Informe de Laboratorio 06

Tema: Aplicaicon del Django a nuestra base de datos

Nota

Estudiante	Escuela	Asignatura
Arce Mayhua Leonardo Velasque Arcos Mikhail Quispe Saavedra Dennis	Escuela Profesional de Ingeniería de Sistemas	Programación web 2 Semestre: I Código:

Laboratorio	Tema	Duración
06	Aplicaicon del Django a nuestra base de datos	04 horas

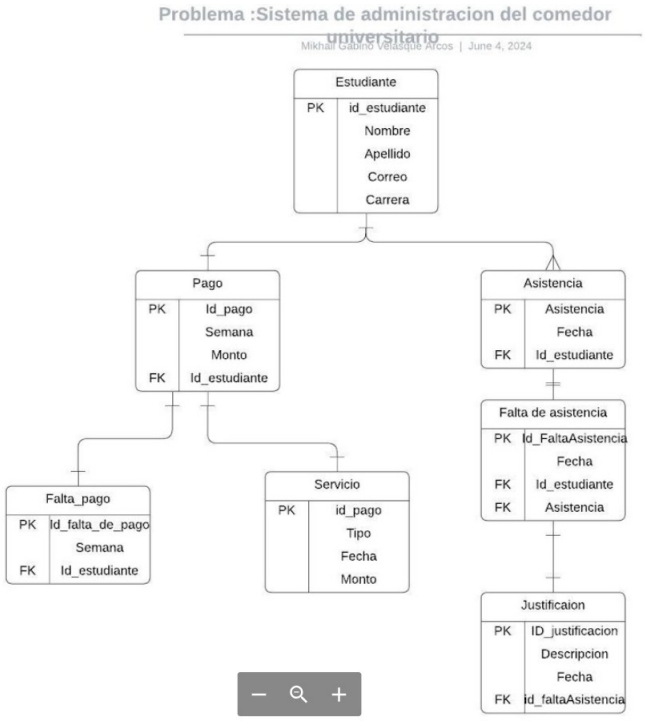
Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 3 de junio del 2024	Al 9 de Junio del 2024

1. INTRODUCCION

- En el siguiente informe se expondrán las actividades del laboratorio 06 del curso de Programación Web 02.
- La presentación de los ejercicios será de la siguiente manera:
- Primero: Se expondrán las partes más importantes del código. En este caso, se detallará primero nuestra problemática encontrada en la mejora del sistema del comedor universitario de la UNSA.
- Segundo: También se expondrá lo que se hizo dentro de Django y cómo plasmamos nuestro primer borrador de un diagrama ER para esta base de datos.

2. Problemática a tratar y el diagrama ER (primer borrador)

- En este caso, evaluamos el sistema del comedor universitario y cómo este tiene algunas deficiencias para con el comensal, empezando con la falta de un calendario que muestre la cantidad de días que el comensal no asistió al comedor, así como las fechas en que no realizó los pagos correspondientes para una semana determinada.
- También se observa la falta de una pestaña o sistema de justificación donde el comensal pueda acceder fácilmente para justificar sus faltas/inasistencias y la falta de pagos realizados.



3. Clase Models

En esta sección, definimos los modelos que representan las entidades del sistema del comedor universitario. Cada uno de estos modelos tiene atributos únicos y específicos, así como relaciones bien definidas entre ellos. Estos atributos permiten una gestión detallada y precisa de la información relacionada con el sistema del comedor universitario.

3.1. Modelo Asistencia

El modelo `Asistencia` representa la asistencia de los estudiantes al comedor universitario. Cada registro de asistencia está relacionado con un estudiante y contiene información como el estado de asistencia (registrado a través de `status`), la fecha de creación y modificación del registro (`created` y `modified`, respectivamente).

Listing 1: Asistencia

```
from django.db import models
from .Estudiante import Estudiante

class Asistencia (models.Model):
    Estudiante = models.ForeignKey(Estudiante, on_delete=models.CASCADE)
    id_Asistencia = models.BigAutoField(primary_key=True)
    status = models.BooleanField(default=False)
    created = models.DateTimeField(editable=False, null=False, auto_now_add=True)
    modified = models.DateTimeField(null=False, auto_now=True)
```

3.2. Modelo Estudiante

El modelo `Estudiante` representa a los estudiantes que utilizan el comedor universitario. Contiene información básica sobre los estudiantes, como su nombre, apellido, carrera y correo electrónico. Además, se registra el estado del estudiante (`status`) y las fechas de creación y modificación del registro.

