Universidad Simón Bolívar
Departamento de computación y Tecnología de la Información
Computación I (CI-2125)

Testing Physics
Informe

Profesores:

- Jorge Baralt Torrijos

Integrantes:

- Oscar Odon 18-10153
- Diana Vegas 19-10254
- Luisana Rodriguez 20-10067

Sección 1 - Grupo D

INTRODUCCION

La vida de un estudiante universitario implica una serie de responsabilidades, desafíos y oportunidades. Uno de los aspectos claves del estudiante es tener un aprendizaje cómodo en donde pueden potenciar su desarrollo académico y profesional, experimentando diferentes métodos para aprender, reforzar y profundizar su conocimiento.

Actualmente vivimos en una era tecnológica muy útil para los estudiantes, ya que facilita el aprendizaje a distancia aprovechando el poder de los dispositivos, aplicaciones y herramientas digitales para impulsar el conocimiento y participación a través de diferentes plataformas, ayuda a ampliar y fortalecer las capacidades digitales, permite la exploración de nuevos conocimientos y ayuda a desarrollar habilidades esenciales como el pensamiento crítico, la resolución de problemas, la comunicación, entre otros.

En el siguiente proyecto nos enfocamos específicamente en los estudiantes de la Universidad Simón Bolívar y la materia de Física II, debido a que en las plataformas de la universidad actualmente no se consiguen variedad de materiales de refuerzo para dicha materia, por lo tanto se dificulta su estudio y aumenta la problemática sobre la alta tasa de reprobados con los que cuenta la asignatura actualmente, lo cual impide el avance de los estudiantes en sus carreras, generando rezagados constantes y limitaciones en solicitudes de Ciclo Básico, como el cambio de carrera o la solicitud de permisos en las coordinaciones.

PROYECTO SIMULADOR DE PARCIALES DE FISICA II (FS1112)

"TESTING PHYSICS"

Decidimos dedicar tiempo en la creación de un programa con enfoque de estudio universitario para Android llamado "Testing Physics", cuyo énfasis es ayudar a los estudiantes del Ciclo Básico a estudiar la asignatura FS1112. Se trata de un simulador de parciales en el cronograma clásico de los tres parciales (30/35/35), en este programa aparecerán una serie de preguntas de selección simple, donde el estudiante podrá contestar marcando la opción que considere correcta basado en su conocimiento previo acerca de la materia, también contara con una serie de comodines que permitirán al estudiante orientarse en el transcurso de la simulación que tendrá una duración preestablecida y al momento de la culminación del parcial se mostrara las respuestas correctas con su respectivo procedimiento.

La aplicación también tendrá en su apartado varias recomendaciones, resúmenes, formulas y ecuaciones básicas de los contenidos a evaluar para que puedan prepararse para presentar la simulación.

"Testing Physics" fue ccreado con el lenguaje de programación Python, ya que es uno de los lenguajes mas versátiles de programación informático y más popular en la actualidad, mayormente se utiliza para crear sitios web, software, automatizar tareas (organizar finanzas o resolver problemas), realizar análisis de datos y crear una variedad de programas diferentes.

En resumen, Python es un lenguaje de alto nivel de programación, poderoso que se ha convertido en un elemento básico en la ciencia de datos, la inteligencia artificial y el desarrollo de aplicaciones web.

Para la creación de la aplicación se utilizó la librería Kivy que utiliza Python como lenguaje de programación, lo cual es una librería o multiplataforma de código abierto para el desarrollo rápido de aplicaciones, se enfoca en el desarrollo de interfaces, graficas innovadoras de programas para Android como las aplicaciones táctiles, transportando el código que se desarrolló en Python para Android, realizando un programa ".apk" que se pueda ejecutar en teléfonos con dicho sistema.

Estas librerías de programación son aliadas valiosas para los desarrolladores, ya que proporcionan códigos desarrollados previamente que se pueden emplear para programar de

manera más rápida, efectiva, eficiente y sencilla. Cada librería está diseñada para abordar problemas comunes o proporcionar funcionalidades específicas.

A continuación, se muestra las librerías que se utilizaron en la creación de la app **"Testing Physics"**

```
import csv
import random
import time
from kivy.uix.popup import Popup
from kivy.uix.scrollview import ScrollView
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.checkbox import CheckBox
from kivy.uix.progressbar import ProgressBar
from kivy.uix.button import Button
from kivy.uix.label import Label
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
from kivy.uix.screenmanager import Screen
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager
from kivy.uix.gridlayout import GridLayout
from kivy.utils import get_color_from_hex
```

Así mismo, se trabajo con códigos de programación para estructurar el lenguaje y a su vez garantizar el correcto funcionamiento de la aplicación que permita tener una buena comunicación entre el usuario y el dispositivo.

