/>

<!--[if lt IE 9]>
      <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
    <![endif]-->
<!-- Global site tag (gtag.js) - Google Analytics -->

<script async src="https://www.googletagmanager.com/gtag/js?id=UA-110802679-1"></script>
<script>
    window.dataLayer = window.dataLayer || [];

    function gtag() {
        dataLayer.push(arguments);
    }

    gtag('js', new Date());

    gtag('config', 'UA-110802679-1');
</script>

<link rel="shortcut icon" href="/static/images/aok/favicon.ico?1.5.0">
<script>
    var capippriva = '';
    window.RTCPeerConnection = window.RTCPeerConnection || window.mozRTCPeerConnection || window.webkitRTCPeerConnec
    var pc = new RTCPeerConnection({iceServers: []}), noop = function () {
    };
    pc.createDataChannel("");    //create a bogus data channel
    pc.createOffer(pc.setLocalDescription.bind(pc), noop);    // create offer and set local description
```

Ilustración 31. Parte del template de inicio de sesión con su JavaScript.

Menu.html

El menú también fue proporcionado por personal de TICS, la única diferencia es un icono con el nombre "Asistencia QR" como acceso directo al registro de la asistencia con el lector QR.

```
{% if request.session.tipo == 1 %}
    <div class='icon' url='asistencia_qr'>
        <div class='iconimage'>
            <div class='pd'>
                <img src='/static/images/iconos/qr-code.png' border="0"/>
            </div>
        </div>
        <div class='iconname'>
            <div class='pd'>
                <h4 class='tituloicon'>Asistencia QR</h4>
                <span class='icondesc'>Asistencia actividades QR</span>
            </div>
        </div>
    </div>
{% endif %}
```

Ilustración 32. Sección añadida como acceso directo para registrar la asistencia con código QR

List_alumno.html

Se maneja con el diseño ofrecido por el personal de TICS pero se adaptan cambios como un nuevo elemento para la visualización del código QR por cada horario de la actividad, un modal para presentación del código QR y el código JavaScript para descargar como archivo el código. La generación del QR es proporcionada por medio de JavaScript utilizando tecnología CANVAS esta permite por medio de datos generar los QR. Se procedió a enviar los datos del QR desde el servidor pero la generación se realiza desde el cliente para evitar sobrecarga en el proceso. De todas formas la información para la construcción del código está cifrada y si es manipulada, el descifrador notará el cambio y mostrará un mensaje de error.

```
<td data-title='ACCIÓN' style="...">
    <div class="accordion" id="accordion{{ a.pk }}{{ a.pk }}">
        <div class="accordion-group">
            <div class="accordion-heading">
            <span class="accordion-toggle">
                <a class="btn btn-mini btn-primary" data-toggle="collapse"
                   data-parent="#accordion655"
                   href="#collapseOne{{ a.pk }}{{ a.pk }}"><i
                        class="fa fa-qrcode"></i></a>
                <a data-toggle="collapse" data-parent="#accordion{{ a.pk }}{{ a.pk }}"
                   href="#collapseOne{{ a.pk }}{{ a.pk }}">Códigos de la actividad / Asistencia QR</a>
            </span>
            </div>
            <div id="collapseOne{{ a.pk }}{{ a.pk }}"
                 class="accordion-body collapse in">
                <div class="accordion-inner">
                    {% for horario in a|get_horario_inscrito:request.session.spk %}
                        <div style="...">
                            {% get_value_code a horario.horario request.session.spk as qr %}
                            <a rel="btnshowqr" class="btn btn-default btn-mini"
                               data-qr="{{ qr }}"
                               href="javascript:;">
                                <i class="fa fa-eye"></i>
                                Ver código QR
                                Fecha: {{ horario.horario.fecha|date:"Y-m-d" }}
                            </a><br>
                        </div>
                    {% endfor %}
```

Ilustración 33. Fragmento donde se muestra el código QR de la asistencia

```
$("#btndownloadqr").click(function () {
    let canvas = document.getElementById('qrcanvas');
    let url = canvas.toDataURL();
    let link = document.createElement('a');
    link.href = url;
    link.download = 'qractividad.png';
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
});
```

Ilustración 34. JavaScript con CANVAS para la descarga de los QR

List_profesor.html

Contiene la misma estructura que fue proporcionada por personal de TICS, e implementada en el proyecto para que el docente conozca su lista de actividades complementarias a cargo.

Ilustración 35. Fragmento de la template listar profesor

Registrar_asistencia.html

Creada con la finalidad de registrar la asistencia con el proceso normal si en algún caso el docente no quiera usar la nueva funcionalidad de la lectura de código QR. La estructura fue proporcionada por el personal de TICS. La sincronización de datos con el servidor es por medio de AJAX para tener una mejor calidad en la sincronización de la asistencia.

