Zuhar Regal 4227160

IFS354 Individual Assignment

Due Date: 1 June 2024 11:55 PM

Lecturer: Ruchen Wyngaarg

Table of Contents

Company Overview

TechInnovators Inc. is a rapidly growing engineering firm that has established itself as a leader in the design and development of cutting-edge Internet of Things (IoT) devices and high-end computing equipment. Founded in 2010, the company has experienced tremendous growth, expanding from a small team of 10 engineers to over 100 employees across multiple offices and development labs.TechInnovators Inc. prides itself on its innovative spirit and commitment to pushing the boundaries of technology. The company's diverse portfolio includes smart technology devices, industrial automation solutions, and high-performance computing systems for data analysis and product development. With a strong emphasis on research and development, TechInnovators Inc. heavily invests in state-of-the-art equipment and tools to ensure its engineers and developers have access to the resources they need to create revolutionary products.

TechInnovators Inc

Challenges

As TechInnovators Inc. continues to grow, the company has encountered significant challenges in managing and tracking its extensive inventory of equipment and devices. With multiple development teams working on various projects simultaneously, the efficient allocation and utilization of resources have become increasingly complex. The company's inventory consists of a wide range of specialized equipment, including:

1. Prototyping boards and development kits for IoT devices
2. High-performance computing systems
3. Testing and measurement equipment for quality assurance
4. Specialized tools and instruments for hardware and software development

With no centralized system in place, employees often struggle to locate the specific equipment they need for their projects. This has led to significant delays and inefficiencies, as engineers and developers spend valuable time searching for the necessary resources instead of focusing on their core tasks.

Additionally, the lack of visibility into equipment usage patterns makes it challenging for TechInnovators Inc. to optimize resource allocation and plan for future equipment needs effectively. Without accurate data on which equipment is being utilized and how frequently, the company faces the risk of either over-investing in unnecessary resources or facing shortages that could hinder product development. Furthermore, the absence of a tracking system increases the risk of equipment loss or misplacement, which could result in substantial financial losses and further delays in project timelines. As TechInnovators Inc. continues to expand and take on more ambitious projects, the need for a robust and efficient equipment tracking system has become increasingly apparent. The company recognizes that addressing this challenge is crucial to maintaining its competitive edge and delivering innovative products to market on time and within budget.

Introduction

As the lead software engineer at Techinnovators IncI.,I have been tasked to develop a functional equipment inventory tracking system to address the challenges faced by the company in managing its extensive inventory of specialized equipment.The proposed solution uses the company's existing enterprise software built on Oracle APEX and utilize Python for integration and lightweight applications.

<u>Overview of Solution</u>

Oracle APEX Integration

• Developed an Oracle APEX application to serve as the central inventory management system.
• Implemented functionality to add new equipment and manage existing equipment quantities.
• Created user interfaces for checking out equipment (equipment quantity is reduced to one when in use), tracking equipment usage and viewing reports.This interface is built for the admin personnel who are responsible for assigning equipment to employees.