Listing 2: Estudiante

```
from django.db import models

class Estudiante(models.Model):
    id_Estudiante = models.BigAutoField(primary_key=True)
    nombre = models.CharField(max_length=200)
    apellido = models.CharField(max_length=200)
    carrera = models.CharField(max_length=100)
    correo = models.CharField(max_length=100)
    status = models.BooleanField(default=False)
    created = models.DateTimeField(editable=False, null=False, auto_now_add=True)
    modified = models.DateTimeField(null=False, auto_now=True)
```

3.3. Modelo FaltaAsistencia

El modelo `FaltaAsistencia` registra las faltas de asistencia de los estudiantes al comedor universitario. Cada registro de falta de asistencia está relacionado con un registro de asistencia y contiene información sobre la fecha de la falta (`fecha`), el estado de la falta (`status`) y las fechas de creación y modificación del registro.

Listing 3: FaltaAsistencia

```
from django.db import models
from .Asistencia import Asistencia

class FaltaAsistencia (models.Model):
    Asistencia = models.ForeignKey(Asistencia, on_delete=models.CASCADE)
    id_FaltaAsistencia = models.BigAutoField(primary_key=True)
    fecha = models.DateTimeField(editable=False, null=False, auto_now_add=True)
    status = models.BooleanField(default=False)
    created = models.DateTimeField(editable=False, null=False, auto_now_add=True)
    modified = models.DateTimeField(null=False, auto_now=True)
```

3.4. Modelo FaltaPago

El modelo `FaltaPago` registra las faltas de pago de los estudiantes al comedor universitario. Cada registro de falta de pago está relacionado con un registro de pago y contiene información sobre la semana correspondiente al pago (`semana`), el estado de la falta (`status`) y las fechas de creación y modificación del registro.

Listing 4: FaltaPago

```
from django.db import models
from .Pago import Pago

class FaltaPago(models.Model):
    pago = models.ForeignKey(Pago, on_delete=models.CASCADE)
```

```
    semana = models.IntegerField(default=1)
    status = models.BooleanField(default=False)
    created = models.DateTimeField(editable=False, null=False, auto_now_add=True)
    modified = models.DateTimeField(null=False, auto_now=True)
    id_falta_pago = models.BigAutoField(primary_key=True)
```

3.5. Modelo Justificacion

El modelo `Justificacion` registra las justificaciones de las faltas de asistencia de los estudiantes. Cada justificación está relacionada con un registro de falta de asistencia y contiene una descripción de la justificación, la fecha de la justificación (`fecha`), el estado de la justificación (`status`) y las fechas de creación y modificación del registro.

Listing 5: Justificacion

```
from django.db import models
from .FaltaAsistencia import FaltaAsistencia

class Justificacion(models.Model):
    FaltaAsistencia = models.ForeignKey(FaltaAsistencia, on_delete=models.CASCADE)
    id_justificacion = models.BigAutoField(primary_key=True)
    descripcion = models.TextField()
    fecha = models.DateTimeField(editable=False, null=False, auto_now_add=True)
    status = models.BooleanField(default=False)
    created = models.DateTimeField(editable=False, null=False, auto_now_add=True)
    modified = models.DateTimeField(null=False, auto_now=True)
```

3.6. Modelo Pago

El modelo `Pago` registra los pagos realizados por los estudiantes en el comedor universitario. Cada registro de pago está relacionado con un estudiante y contiene información sobre la semana correspondiente al pago (`semana`), el monto del pago (`monto`), el estado del pago (`status`) y las fechas de creación y modificación del registro.