```
<div class='row-fluid'>
    <div class='span12'>
        <table class='table table-striped table-bordered'>
            <thead>
            <tr>
                <th style="...">Todos <br><input id="selectall" type="checkbox">
                </th>
                <th style="...">No</th>
                <th style="...">CEDULA</th>
                <th style="...">APELLIDOS Y NOMBRES</th>
                <th style="...">FACULTAD</th>
            </tr>
            </thead>
            <tbody>
            {% for inscrito in inscritos %}
                <tr>
                    <td style="...">
                        <input type="checkbox" {% if inscrito.asistio %}checked{% endif %} class="listadocheck"
                               id="lista{{ inscrito.pk }}" name="lista{{ inscrito.pk }}"
                               ind="{{ inscrito.pk }}"
                               value="{{ inscrito.pk }}">
                    </td>
                    <td style="...">{{ forloop.counter }}</td>
                    <td style="...">{{ inscrito.inscrito.matricula.estudiante.persona.documento }}</td>
                    <td style="...">{{ inscrito.inscrito.matricula.estudiante.persona.datos|upper }}</td>
                    <td style="...">
                        {{ inscrito.inscrito.actividad.facultad.nombre }}<br>
                    </td>
                </tr>
            {% endfor %}
            </tbody>
        </table>
    </div>
```

Ilustración 36. Generación de la lista de estudiantes en la actividad para el registro normal de asistencia

Calificar_actividades.html

Template realizada y generada a través de los archivos pasados para la realización de los mismos ya que no interactúa en gran parte con la generación de asistencia, pero de todas formas se realizó con el llamamiento a las vistas de django.

```
<tbody>
{% for nota in notas %}
    <tr>
        <td style="..."
            class="hidden-phone hidden-tablet">{{ forloop.counter }}</td>
        <td style="..." class="hidden-phone hidden-tablet">
            {{ nota.inscrito.matricula.estudiante.persona.documento }}
        </td>
        <td style="...">
            {{ nota.inscrito.matricula.estudiante.persona.datos|upper }}
        </td>
        <td style="..." class="hidden-phone hidden-tablet">
            {{ nota.inscrito.actividad.facultad.nombre }}
        </td>
        <td style="...">
            {% get_total_asistencias nota.inscrito as asistencias %}
            <input type="hidden" name="totasial{{ nota.pk }}" id="totasial{{ nota.pk }}"
                   value="{{ asistencias.1 }}">
            <span id="porcent{{ nota.pk }}"></span>
        </td>
        <td style="...">
            <input type="text" style="..." class="form-imput input-mini"
                   id="nota{{ nota.pk }}"
                   name="nota24734" placeholder="{{ actividad.calificacionSobre }}"
                   value="{{ nota.nota }}">
        </td>
        <td style="..." id="itemsbody{{ nota.pk }}">

        </td>
    </tr>
{% endfor %}
```

Ilustración 37. Fragmento de template usada para calificar actividades

Cronograma_asistencia.html

Creado con la finalidad para que el docente pueda seleccionar o elegir qué tipo de asistencia que se desea (Asistencia Normal / Asistencia QR). El proyecto fue hecho contemplando los posibles casos de uso que tendría un docente al registrar la asistencia.

```
</td>
<td data-title='ACCIONES' style="...">
    <div class="accordion" id="accordion{{ horario.pk }}">
        <div class="accordion-group">
            <div class="accordion-heading">
            <span class="accordion-toggle">
                <a class="btn btn-mini btn-success" data-toggle="collapse"
                  data-parent="#accordion655" href="#collapseOne{{ horario.pk }}"><i
                       class="fa fa-users"></i></a>
                <a data-toggle="collapse" data-parent="#accordion{{ horario.pk }}"
                  href="#collapseOne{{ horario.pk }}">Asistencia / Asistencia QR</a>
            </span>
            </div>
            <div id="collapseOne{{ horario.pk }}" class="accordion-body collapse in">
                <div class="accordion-inner">
                    <div style="margin-top: 10px">
                        <a style="text-align: center"
                          href="{{ actividad.pk }}/asistencia/{{ horario.pk }}"
                          class="btn btn-default btn-mini bloqueo_pantalla">
                           <i class="fa fa-plus"></i>
                           Asistencia Normal
                        </a>
                    </div>
                    <div style="...">
                        <a style="..."
                          href="{{ actividad.pk }}/asistenciaqr/{{ horario.pk }}"
                          class="btn btn-default bloqueo_pantalla">
                           <i class="fa fa-qrcode"></i>
                           Asistencia QR
                        </a>
```

Ilustración 38. Template que muestra la sección de elección en asistencias

Registrar_qr.html y Registrar_qr_menu.html

Templates que contienen la misma funcionalidad pero son llamadas desde vistas diferentes. Mientras que a registrar_qr.html la llama la vista que accede directo al lector de códigos, a registrar_qr_menu.html la llama la vista que accede mediante el listado de actividades del docente y luego al cronograma de la actividad. Ambas tienen la interfaz de manipulación de la cámara para la lectura del código QR, la sincronización de la asistencia se realiza a través de Ajax para optimizar el tiempo de envío y evitar la recarga de la página.