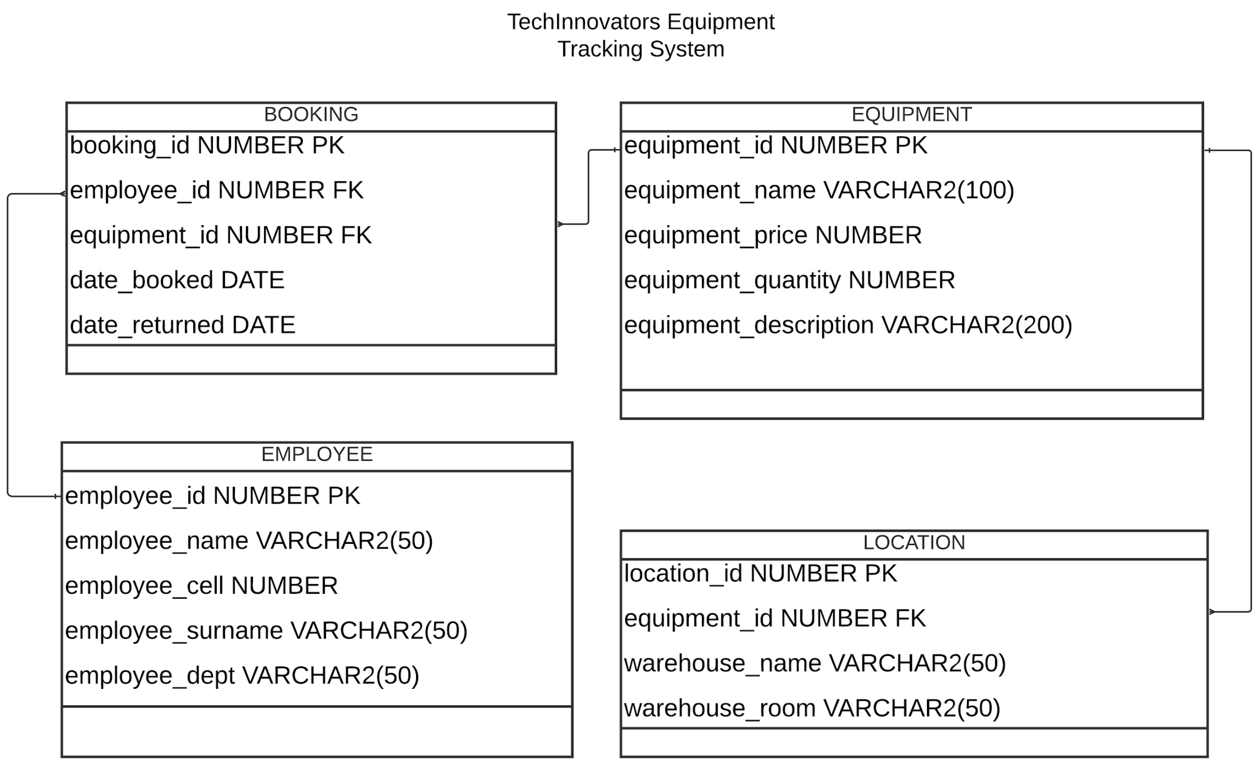
Python-based Integration

• Developed a Python-based application which can be used by employees to interface with the Oracle APEX back-end through custom-built APIs.

• The application allows employees to view a report of the equipment they've used in the past and the equipment they are currently using.
• Employees are able to return/check-in equipment by inputting the Booking
ID associated with specific equipment and scan a QR code embedded with their employee ID to record that the equipment has been returned/checked in.(Inventory quantity is returned back to normal after the equipment has been checked in)

Technical Documentation

Database Schema and Data Dictionary


TechInnovators Equipment Tracking System
BOOKING
booking_id NUMBER PK
employee_id NUMBER FK
equipment_id NUMBER FK
date_booked DATE
date_returned DATE
EQUIPMENT
equipment_id NUMBER PK
equipment_name VARCHAR2(100)
equipment_price NUMBER
equipment_quantity NUMBER
equipment_description VARCHAR2(200)
EMPLOYEE
employee_id NUMBER PK
employee_name VARCHAR2(50)
employee_cell NUMBER
employee_surname VARCHAR2(50)
employee_dept VARCHAR2(50)
LOCATION
location_id NUMBER PK
equipment_id NUMBER FK
warehouse_name VARCHAR2(50)
warehouse_room VARCHAR2(50)

BOOKING Table

This table keeps track of the bookings made by employees for equipment. Each record represents a single booking.

- booking_id: A unique identifier for each booking. It serves as the primary key (PK) of the table.

- employee_id: A reference to the employee who made the booking. This is a foreign key (FK) that links to the `EMPLOYEE` table.

- equipment_id: A reference to the equipment being booked. This is a foreign key (FK) that links to the `EQUIPMENT` table.

- date_booked: The date on which the equipment was booked.

- date_returned: The date on which the equipment was returned.

EQUIPMENT Table

This table contains details about the equipment available for booking. Each record represents a piece of equipment.

- equipment_id: A unique identifier for each piece of equipment. It serves as the primary key (PK) of the table.

- equipment_name: The name of the equipment.

- equipment_price: The price of the equipment.

- equipment_quantity: The quantity of this equipment available.

- equipment_description: A description of the equipment.

EMPLOYEE Table

This table stores information about employees who can book equipment. Each record represents an individual employee.

- employee_id: A unique identifier for each employee. It serves as the primary key (PK) of the table.

- employee_name: The first name of the employee.

- employee_cell: The cell phone number of the employee.

- employee_surname: The surname (last name) of the employee.

- employee_dept: The department to which the employee belongs.

LOCATION Table

This table contains information about the locations where equipment is stored. Each record represents a location.

- location_id: A unique identifier for each location. It serves as the primary key (PK) of the table.

- equipment_id: A reference to the equipment stored at this location. This is a foreign key (FK) that links to the `EQUIPMENT` table.

- warehouse_name: The name of the warehouse where the equipment is stored.

- warehouse_room: The specific room within the warehouse where the equipment is stored.

Relationships

- The `BOOKING` table has foreign keys referencing the `EMPLOYEE` and `EQUIPMENT` tables, establishing a relationship between bookings, employees, and equipment.

- The `LOCATION` table has a foreign key referencing the `EQUIPMENT` table, establishing a relationship between equipment and their storage locations.

This schema ensures that each piece of equipment can be tracked by its bookings and locations, and each booking can be associated with a specific employee and equipment item.

System Architecture and Design

The architecture is a hybrid of Oracle Apex and python applications, making use of Oracle APEX Database for centralized management and Python for user interaction.

System Design Components:

<u>1. Oracle APEX Application:</u>

- Central Inventory Management System: Manages all equipment and booking data.
- User Interfaces for Admins:
  - Equipment Management: Add, update, and manage equipment.
  - Checkout and Tracking: Check out equipment to employees, automatically reduces quantity by 1 when equipment is checked out.
  - Reports: Generate reports on equipment usage, patterns etc (more details under end user manual).