Listing 6: Pago

```
from django.db import models
from .Estudiante import Estudiante

class Pago(models.Model):
    Estudiante = models.ForeignKey(Estudiante, on_delete=models.CASCADE)
    id_pago = models.BigAutoField(primary_key=True)
    semana = models.IntegerField(default=1)
    monto = models.FloatField(default=0.0)
    status = models.BooleanField(default=False)
    created = models.DateTimeField(editable=False, null=False, auto_now_add=True)
    modified = models.DateTimeField(null=False, auto_now=True)
```

3.7. Modelo Servicio

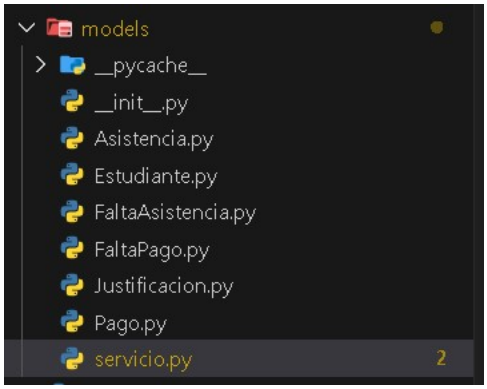
El modelo `Servicio` registra los servicios proporcionados en el comedor universitario. Cada registro de servicio está relacionado con un registro de pago y contiene información sobre el tipo de servicio

(`tipo`), el estado del servicio (`status`) y las fechas de creación y modificación del registro.

Listing 7: Servicio

```
from django.db import models
from .Pago import Pago

class Servicio(models.Model):
    id_pago = models.ForeignKey(Pago, on_delete=models.CASCADE)
    id_servicio = models.BigAutoField(primary_key=True)
    tipo = models.CharField(max_length=100)
    status = models.BooleanField(default=False)
    fecha = models.DateTimeField(editable=False, null=False, auto_now_add=True)
    modified = models.DateTimeField(null=False, auto_now=True)
```



Cada una de estas entidades, como se puede observar, tiene atributos únicos y específicos, así como relaciones bien definidas entre ellas. Estos atributos permiten una gestión detallada y precisa de la información. Por ejemplo, el calendario de asistencia y pagos incluye atributos como la fecha, el estado de asistencia, y las fechas de modificación. Además, se establecen relaciones entre las entidades, como las que existen entre el comensal y sus registros de asistencia, y entre el comensal y sus justificativos de inasistencia o falta de pagos.

Algunos ejemplos de estos atributos son `create`, que registra la fecha de creación del registro; `status`, que indica el estado actual del registro, ya sea de asistencia o de pago; `fecha`, que especifica la fecha correspondiente a cada evento; y `modifig`, que almacena la fecha y hora de la última modificación del registro. Estos atributos y relaciones permiten una administración eficiente y detallada de todos los aspectos relacionados con el comedor universitario.

4. Clase admin.py

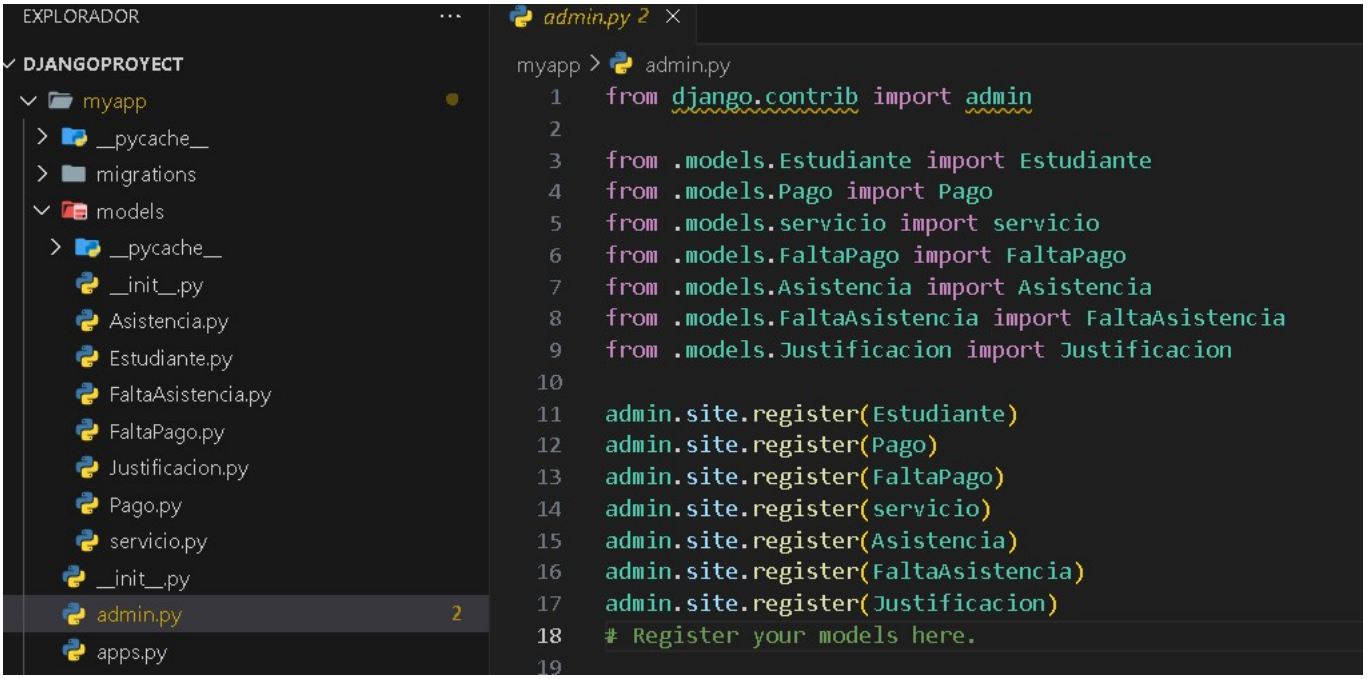
- Después de haber creado cada una de las tablas dentro de los modelos, el siguiente paso es importarlas en `admin.py`. Para registrar cada modelo en el sitio de administración de Django, utilizamos la función `admin.site.register(modelo_x)`, donde `modelo_x` es el nombre del modelo que queremos registrar. Este proceso permite que las tablas creadas sean accesibles y administrables desde la interfaz de administración de Django, facilitando la gestión de los datos.