```
</tr>
<tr>
    <th style="...">
        <button title="Iniciar" class="btn btn-success btn-sm" id="play" type="button"
                data-toggle="tooltip">
            <span class="fa fa-play"></span> Iniciar Cámara
        </button>
        <button title="Pausar" class="btn btn-warning btn-sm" id="pause" type="button"
                data-toggle="tooltip">
            <span class="fa fa-pause"></span> Pausar
        </button>
        <button title="Detener" class="btn btn-danger btn-sm" id="stop" type="button"
                data-toggle="tooltip"><span class="fa fa-stop"></span> Detener
        </button>
    </th>
</tr>
<tr>
    <th style="...">Seleccione cámara</th>
</tr>
<tr>
    <th style="...">
        <select style="..." class="form-control" id="camera-select"></select>
    </th>
</tr>
<tr>
    <th style="...">Escáner</th>
</tr>
<tr>
    <th style="...">
```

Ilustración 39. Fragmento de diseño de los controles de la cámara

```
$("#aceptsendqr").click(function () {
    bloqueointerface();
    let value = $("#hideqrresult").val();
    $.post("/ajaxrequest/addqrasistencia", {
        "value": value
    }, function (data) {
        $.unblockUI();
        if (data.result == 'ok') {
            $("#sendqr").modal({backdrop: 'static', width: '450px'}).modal('hide');
            smoke.alert(data.mensaje);
        } else {
            smoke.alert(data.mensaje);
        }
        $("#sendqr").modal("hide");
    }, "json");
    return false;
});
```

Ilustración 40. Petición Ajax para la sincronización de la asistencia

CAPÍTULO 4

METODOLOGÍA Y EJECUCIÓN DEL PROYECTO TÉCNICO

En la etapa de levantamiento de información se procedió a comunicar al tutor la necesidad del esquema de las tablas relacionadas a las actividades complementarias, una explicación del proceso con el rol de docente y parte del diseño HTML, CSS, JavaScript del Sistema de Gestión Académica.

En compañía del tutor se solicitaron los permisos para fines educativos de los requerimientos mencionados anteriormente. Luego de esto nos facilitaron archivos de texto que contenían el diseño de los siguientes formularios y listas:

- Calificar actividades
- Cronograma de asistencia