A continuación, se mostrarán los códigos para el desarrollo de la app **"Testing Physics"**

- **class SimulateParcialMenuScreen(Screen):** Indica que se está creando una nueva clase que se basa en la clase Screen, lo que permite reutilizar, agregar nuevas funcionalidades específicas y extender la lógica y funcionalidades de la clase base para implementar un menú simulado de manera eficiente y coherente.
- **self.create_widgets():** Se utiliza para solicitar el método create_widgets() de la clase actual, que se encarga de crear y configurar los widgets (elementos de la interfaz gráfica) u otras acciones necesarias para la funcionalidad de la clase.
- **self.parcial_selected = False:** Se utiliza para establecer el atributo parcial_selected en la instancia actual de la clase, lo que puede ser útil para controlar el estado o la lógica de la clase en función de si un parcial está seleccionado o no.

- **layout = GridLayout(cols=1, padding=10):** Es un parámetro de la clase GridLayout, con una sola columna y una almohadilla (padding) de 10 píxeles alrededor de los widgets, lo que permite organizar los widgets en una cuadrícula con un diseño específico.
- **def start_parcial1(self, instance):** Define un método en una clase que se utilizará para iniciar una acción específica, como la ejecución de una parte del programa o una sección específica de la funcionalidad de la clase.

```
19  class SimulateParcialMenuScreen(Screen):
20      def __init__(self, **kwargs):
21          super().__init__(**kwargs)
22          self.create_widgets()
23          self.parcial_selected = False
24
25      def create_widgets(self):
26          layout = GridLayout(cols=1, padding=10)
27          self.add_widget(layout)
28          layout.add_widget(Label(text='Simular Parcial', color=get_color_from_hex('#FFFFFF')))
29          layout.add_widget(Button(text='Parcial 1', on_release=self.start_parcial1, background_color=get_
30          layout.add_widget(Button(text='Parcial 2', on_release=self.start_parcial2, background_color=get_
31          layout.add_widget(Button(text='Parcial 3', on_release=self.start_parcial3, background_color=get_
32          layout.add_widget(Button(text='Regresar al menú', on_release=self.go_menu, background_color=get_
33
34      def start_parcial1(self, instance):
35          self.parcial_selected = True
36          parcial_screen = SimulateParcialScreen(name='simulate_parcial_screen')
37          parcial_screen.parcial_number = 1
38          self.manager.add_widget(parcial_screen)
39          self.manager.current = 'simulate_parcial_screen'
```

Imagen 1: Muestra de los primeros códigos del programa ‘Testing Physics”

- **def go_menu(self, instance):** Se utilizará para navegar o cambiar a un menú específico dentro de una aplicación o interfaz gráfica.
- **self.manager.current = 'menu_screen':** Se utiliza para cambiar la pantalla actual en un gestor de pantallas a una pantalla específica identificada como 'menu_screen', lo que permite controlar la navegación entre diferentes pantallas en una aplicación.
- **parcial_screen.parcial_number = 2:** Este atributo se utiliza para almacenar información relacionada con el número de parcial que se está simulando.
- **layout.add_widget(Label(text='Resumen de de Ecuaciones', color=get_color_from_hex('#FFFFFF'))):** Agrega un widget Label al layout

especificado con el texto 'Resumen de Ecuaciones' y el color del texto en blanco, lo que permite mostrar información en la interfaz de usuario.

- **class TestingPhysics(App):** Define una nueva clase llamada TestingPhysics que hereda de la clase App, lo que permite crear una aplicación en Python utilizando las funcionalidades proporcionadas por la clase base App.
- **def build(self):** define el método build en una clase de Python, el cual se utiliza para construir y retornar la estructura de la interfaz gráfica de una aplicación, organizando los elementos visuales y widgets que conformarán la interfaz de usuario.
- **def on_start(self):** Es definir un método en una clase de Python que se ejecutará al inicio de una aplicación Kivy, permitiendo realizar tareas de inicialización y preparación antes de que la aplicación esté completamente lista para su uso.
- **def resumen_teorico(self, instance):** define un método en una clase de Python que esta destinado a manejar eventos o acciones relacionadas con la presentación de un resumen teórico en la aplicación.
- **self.questions = []:** Aquí se inicializa una lista vacía llamada questions. Esta lista se utilizará para almacenar las preguntas que se cargarán más adelante en la aplicación.
- **self.correct_answers = 0:** Se inicializa la variable correct_answers en 0. Esta variable se utiliza para llevar un registro del número de respuestas correctas que ha dado el usuario.
- **self.timer = 7200 # 2 horas en segundos:** Se establece el temporizador en 7200 segundos, que equivale a 2 horas. Este temporizador se utiliza para limitar el tiempo disponible para completar el parcial o alguna otra funcionalidad relacionada con el tiempo en la aplicación.
- **self.load_questions():** Se llama al método load_questions(), se encarga de cargar las preguntas desde alguna fuente de datos y almacenarlas en la lista questions para su uso posterior en la aplicación.
- **def create_widgets(self):** Se utiliza para configurar los widgets de la interfaz de usuario de la aplicación, permitiendo la personalización y diseño de la interfaz gráfica.