<u>2. Python-based Application:</u>

- Employee Interaction:
  - View Reports: Employees can view their past and current equipment usage by inputting their employee id
  - Return/Check-in Equipment: employees input Booking ID and scan QR code to check in equipment, equipment quantity is updated to initial amount.

<u>3. Data Flow</u>

Data flows between the Oracle APEX backend and the Python application through RESTful APIs. This ensures that both systems are synchronized and data is up-to-date.

<u>4. Workflow</u>

Admin Actions:

- Admins add or update equipment data in Oracle APEX.
- Changes in equipment status (checked out) are updated in real-time.

5.Employee Actions:

- Employees use the Python application to view reports or check-in equipment.
- Python application sends and retrieves data from Oracle APEX via APIs.

6.User Interfaces

Oracle APEX Application Interface:

1. Admin Home:
   - Overview of various application functions.

2. Equipment Management:
   - Forms for adding new equipment.
   - Form for updating existing equipment quantities.

3. Checkout and Tracking:
   - Form to assign equipment to employees.
   - Real-time tracking of equipment bookings.

4. Reports:
   - Comprehensive equipment usage reports etc (more details under end user manual).
   - Detailed booking summaries and inventory levels.

Python Application Interfaces:

1. Report Viewer:
   - Input field for Employee ID.
   - Display past and current equipment usage for employees.

2. Check-in Interface:
   - Input field for Booking ID.

- QR code scanner integration for employee verification.
- Confirmation of checkin and equipment return.

The system design ensures that TechInnovators Inc. has a robust, efficient, and secure equipment management solution. Oracle APEX handles centralized inventory management, while the Python application provides a user-friendly interface for employees to interact with the system. The integration via RESTful APIs ensures data consistency and real-time updates across the system.

System Architecture

This architecture involves multiple components integrated to form a comprehensive system for managing and interacting with the TechInnovators Equipment Tracking System. Here's an explanation of each component and how they interact:

<u>Components:</u>

1. Admin Application:
   - Purpose: This application is used by administrators to manage the system, such as adding new equipment, updating employee information, and overseeing bookings.
   - Platform: Oracle Apex (Application Express) is used to build this application.
   - Features: Provides a web-based interface for performing administrative tasks, managing database records, and generating reports.

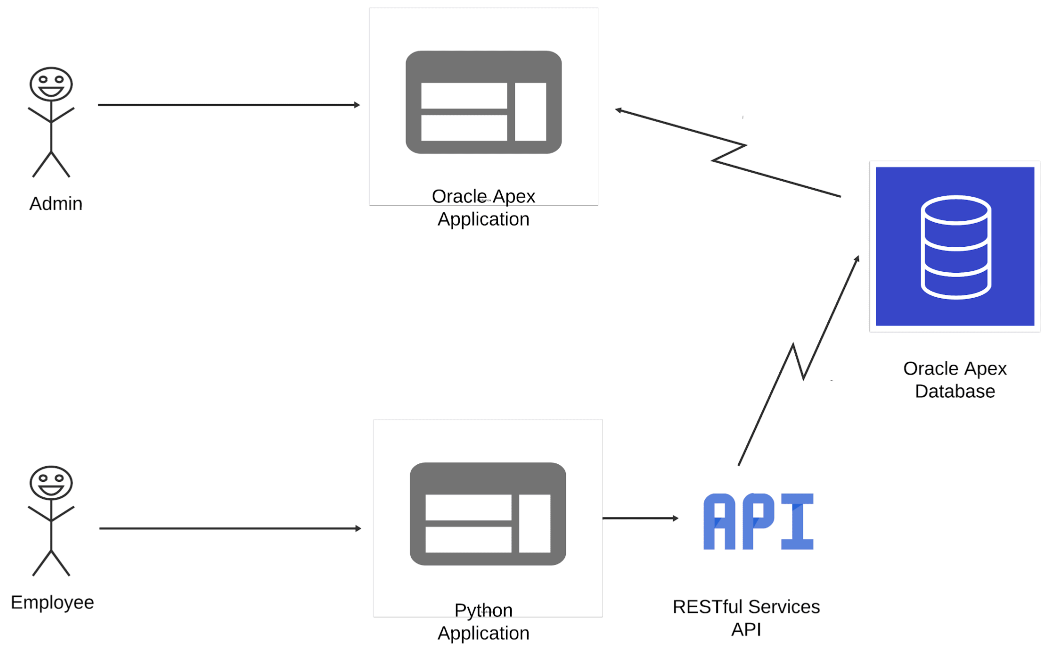
2. Employee Application:
   - Purpose: This application is designed for employees to chEckin equipment, view their booking history.
   - Platform: Built using python, providing a user-friendly interface for employee interactions.

3. Oracle Apex Database:

- Purpose: This is the central database that stores all the data related to the equipment tracking system.
- Technology: Oracle Database, managed using Oracle Apex.
- Contents: Tables such as BOOKING, EQUIPMENT, EMPLOYEE, and LOCATION, as described in the database dictionary.