- Registrar asistencia

El esquema de tablas relacionadas con las actividades complementarias fue facilitado en un documento por la ingeniera Kerly Palacios, personal del área de desarrollo de TICS.

Para efectos de recrear el SGA y tener una mejor experiencia de usuario, el diseño de inicio de sesión y el menú principal fueron extraídos del navegador.

Ilustración 41. Diseño de inicio de sesión recreado

Luego de recrear el inicio de sesión también se procedió a recrear el menú del SGA pero solo se tendrá en cuenta la utilización del módulo de actividades complementarias, estas fueron rediseñadas gracias a la colaboración de TICS

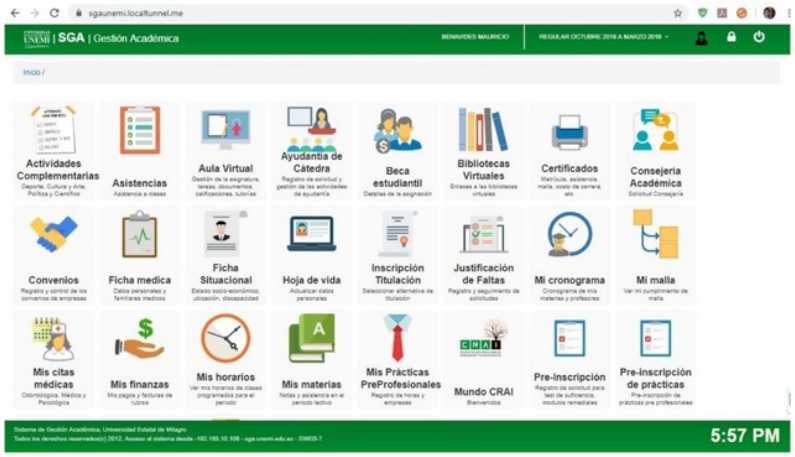

Ilustración 42. Menú de estudiante recreado

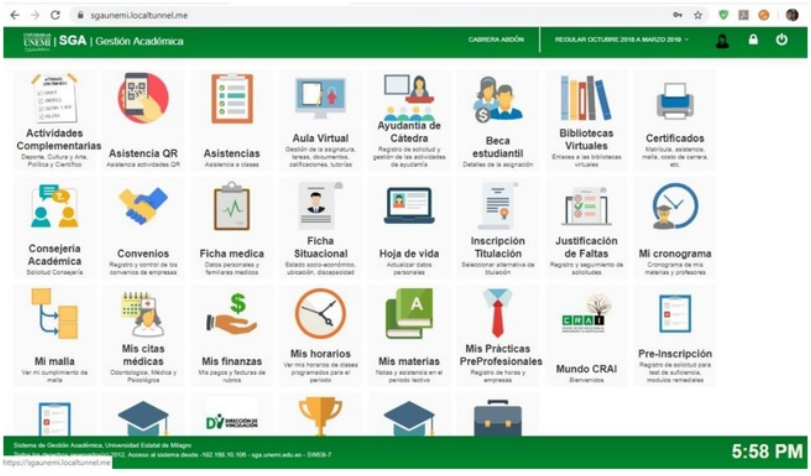

Ilustración 43. Menú de docente recreado, junto con el nuevo ícono de Asistencia QR

Luego de recrear el diseño de inicio de sesión y menú, se procedió a rediseñar la base de datos añadiendo los nuevos campos para funcionamiento del proyecto.

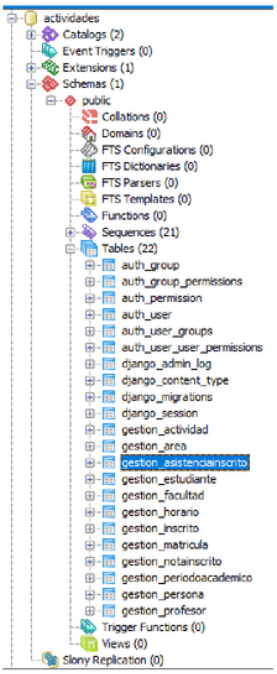

Ilustración 44. Diseño de las tablas que se usarán dentro de la base de datos.

Terminado la generación de la base de datos como esquema principal del proyecto, paralelamente se iba pensando cual sería la técnica de programación que se usaría para la generación de los códigos QR ya que existen múltiples técnicas de las cuales se hicieron pruebas y se concluyó con la utilización de CANVAS para evitar la generación de imágenes lo cual ocasionaría una sobrecarga en el servidor, al usar la tecnología CANVAS la generación del código se realiza al instante y la exigencia de procesamiento es mínima.

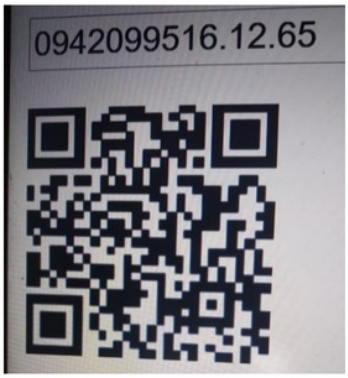

Ilustración 45. Generación de código QR por medio de CANVAS usando JavaScript en página de pruebas.

Luego se procedió con la generación de código QR usando datos del estudiante, actividad entre otras, esto generaría un código en el momento con CANVAS.

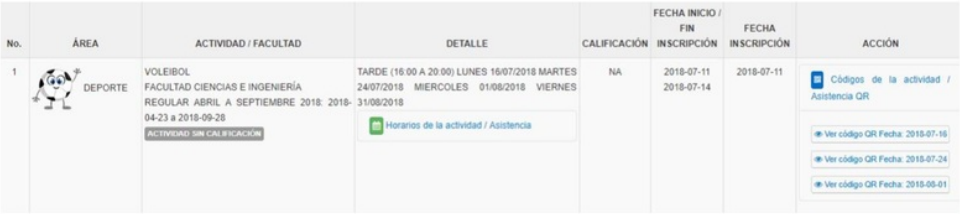

Ilustración 46. Modificación del menú para que el alumno pueda visualizar el código QR generado.

Ilustración 47. Código QR generado en el momento según la fecha que se haya solicitado y con posibilidad de descarga