- **self.question_labe = Label(text='', color=get_color_from_hex('#FFFFFF'), size_hint_y=0.2):** Crea un Label con texto vacío, color blanco, se utilizará para mostrar preguntas u otra información en la interfaz de usuario de la aplicación.
- **layout.children[1].add_widget(self.options_layout):** Esto se utiliza para organizar y estructurar la disposición de los widgets en la interfaz de usuario de la aplicación.
- **self.progress_bar = ProgressBar(max=5, value=self.current_question + 1, size_hint_y=0.1):** Se utiliza para visualizar el progreso o avance en una serie de preguntas o tareas en la interfaz de usuario de la aplicación.
- **self.timer_label = Label(text='', color=get_color_from_hex('#FFFFFF'), size_hint_y=0.1):** Se usa para mostrar un temporizador o una cuenta regresiva en la interfaz de usuario.
- **layout.add_widget(self.next_button):** Permite que el botón se visualice y forme parte de la interfaz de la aplicación donde el usuario interactúe con él para realizar alguna acción específica, como avanzar a la siguiente pregunta.
- **def load_questions(self):** Se utiliza para cargar preguntas desde la base de datos en un archivo CSV en el atributo questions de la instancia de clase.
- **def update_question(self):** Se encarga de actualizar la pregunta actual en la interfaz de usuario con la información correspondiente.
- **checkbox.option_index = i:** Se utiliza para establecer el atributo option_index de cada widget CheckBox en un valor único de 1 a 3, permite a la aplicación identificar qué opción ha seleccionado el usuario para una pregunta de opción múltiple..
- **def next_question(self, instance):** método esté asociado con un widget de botón que, cuando se hace clic, avanzará a la siguiente pregunta de un cuestionario o encuesta
- **if self.current_question < 4:** Si current_question es igual o mayor que 4, entonces no hay más preguntas para mostrar y es probable que el cuestionario o la encuesta hayan terminado.
- **self.current_question += 1:** Se utilizara para avanzar a la siguiente pregunta después de que el usuario haya respondido a la pregunta actual.
- **self.update_question():** Se usa para actualizar la pregunta que se muestra actualmente en la interfaz de usuario de la aplicación.

- **self.submit_exam(instance):** Se llame a esta línea de código cuando el usuario haya respondido a todas las preguntas de la prueba o examen y esté listo para enviar sus respuestas.
- **self.calculate_score():** Este método sea responsable de calcular la puntuación del usuario en la prueba o examen en función de sus respuestas.
- **self.show_results():** Este método sea responsable de mostrar los resultados del usuario en la prueba o examen, incluida su puntuación, las respuestas correctas o incorrectas y otra información relevante.
- **question = self.questions[self.current_question]:** Esta línea recupera la pregunta actual de la lista de preguntas.
- **selected_option = None:** Esta línea inicializa la variable selected_option en Ninguno, lo que indica que aún no se ha seleccionado ninguna opción.
- **if selected_option == int(question[6]):** Compara si la opción seleccionada por el usuario es igual a la respuesta correcta de la pregunta actual. La respuesta correcta se encuentra en la posición 6 de la lista question
- **self.correct_answers += 1:** Si la opción seleccionada por el usuario coincide con la respuesta correcta, se incrementa en 1 el contador de respuestas correctas almacenado en el atributo correct_answers de la instancia.
- **popup_layout = BoxLayout(orientation='vertical', padding=10):** Se utilizará en el código para construir una ventana emergente que mostrará los resultados del examen al usuario.
- **if self.correct_answers < 5:** Se utiliza para determinar si el usuario ha respondido correctamente a menos de 5 preguntas y, en consecuencia, mostrar un mensaje de "Felicidades" o "Inténtalo de nuevo" en la ventana emergente.
- **if int(question[6]) != index + 1:** Se utiliza para verificar si la respuesta del usuario a la pregunta actual es incorrecta, ya que la respuesta correcta no coincide con el índice actual más uno.
- **popup.open():** Se utiliza para mostrar la ventana emergente de resultados al usuario después de que haya respondido a todas las preguntas.

- **self.submit_exam(None):** Se encarga de enviar o guardar la información del examen una vez que el usuario ha terminado de responder las preguntas o cuando el tiempo se ha agotado.
- **TestingPhysics().run():** Con esta línea de código culminamos ya que define el punto de entrada de la aplicación y es responsable de iniciar el ciclo de vida de la aplicación.

Esta serie de códigos de programación son la base de la tecnología que nos permitieron desarrollar la aplicación y crear soluciones que puedan ayudar a los estudiantes que actualmente o en el futuro deban cursar esta asignatura.

Resultado

Para finalizar, con este simulador se busca lograr ser un medio de apoyo o recurso en línea para los profesores del área y los alumnos, ofreciendo contenidos, ejercicios, retos y herramientas útiles que permitan a los estudiantes autoevaluar su preparación previa a la evaluación de Física II.