4. Python Application:
   - Purpose: This application serves as an intermediary to perform more complex logic or computations that might not be directly handled by Oracle Apex. It might also be used for data analysis, generating custom reports, or interacting with other external systems.
   - Features: Executes complex business logic, data processing tasks, and serves as a backend for the RESTful Services API.

5. RESTful Services API:
   - Purpose: Provides a set of web services that enable communication between different applications (such as the Admin Application, Employee Application, and the Python Application) and external systems.
   - Technology: Implemented in Python, offering endpoints to interact with the Oracle Apex database.



API Documentation

Custom-built RESTful APIs facilitate communication between the Oracle APEX backend and the Python application. Verified using Postman.

Table Name	**Endpoint**	**Method**	**Parameter Name**	**Data Type**	**Description**
Booking	/checkin	POST	p_date_returned	String	The date the equipment is returned, formatted as "YYYY/MM/DD"
Booking	/checkin	POST	p_booking_id	Integer	The unique identifier for the booking.
Booking	/employee_id	GET	/	/	Fetches all booking data based on optional query parameters
Equipment	/equipment	GET	/	/	Fetches all equipment or equipment

					data based on optional query
Equipment	/equipment	POST	p_equipment_id	Integer	The unique identifier for the equipment.
			p_equipment_quantity	Integer	The quantity of the equipment, used to update equipment quantity back to original amount.

POST /booking

- Base URL : https://apex.oracle.com/pls/apex/ifs354_ind_assignment/test/checkin
- Description: This endpoint is used to update the return date of a booking in the system.
- Parameters:
  - `p_date_returned` (String): This parameter takes the date when the equipment is returned. It must be in the "YYYY/MM/DD" format.
  - `p_booking_id` (Integer): This parameter specifies the unique identifier for the booking. It is essential for identifying which booking record needs to be updated.

GET /Employee_id

- Base URL: https://apex.oracle.com/pls/apex/ifs354_ind_assignment/test/employee_id
- Description: This endpoint is used to retrieve booking records from the system. It can be configured to fetch all booking data based on query.

POST /equipment
- Base URL: https://apex.oracle.com/pls/apex/ifs354_ind_assignment/test/equipment
- Description: This endpoint is used to update the quantity of a specific piece of equipment in the inventory.
- Parameters:
  - `p_equipment_id` (Integer): This parameter specifies the unique identifier for the equipment. It is essential for identifying which equipment record needs to be updated.
  - `p_equipment_quantity` (Integer): This parameter specifies the quantity of the equipment that needs to be updated.

GET /equipment
- Base URL: https://apex.oracle.com/pls/apex/ifs354_ind_assignment/test/equipment
- Description: This endpoint is used to retrieve equipment records from the system. It can be configured to fetch the equipment id and equipment quantity.

Third Party Libraries and Dependencies

To build and integrate the Oracle APEX application with the Python-based interface,third-party libraries and dependencies are used. These libraries facilitate various functionalities, such as HTTP requests, JSON handling, QR code scanning, and GUI creation. Below is a comprehensive list of these dependencies along with their purposes.

Python Libraries

1. http.client
   - Purpose: To handle HTTP requests and responses.

- Description: This built-in Python module provides classes for working with HTTP requests and responses, which are essential for interacting with the RESTful APIs of the Oracle APEX backend.

2. json
   - Purpose: To parse JSON data.
   - Description: This built-in Python module is used to encode and decode JSON data, which is the format used for data interchange between the Python application and the Oracle APEX backend.

3. tkinter
   - Purpose: To create GUI elements.
   - Description: This built-in Python module is used to create graphical user interfaces. It provides various widgets like buttons, labels, and text fields for building the application's user interface.

4. tkinter.simpledialog
   - Purpose: To handle simple dialog boxes.
   - Description: This module is part of tkinter and is used for creating simple dialog boxes that prompt the user for input, such as employee IDs and booking IDs.

5. tkinter.ttk
   - Purpose: To provide themed widget set for tkinter.
   - Description: This module enhances tkinter by adding themed widgets, which provide a more modern look and feel to the application.

6. tkinter.messagebox
   - Purpose: To display message boxes.
   - Description: This module is used to display various types of message boxes, such as error messages, information alerts, and confirmation dialogs.

7. cv2 (OpenCV)
- Purpose: To handle QR code scanning.
- Description: OpenCV is a powerful library for computer vision tasks. In this application, it is used to capture video from the camera and decode QR codes embedded with employee IDs.

8. pyzbar
- Purpose: To decode QR codes.
- Description: Pyzbar is a library used for reading one-dimensional and two-dimensional barcodes, including QR codes. It is used in conjunction with OpenCV to scan and decode QR codes from video frames.

9.Datetime Library:
- The `datetime` library is imported to handle date and time operations.
- When retrieving employee bookings, the dates (`date_booked` and `date_returned`) are formatted using the `datetime` library to make them more user-friendly. If `date_returned` is not available, it displays "None."