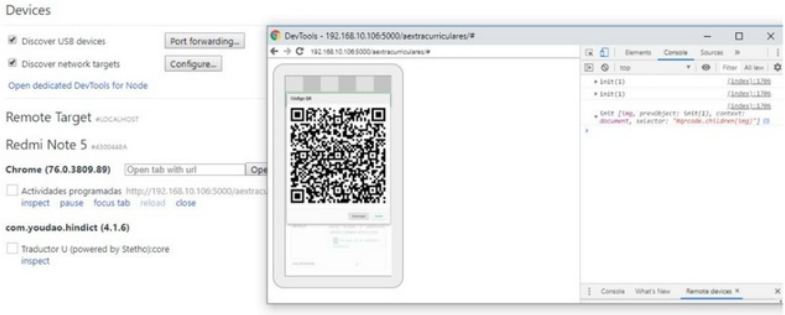

Ilustración 48. Fase de pruebas de generación de códigos QR desde dispositivos, usando el debugger de Chrome en Android.

Luego se modificó el diseño del listado de actividades por docente para agregar las acciones de registro de asistencia normal o asistencia QR. Teniendo en cuenta que a pesar de tener una nueva funcionalidad el docente puede necesitar de la funcionalidad básica para efectos de comodidad.

Ilustración 49. Listado de actividades del docente.

Ilustración 50. Acciones para toma de asistencia normal o asistencia QR.

Una vez detallado como el docente podrá entrar a la toma de asistencia de sus actividades, se muestra el diseño para la asistencia básica y más abajo el diseño del lector de códigos QR, esta sección tiene apartados que le permitirá seleccionar la cámara, iniciar, pausar, detener el proceso de lectura y parametrizar configuraciones como el zoom, contraste, brillo y orientación.

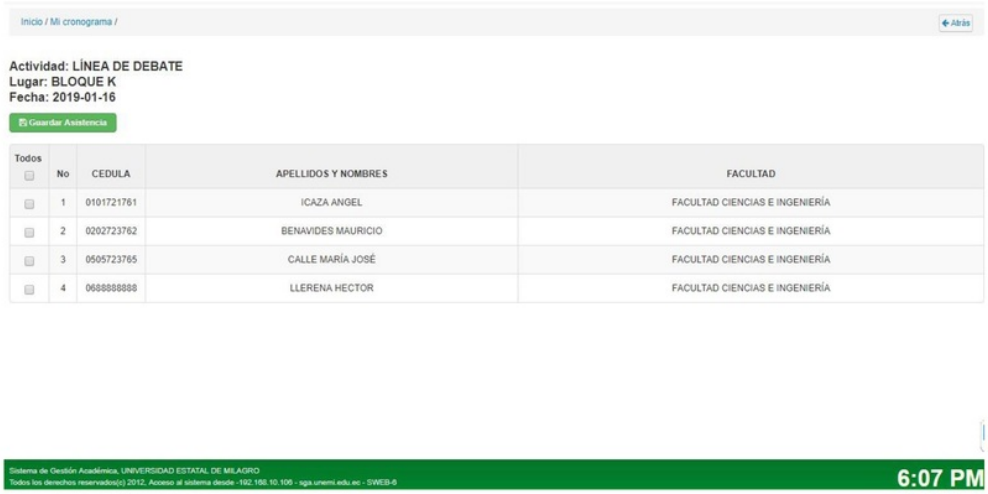

Ilustración 51. Página de asistencia normal de alumnos

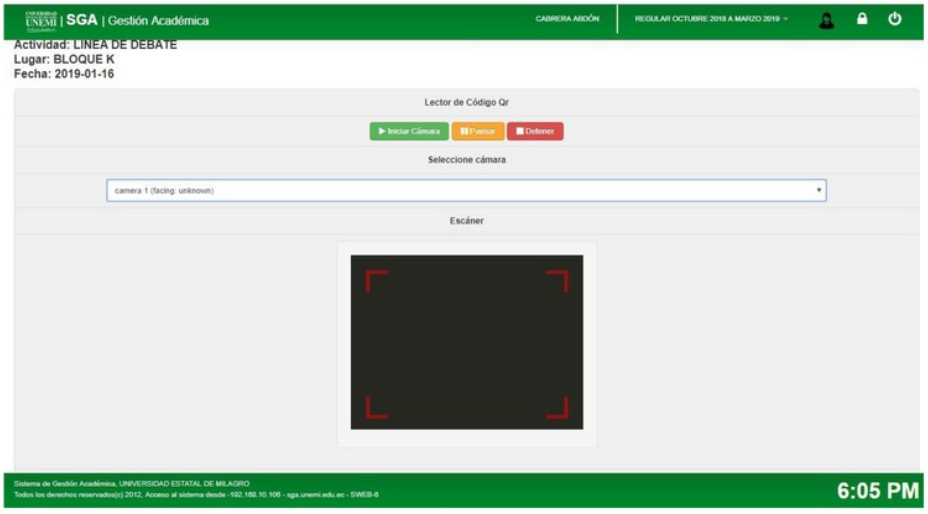

Ilustración 52. Página de registro de asistencia QR usando cámara del dispositivo (Smartphone, Tablet)

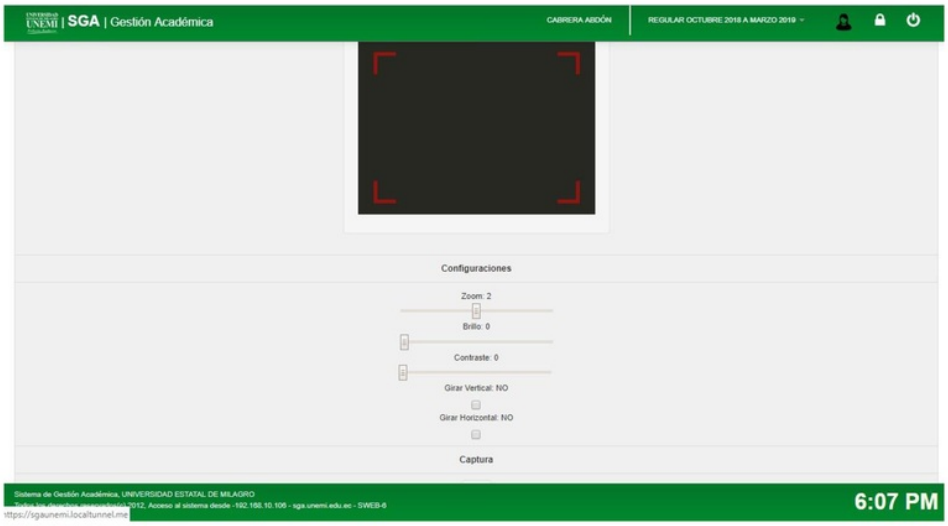

Ilustración 53. Menú de configuración de cámara para comodidad en la lectura.

CAPÍTULO 5

EVALUACIÓN DEL PROYECTO TÉCNICO

Para efectos de pruebas se realizaron capturas del módulo de Actividades Complementarias mientras se realiza el escaneo del código QR.

Se procedió a hacer pruebas con un dispositivo móvil debido a que es la forma más óptima de usar la aplicación. Aunque la lectura podría realizarse incluso desde una computadora, pero por cuestiones de presentación, el Smartphone fue la mejor opción.

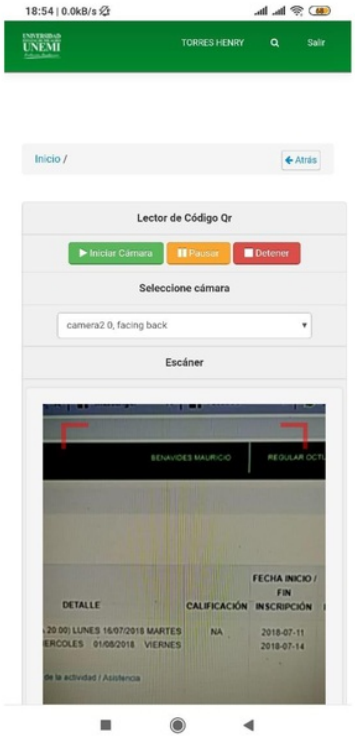

Ilustración 54. Uso de cámara móvil para la lectura del código QR.

Una vez activada la cámara el docente debe proceder a leer el código en cualquier posición ya que los QR contiene posicionamiento dentro del mismo así que sin importar de qué manera se agregué este lo leerá.

En algunos casos, al acceder por primera vez a la aplicación desde un dispositivo nuevo, al iniciar la cámara, se muestra la cámara frontal o la orientación está invertida, para esos