Listing 8: Código dentro del admin.py

```
from django.contrib import admin
```

```python
from .models.Estudiante import Estudiante
from .models.Pago import Pago
from .models.servicio import servicio
from .models.FaltaPago import FaltaPago
from .models.Asistencia import Asistencia
from .models.FaltaAsistencia import FaltaAsistencia
from .models.Justificacion import Justificacion

admin.site.register(Estudiante)
admin.site.register(Pago)
admin.site.register(FaltaPago)
admin.site.register(servicio)
admin.site.register(Asistencia)
admin.site.register(FaltaAsistencia)
admin.site.register(Justificacion)
```



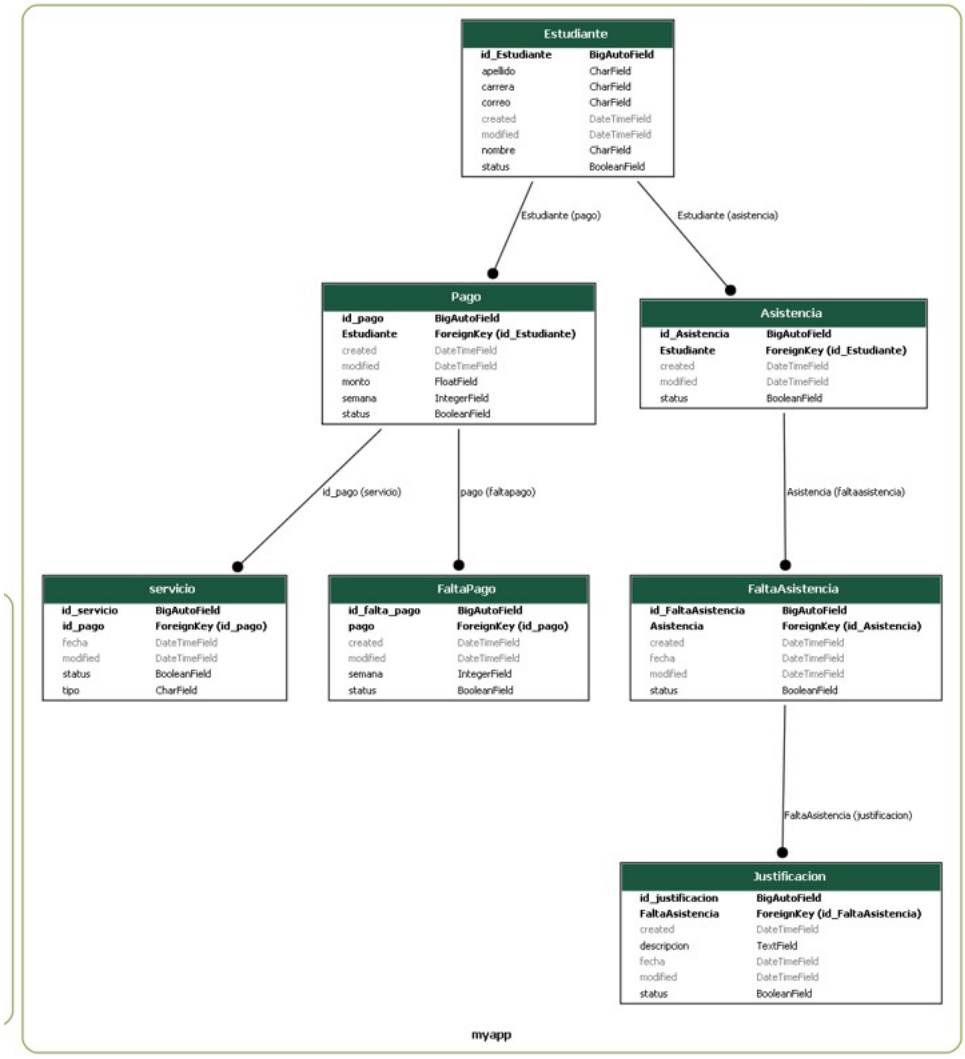
5. Diagramas de bases de datos (ERD) desde Django