For the Oracle APEX application and database, the system relies primarily on Oracle's built-in capabilities. However, some third-party tools and libraries can be integrated or used alongside Oracle APEX and its database to enhance functionality. Here are some common third-party components and dependencies that might be relevant:

<u>Third-Party Libraries and Tools for Oracle APEX Application and Database</u>

4. APIs and Integrations:
- RESTful Web Services: Allows integration with external systems, enabling APEX to fetch or send data to other platforms.

5. Authentication and Authorization:
- Used for securing API calls and providing secure user authentication.

By using these third-party libraries and dependencies, the system efficiently integrates the Oracle APEX backend with the Python-based application. The combination of these technologies allows for seamless data management, user interaction, and solid functionality for both administrators and employees.

Setup and Configuration Guides

Hardware and Software Prerequisites

Oracle Application Express (APEX) has specific hardware and software prerequisites. Below is a detailed list of prerequisites categorised appropriately.

1. Operating System Compatibility

Oracle APEX supports various operating systems as long as they meet the requirements for the Oracle Database:

- HP-UX Itanium

- IBM AIX on POWER Systems

- Linux x86-64

- Microsoft Windows x64

- Oracle Solaris on SPARC

- Oracle Solaris on x86-64

2. Oracle Database Requirements

Database Versions: Oracle APEX release 22.1 requires Oracle Database release 12.1.0.2 or later.

- Database Editions: Supported on all Oracle Database editions, including:

  - Enterprise Edition (EE)

  - Standard Edition (SE)

  - Express Edition (XE)

- Database Configurations: Can be installed in both single-instance database.

<u>3.Browser Requirements</u>

JavaScript-enabled browser: Supports current and prior major releases of:

  - Google Chrome

  - Mozilla Firefox

  - Apple Safari

  - Microsoft Edge

<u>4.Web Server Requirements</u>

Oracle REST Data Services (ORDS): Requires version 20.x or later for Oracle APEX.

<u>5.Disk Space Requirements</u>

- For APEX Software Files:

  - 310 MB for English-only download (apex_22.1_en.zip)

  - 705 MB for full download (apex_22.1.zip)

- In APEX Tablespace:

  - 220 MB free space

- In SYSTEM Tablespace:

  - 100 MB free space

- For Each Additional Language (other than English):
  - 60 MB free space in APEX tablespace

6. Additional Hardware Prerequisites

For running an application server like Oracle WebLogic Server to host ORDS, the following hardware requirements must be met:

- RAM:
  - Minimum: 4 GB
  - Preferred for production: 8 GB or more
- Disk Space: 10 GB or more
  - Minimum: 8 GB
  - Recommended for production: 16 GB or more
- Network Requirements:
  - Static IP address or a loopback adapter with a static IP address
  - High-speed internet connection for external API calls and data transfer

Software Prerequisites for Oracle APEX and Database

1. Version: Oracle APEX 20.1 or later.

2. Account: You must create an Oracle account to download the software from the Oracle website.

3. Download:

Oracle APEX can be downloaded from [Oracle APEX Downloads](https://www.oracle.com/tools/downloads/apex-downloads.html).

- Options available: Download for all languages or the English-only version.

- Agreement: You must agree to the Oracle License Agreement before downloading.

Operating System

Oracle APEX is compatible with various operating systems:

- Windows

- macOS

- Linux

If you are licensed for Oracle Database, no additional licenses are required to set up and use Oracle APEX. Ensure the system meets the required hardware prerequisites as outlined previously.

Hardware prerequisites For the Server Running the Python Application

- Network: High-speed internet connection. The server needs to have at least 50 Mbps downstream capacity from the internet. The connection between client and server should have at least 20 Mbps bandwidth, and no more than 200ms latency.

- Processor (CPU): Python itself is not very demanding on the CPU, so even older processors should be able to run Python code without any problems. A dual-core processor or higher is recommended for better performance, but Python can run on single-core processors as well.

- Memory (RAM): For basic Python programming, 2GB of RAM should be sufficient. However, for more complex programs or when working with large data sets, you may need more RAM, 8 GB RAM

- Storage: Python itself does not require much storage space. The Python interpreter and standard libraries take up only a few hundred megabytes of disk space. Depending on the size of the projects you are working on and the data you are handling, you may need additional storage space, at least 500MB of free disk space even if the IDE is already installed.

- Operating System: Python is supported on various operating systems, including Windows, macOS, and Linux. Ensure that your hardware meets the requirements.

<u>Software Prerequisites for the Python Application</u>

1. Python:

   - Version: Python 3.8 or later can be downloaded from [python downloads] (https://www.python.org/downloads/)

2. Python Libraries:

   - `To install the specified Python packages, you can use the package manager `pip`, which is included with Python. Below are the commands to install each of the packages:

- `requests`
- `json` (part of the Python Standard Library, no installation required)
- `http.client` (part of the Python Standard Library, no installation required)
- `tkinter`
- `opencv-python` (for OpenCV)
- `pyzbar`
- `Date` (no specific package named "Date"; might refer to `datetime` in the Python Standard Library)

4. IDE/Code Editor:

   - PyCharm can be downloaded from [Jetbrains](

https://www.jetbrains.com/pycharm/download/?section=windows)

   - Visual Studio Code to enable code scanner can be downloaded from (https://aka.ms/vs/17/release/vc_redist.x64.exe)

Setup Instructions

To set up a system that involves Oracle APEX and Python, you need to integrate both technologies to interact with each other. Here's a high-level outline to set up and integrate Python with Oracle APEX:

Step 1: Set Up Oracle APEX

1. Install Oracle APEX: Follow the [Oracle APEX installation guide](https://docs.oracle.com/en/database/oracle/application-express/).