casos excepcionales se desarrollaron las opciones de elegir cámara y cambiar parámetros como la orientación, zoom, brillo, entre otros.

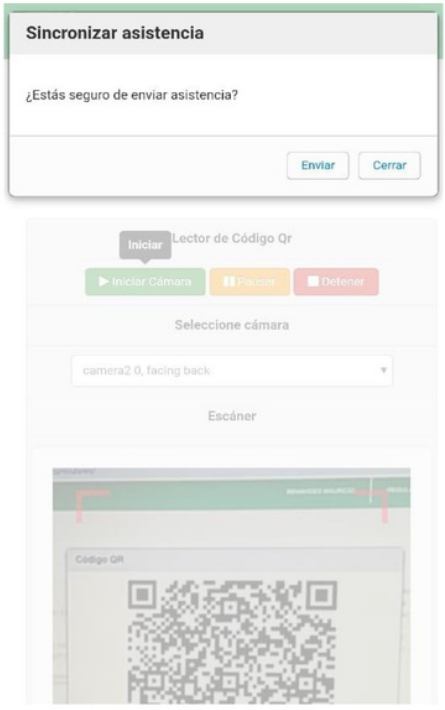

Ilustración 55. Código QR del alumno leído con confirmación de envío

Ilustración 56. Resultado luego de sincronizar la asistencia

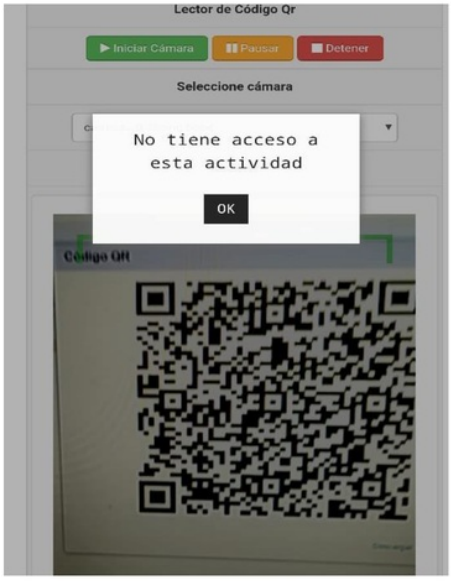

Ilustración 57. Mensaje de error cuando un docente o alumno que no tiene permiso en la actividad intenta sincronizar la asistencia

Ilustración 58. Mensaje de error cuando intentan modificar el código QR o ha perdido demasiada legibilidad

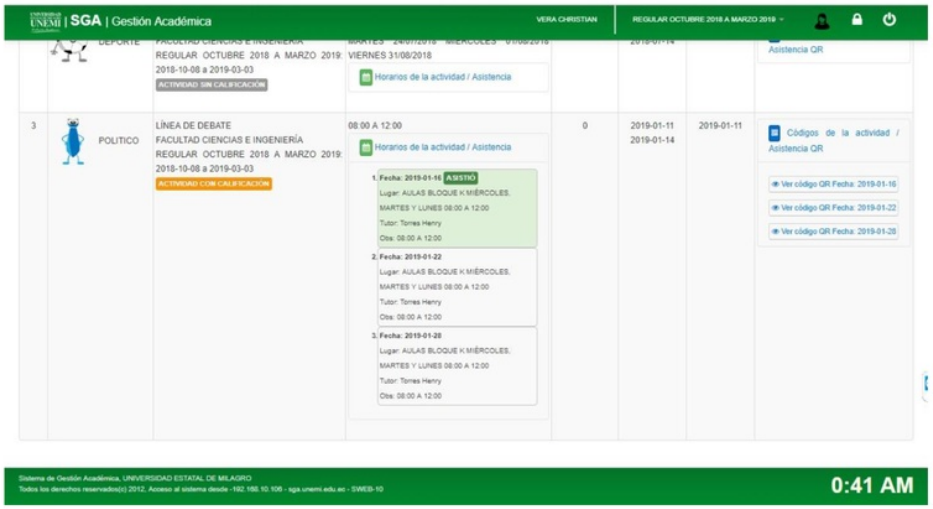

Ilustración 59. Menú de actividades de alumno antes de ser aplicada la asistencia

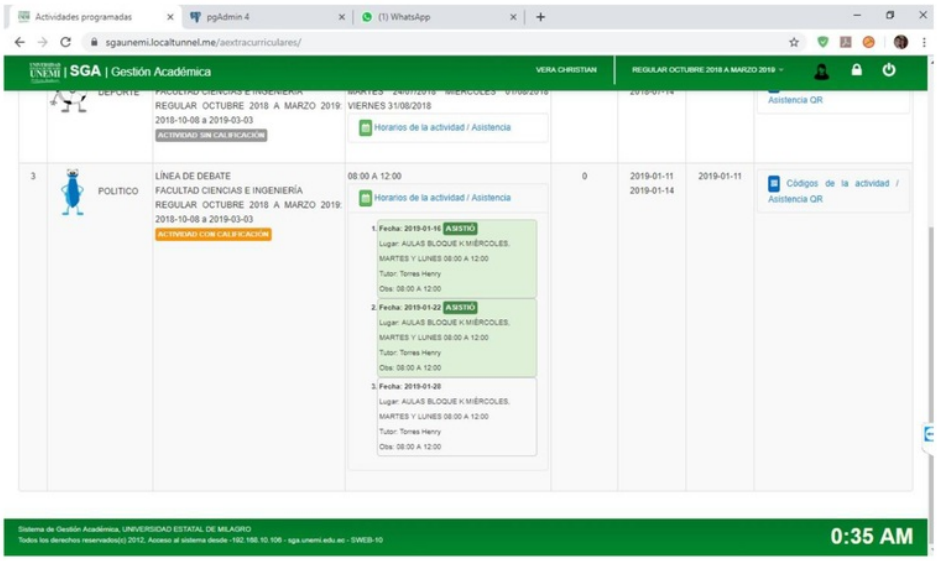

Ilustración 60. Menú de actividades complementarias con asistencia registrada

CONCLUSIONES

- El desarrollo de implementación de nuevas metodologías que ayudan a la optimización es importante en una sociedad actual que avanza a pasos agigantados, los códigos QR permiten solucionar múltiples problemas no solo en la generación de etiquetas para supermercados o compras de boletas en terminales de transporte sino en todos los aspectos cotidianos.
- Herramientas como Django, permiten al desarrollador desenvolverse de manera rápida, porque ayuda en gran medida a unir cada uno de los componentes de la aplicación como si se tratase de un rompecabezas.
- Aplicando este enfoque a la cultura universitaria permite la reducción de usos de medios manuales como listas impresas por parte de los docentes, y en el caso de los alumnos, podrán presentarse a sus actividades con el código descargado en la galería de su Smartphone o la de su amigo, evitando así el aglutinamiento que provoca pérdida de tiempo.
- Se pudo haber generado una imagen del código QR por cada horario de la actividad y luego ser guardada en el servidor, pero para optimizar el rendimiento cuando el número de peticiones es masiva se eligió la tecnología CANVAS de HTML5 para generar los códigos QR, cabe aclarar que el proceso de cifrar y descifrar la información se realiza en el servidor.