- A continuación, utilizamos una función para generar un diagrama detallado de todo lo que hemos hecho hasta ahora. Esto nos permite comparar con el primer diagrama inicial y comprender mejor el trabajo realizado, demostrando la efectividad de esta herramienta.

Listing 9: Comandos para la instalación y uso de Graphviz

```
1. pip install graphviz
2. pip install django-extensions
3. Agregar 'django_extensions' a INSTALLED_APPS en settings.py
4.  Aadir  el siguiente cdigo a settings.py:
GRAPH_MODELS = {
    'all_applications': True,
```

```
    'graph_models': True,
}
5. pip install pyparsing pydot
6. py manage.py graph_models -a > erd.dot
7. py manage.py graph_models -a
8. py manage.py graph_models -a > erd.dot && py manage.py graph_models --pydot -a -g -o
    erd.png
```



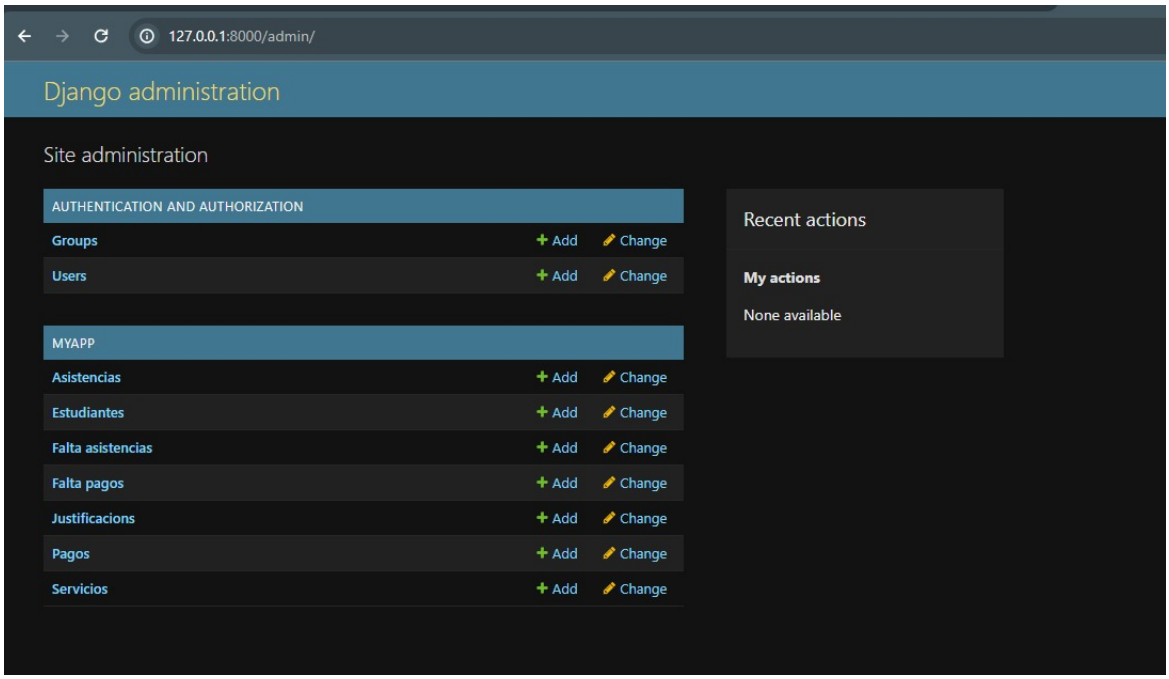
6. Ejecución del manage.py

- Para finalizar, ejecutamos el `manage.py`, lo que aplicará todos estos cambios a la base de datos y nos mostrará detalladamente todos los modelos que hemos creado, así como las configuraciones realizadas, entre otros detalles.