Step 2.log in to workspace with credentials :

WORKSPACE NAME : IFS354_IND_ASSIGNMENT
USERNAME : RWYNGAARD@UWC.AC.ZA
PASSWORD : 1234567!

Step 3: Set up Python x Oracle Apex Application

Configure Oracle APEX to Work with Python

1. Enable RESTful Services in APEX:

   - Log in to Oracle APEX as an administrator.

   - Go to the SQL Workshop > RESTful Services.

   - Enable RESTful services for the schema.

2. Create RESTful Web Services:

   - Create RESTful web services in APEX that will interact with the Oracle database. This will allow the Python scripts to send HTTP requests to APEX, which will execute SQL or PL/SQL and return results. (Under API Documentation)

3. Go to app builder and run TechInnovators Inc. Equipment Tracking System

Step 4: Run Python Scripts in Pycharm

CHECK IN SCRIPT

Python Script Folder (Zuhar Regal 4227160 Individual Assignment\Python Scripts\Employee_CheckIn.py)

```
import cv2
import http.client
import tkinter as tk
from tkinter import simpledialog, messagebox
from datetime import date
from pyzbar.pyzbar import decode
import json


def get_employee_id_from_qr():
   # Function to capture employee ID from QR code
   capture = cv2.VideoCapture(0)

   while True:
       ret, frame = capture.read()
       cv2.imshow("QR Scanner", frame)

       if cv2.waitKey(1) & 0xFF == ord('q'):
           break

       # Decode QR code
       decoded_objects = decode(frame)
       if decoded_objects:
           for obj in decoded_objects:
               data = obj.data.decode('utf-8')
               return data

   capture.release()
   cv2.destroyAllWindows()

   return None


def get_id_from_api(endpoint, booking_id):
   # Function to fetch data from API based on endpoint and booking ID
   try:
       conn = http.client.HTTPSConnection("apex.oracle.com")
       api_url =
f"https://apex.oracle.com/pls/apex/ifs354_ind_assignment/test/{endpoi
nt}?booking_id={booking_id}"
       print(f"Fetching {endpoint} with URL: {api_url}")  # Debugging
print statement
```

```
       conn.request("GET", api_url)
       res = conn.getresponse()
       data = res.read().decode('utf-8')
       response_data = json.loads(data)
       if 'items' in response_data:
           for item in response_data['items']:
               if item['booking_id'] == booking_id:
                   return item
   except Exception as e:
       # Display error message if an exception occurs during API
request
       messagebox.showerror("Error", f"An error occurred while
fetching data from the API: {str(e)}")
       return None


def update_equipment_quantity(equipment_id):
   # Function to update equipment quantity in the database
   try:
       print("Updating equipment quantity for equipment ID:",
equipment_id)

       # Fetch current equipment quantity
       conn = http.client.HTTPSConnection("apex.oracle.com")
       api_url =
f"https://apex.oracle.com/pls/apex/ifs354_ind_assignment/test/equipme
nt?pequipment_id={equipment_id}"
       conn.request("GET", api_url)
       res = conn.getresponse()
       data = res.read().decode('utf-8')
       equipment_data = json.loads(data)

       # Check if equipment data is retrieved successfully
       if 'items' in equipment_data:
           for item in equipment_data['items']:
               if item['equipment_id'] == equipment_id:
                   current_quantity = item['equipment_quantity']
                   break
       else:
           # Display error message if equipment data retrieval fails
           messagebox.showerror("Error", f"Failed to fetch current
equipment quantity for equipment ID: {equipment_id}")
           return

       # Increment quantity by one
```

```
        updated_quantity = current_quantity + 1

        # Update equipment quantity
        api_url =
"https://apex.oracle.com/pls/apex/ifs354_ind_assignment/test/equipmen
t"
        payload = json.dumps({
            "pequipment_id": equipment_id,
            "pequipment_quantity": updated_quantity
        })
        headers = {
            'Content-Type': 'application/json'
        }
        conn.request("POST", api_url, payload, headers)

        # Get response from the update request
        res = conn.getresponse()
        if res.status == 200:
            # Display success message if equipment quantity is updated
successfully
            messagebox.showinfo("Success", "Equipment quantity updated
successfully!")
        else:
            # Display error message if equipment quantity update fails
            messagebox.showerror("Error", f"Failed to update equipment
quantity with status code: {res.status}")
    except Exception as e:
        # Display error message if an exception occurs during
equipment quantity update
        messagebox.showerror("Error", f"An error occurred while
updating equipment quantity: {str(e)}")


def make_request():
    # Function to make API request based on user input
    booking_id = simpledialog.askinteger("Booking ID", "Enter Booking
ID:")
    if booking_id is not None:
        employee_id = get_employee_id_from_qr()
        if employee_id is not None:
            print(f"Scanned employee ID: {employee_id}")
            employee_data = get_id_from_api("employee_id", booking_id)
            if employee_data is not None:
                employee_id_from_api = employee_data['employee_id']
```

```
               equipment_id_from_api =
employee_data.get('equipment_id')
               print(f"Employee ID from API: {employee_id_from_api}")
               if int(employee_id) == int(employee_id_from_api):
                   today_date = date.today().strftime("%Y/%m/%d")
                   conn =
http.client.HTTPSConnection("apex.oracle.com")
                   api_url =
f"https://apex.oracle.com/pls/apex/ifs354_ind_assignment/test/checkin
?pbooking_id={booking_id}&pdate_returned={today_date}"
                   conn.request("POST", api_url)
                   res = conn.getresponse()

                   if res.status == 200:
                       # Display success message if API request is
successful
                       messagebox.showinfo("Success", "Check-In
successful!")
                       if equipment_id_from_api is not None:

update_equipment_quantity(equipment_id_from_api)
                       else:
                           print(f"Failed to fetch equipment ID for
booking ID: {booking_id}")
                   else:
                       # Display error message if API request fails
                       messagebox.showerror("Error", f"Request failed
with status code: {res.status}")
               else:
                   # Display error message if employee ID does not
match
                   messagebox.showerror("Error", "Invalid employee
ID: Scanned employee ID does not match the employee ID from the
API.")
           else:
               # Display error message if no employee data is found
               messagebox.showerror("Error", f"No employee data found
for booking ID {booking_id}.")
       else:
           # Display error message if QR code scanning fails
           messagebox.showerror("Error", "Failed to scan QR code.")
   else:
       # Display error message if no booking ID is provided
       messagebox.showerror("Error", "No booking ID provided.")
```

```
# Create Tkinter window
root = tk.Tk()
root.title("Check-In Application")

# Create button to trigger API request
btn_checkin = tk.Button(root, text="Check-In", command=make_request)
btn_checkin.pack(pady=10)

# Run the Tkinter event loop
root.mainloop()
```