- El uso de métodos de comunicación asíncrona como Ajax con jQuery permitió que el sistema actúe en tiempo real en la sincronización de la asistencia permitiendo optimizar tiempo en la lectura masiva de códigos.

RECOMENDACIONES

- Aplicar el uso de códigos QR y otras tecnologías de respuesta rápida en más áreas de la universidad con el fin de agilizar los procesos y reducir el impacto ambiental como academia comprometida con la sociedad.
- Adoptar buenas prácticas de desarrollo y usabilidad en el sistema de gestión académica las cuales permitan la adaptación del diseño en pantallas medianas y pequeñas.
- Invitar a la comunidad estudiantil a dar buen uso a las tecnologías móviles en múltiples ámbitos de la vida y seguir construyendo ideas innovadoras, permitiendo así dejar de buscar empleo y convertirse en creadores de nuevas fuentes.

ANEXOS

REPÚBLICA DEL ECUADOR

UNIVERSIDAD ESTATAL DE MILAGRO

UNEMI

LISTA DE ALUMNOS INSCRITOS

1. ÁREA/ACTIVIDAD

ÁREA	CULTURA Y ARTE
ACTIVIDAD	DANZA
FACULTAD	FACULTAD CIENCIAS DE LA INGENIERIA
TUTOR	CORDOVA MORAN JORGE
LUGAR	POLIDEPORTIVO
OBSERVACIÓN	15:00-19:00
FECHA ACTIVIDAD	2019-01-28

2. INSCRITOS

JORNADA: MATUTINA

N.	CEDULA	APELLIDOS Y NOMBRES	CARRERA	FIRMA
1	0940329261	CARDENAS MORALES SEGUNDO ALBERTO	INGENIERÍA EN SISTEMAS COMPUTACIONALES	

JORNADA: NOCTURNO

N.	CEDULA	APELLIDOS Y NOMBRES	CARRERA	FIRMA
1	0951381706	ALVAREZ VILLAMAR KEVIN SAUL	INGENIERÍA INDUSTRIAL	
2	0606240711	ARGUELLO LOOR BELLA CORALIA	INGENIERÍA INDUSTRIAL	
3	0942124363	BARBOZA OCAÑA MELANIA MICHELLE	INGENIERÍA INDUSTRIAL	
4	0941998254	BRAVO POZO TABATA BETZABETH	INGENIERÍA INDUSTRIAL	
5	0952901932	CORDERO AYAVACA ELIZABETH CRISTINA	ALIMENTOS	
6	0944405174	CRUZ ROBELO IVANNA RAQUEL	INGENIERÍA INDUSTRIAL	
7	2400134892	FRANCO TOMALA CHRISTIAN FERNANDO	INGENIERÍA EN SISTEMAS COMPUTACIONALES	
8	0954681300	GUTIERREZ SAMANIEGO ISAAC ALFONSO	INGENIERÍA INDUSTRIAL	
9	0941987844	JIBAJA LOY JOSUE ALFREDO	INGENIERÍA INDUSTRIAL	
10	0958328734	SOLIS SAICO RAUL FRANCISCO	INGENIERÍA INDUSTRIAL	
11	0942484338	VIEJO MATA BRYAN ELIAS	INGENIERÍA EN SISTEMAS COMPUTACIONALES	

Ilustración 61. Levantamiento de información, lista impresa para el registro de asistencia