Listing 10: Comando ara el manage.py

```
python manage.py createsuperuser
```

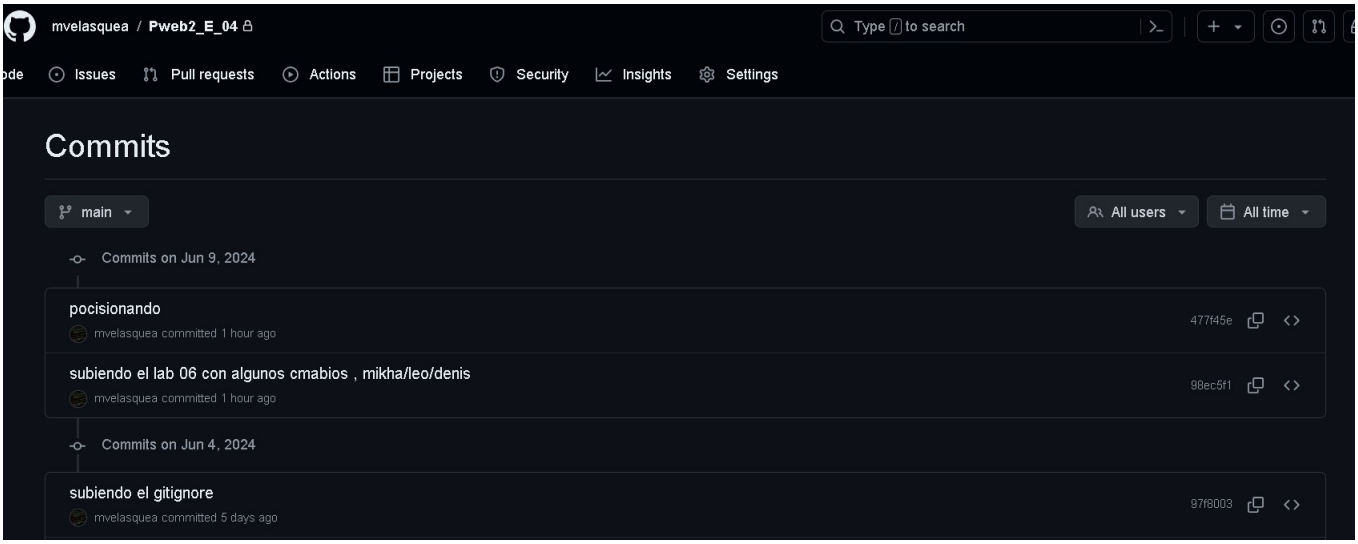
- User :admin
- password :1234



7. URL GitHub

- `https://github.com/mvelasquea/Pweb2_E_04.git`

8. Commits



9. Rúbricas

9.1. Rúbrica para entregable Informe

Tabla 1: Rúbrica para tipo de Informe

Informe		Cumple	No cumple
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y fácil de leer.	20	17
MarkDown	El informe está en formato PDF desde MarkDown README.md, con un formato limpio (buena presentación) y fácil de leer.	17	0
MS Word	El informe está en formato PDF desde plantilla MS Word, con un formato limpio (buena presentación) y fácil de leer.	15	0
Observaciones	Por cada observación se le descontará puntos.	-	-

9.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	1	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	3	
Total		20		18	