This code is a Python application that uses OpenCV for QR code scanning, HTTP client for API requests, and Tkinter for a graphical user interface (GUI). The purpose of the application is to facilitate the check-in process of an employee using a booking ID and an equipment management system. Here's a step-by-step breakdown of what each part of the code does:

1. QR Code Scanning:

- The `get_employee_id_from_qr` function uses OpenCV to open the computer's camera and scan for QR codes. When a QR code is detected, it decodes the data and returns it as the employee ID.

2. API Request Functions:

- `get_id_from_api` takes an API endpoint and a booking ID, constructs a URL, sends a GET request to the API, and retrieves the employee data associated with the booking ID.

- `update_equipment_quantity` updates the quantity of equipment in the database. It first fetches the current quantity from the API, increments it by one, and then sends a POST request to update the quantity in the database.

3. Check-In Process:

- `make_request` is the main function that coordinates the check-in process. It does the following:

  - Prompts the user to input a booking ID using a simple dialog box.

  - Calls `get_employee_id_from_qr` to scan and retrieve the employee ID from a QR code.

  - Uses `get_id_from_api` to fetch employee and equipment data associated with the booking ID from the API.

  - Compares the scanned employee ID with the one retrieved from the API to ensure they match.

  - If the IDs match, it sends a POST request to update the check-in status in the API with the current date.

  -If an equipment ID is associated with the booking, it calls `update_equipment_quantity` to update the equipment quantity.

<u>4. Tkinter GUI:</u>

- Creates a Tkinter window with a title "Check-In Application".

- Adds a button labeled "Check-In" that triggers the `make_request` function when clicked.

- Runs the Tkinter event loop to display the window and handle user interactions.

The application handles various scenarios with error messages using `messagebox` to inform the user if something goes wrong, such as failing to scan the QR code, no booking ID provided, or mismatched employee IDs.