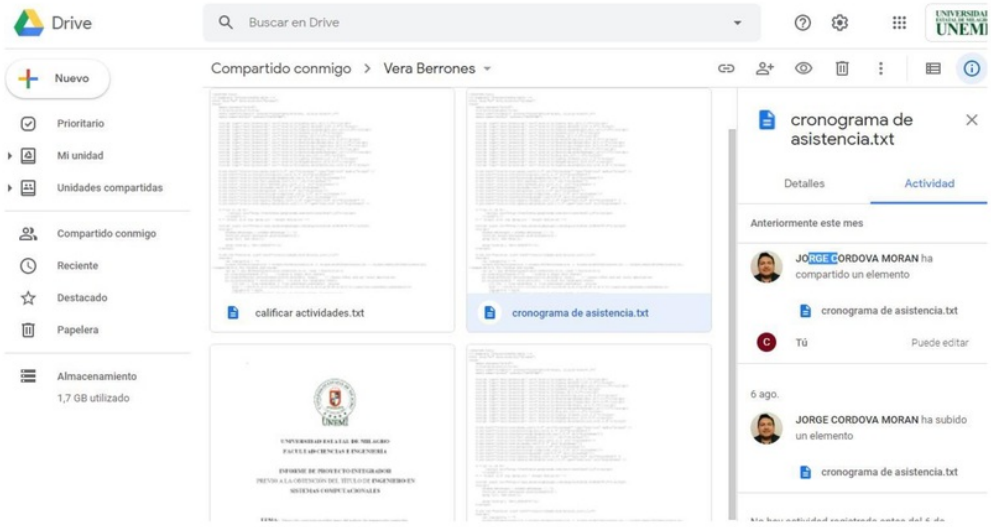

Ilustración 62. Archivos proporcionados por TICS

Ilustración 63. Menú vista desde una interfaz móvil.

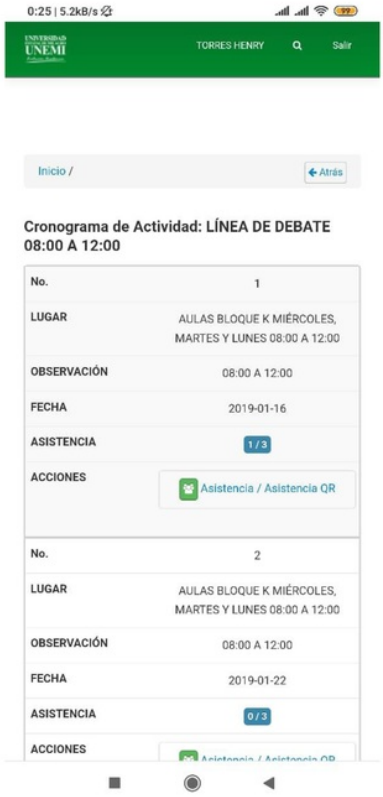

Ilustración 64. Cronograma de actividad docente visto desde una interfaz móvil.

REFERENCIAS BIBLIOGRÁFICAS

Anibarro Z, C. E. (2001). Manual básico de HTML.

Cassanova, G., & Molina, J. (2013). Implementación de códigos QR en materiales docentes, (c), 2–6.

Django Software Foundation. (2019). *Django Documentation: Why Django?* Retrieved from https://www.djangoproject.com/start/overview/

García, J., & Okazaki, S. (2012). El uso de los códigos QR en España. *Distribución y Consumo*, (Mayo-Junio), 46–62. Retrieved from http://dialnet.unirioja.es/servlet/articulo?codigo=3972254&orden=1&info=link

Gonzalez, J., & Garcia, A. (2016). Códigos QR y sus aplicaciones en las ciencias de la salud, *27*(2), 239–248.

Graván, P., & Gutiérrez, Á. (2013). La formación de docentes en estrategias innovadoras de enseñanza y aprendizaje: los códigos de respuesta rápida o códigos QR Teacher training in innovative teaching strategies and learning: quick response codes or QR codes, (Dim), 1–14.

Holovaty, A., & Kaplan, J. (2008). El libro de Django.

Holovaty, A., & Kaplan Mosss, J. (2015). La guía definitiva de Django: Desarrolla aplicaciones web de forma rápida y sencilla, 598. Retrieved from http://github.com/saulgm/djangobook.com

Huidobro, J. M. (2009). Código QR. *Bit*, *172*, 47–49. Retrieved from http://cmapspublic2.ihmc.us/rid=1NS6XZ211-1V8WNZ2-2555/Microcodigos qr.pdf

Izquierdo, A. (2009). CÓDIGOS QR FLEXIBLES: UN PROYECTO CON DISPOSITIVOS MÓVILES PARA EL TRABAJO DE CALENTAMIENTO EN EDUCACIÓN FÍSICA, *23*, 53–71.

Marzal Varó, A., & Gracia Luengo, I. (2014). *Introducción a la programación con Python*. https://doi.org/10.6035/sapientia93

Villarrea, O., & Villamizar, R. (2013). Incrustación de imágenes en códigos de barras bidimensionales de rápida respuesta QR-codes, 277–288.

QR

INFORME DE ORIGINALIDAD

0%	0%	0%	0%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

Excluir citas	Activo	Excluir coincidencias	< 30 words
Excluir bibliografía	Activo		