VIEW REPORT SCRIPT

Python Script Folder (Zuhar Regal 4227160 Individual Assignment\Python Scripts\Employee_ViewReport.py)

```python
import http.client

import json

import tkinter as tk

from tkinter import simpledialog, ttk, messagebox



def fetch_data(api_url):

   # Function to fetch data from the specified API URL

   try:

       # Establishing connection to the API

       conn = http.client.HTTPSConnection("apex.oracle.com")

       conn.request("GET", api_url)

       response = conn.getresponse()


       # Checking if the request was successful

       if response.status == 200:

           data = response.read()

           return json.loads(data.decode('utf-8'))

       else:

           # Displaying error message if request failed

           messagebox.showerror("Error", f"Failed to fetch data from
API. Status: {response.status}")

           return None

   except http.client.HTTPException as http_err:

       # Handling HTTP exceptions
       messagebox.showerror("HTTP Error", f"HTTP Error occurred:
{http_err}")
       return None
   except json.JSONDecodeError as json_err:
       # Handling JSON decoding errors
       messagebox.showerror("JSON Decode Error", f"JSON Decode Error
occurred: {json_err}")
       return None
   except Exception as e:
       # Handling other exceptions
       messagebox.showerror("Error", f"An error occurred while
fetching data: {e}")
       return None



def retrieve_employee_bookings(employee_id):
   # Function to retrieve and display booking data for a given
employee ID
   try:
       # Constructing the API URL to retrieve bookings associated
with the employee ID
       employee_booking_api =
f"https://apex.oracle.com/pls/apex/ifs354_ind_assignment/test/employe
e_id?EMPLOYEE_ID={employee_id}"

       # Fetching booking data for the employee
       employee_booking_data = fetch_data(employee_booking_api)

```

```
        # Handling different scenarios based on the retrieved data

        if employee_booking_data is not None:

            if "items" in employee_booking_data:

                if employee_booking_data["items"]:

                    # Create a new window for displaying the booking
data

                    popup = tk.Tk()

                    popup.title("Bookings")


                    # Create a Treeview widget to display the data in
a table

                    tree = ttk.Treeview(popup)

                    tree["columns"] = ("Equipment ID", "Date Booked",
"Date Returned")

                    tree.heading("#0", text="Booking ID")

                    tree.heading("Equipment ID", text="Equipment ID")

                    tree.heading("Date Booked", text="Date Booked")

                    tree.heading("Date Returned", text="Date
Returned")


                    # Insert data into the tree

                    for booking in employee_booking_data["items"]:

                        if booking.get("employee_id") ==
int(employee_id):

                            booking_id = booking.get("booking_id")

                            equipment_id = booking.get("equipment_id")

                            date_booked = booking.get("date_booked")
```

```python
                            date_returned = booking.get("date_returned")

                            tree.insert("", "end", text=booking_id, values=(equipment_id, date_booked, date_returned))



                    tree.pack(expand=True, fill=tk.BOTH)



                    # Run the main loop of the popup window

                    popup.mainloop()

                else:

                    messagebox.showinfo("Information", f"No bookings found for Employee ID: {employee_id}")

            else:

                messagebox.showerror("Error", "Error fetching booking data.")

        else:

            messagebox.showerror("Error", f"Failed to retrieve data for Employee ID: {employee_id}")

    except Exception as e:

        # Handling other exceptions

        messagebox.showerror("Error", f"An error occurred: {e}")




def employee_functionality():

    # Function to initiate the employee functionality by prompting for employee ID and displaying bookings

    try:
        # Fetching employee ID from user input
        employee_id = simpledialog.askstring("Employee ID", "Enter your Employee ID:")


        if employee_id:
            # Fetching and displaying bookings for the employee
            retrieve_employee_bookings(employee_id)
        else:
            messagebox.showerror("Error", "Please enter your Employee ID.")
    except Exception as e:
        # Handling other exceptions
        messagebox.showerror("Error", f"An error occurred: {e}")



# Call the function to start the employee functionality
employee_functionality()
```

This code is a Tkinter-based Python application that retrieves and displays booking data for an employee based on their Employee ID. Here is a step-by-step explanation of what is happening in the code:

1. `fetch_data(api_url)`:

  - Purpose: Fetches data from a specified API URL.

  - Functionality:

    - Establishes an HTTPS connection to the specified URL.

- Sends a GET request to the API.

  - If the response status is 200 (OK), it reads and decodes the JSON data and returns it.

  - If the request fails, it displays an error message with the status code.

  - Handles HTTP exceptions, JSON decode errors, and other general exceptions by displaying appropriate error messages.

2. `retrieve_employee_bookings(employee_id)`:

- Purpose: Retrieves and displays booking data for a given employee ID.

- Functionality:

  - Constructs an API URL to retrieve bookings associated with the provided employee ID.

  - Calls `fetch_data()` to get the booking data.

  - If the data retrieval is successful and contains booking items:

    - Creates a new Tkinter window (popup) titled "Bookings".

    - Uses a Treeview widget to display the booking data in a table format.

    - Iterates through the booking items and inserts them into the Treeview.

    - Displays the Treeview in the popup window.

  - If no bookings are found, it displays an informational message.

  - Handles other exceptions by displaying error messages.

3. `employee_functionality()`:

- Purpose: Initiates the employee functionality by prompting for an employee ID and displaying bookings.

- Functionality:

  - Prompts the user to enter their Employee ID using a simple dialog box.

- If an Employee ID is provided, it calls `retrieve_employee_bookings()` to fetch and display the bookings.

- If no Employee ID is entered, it displays an error message.

- Handles other exceptions by displaying error messages.

<u>Main Execution:</u>

- The script directly calls `employee_functionality()` to start the process.

- When `employee_functionality()` is called:

  - It prompts the user to enter their Employee ID.

  - It then retrieves and displays the booking data for that Employee ID by calling `retrieve_employee_bookings()`.

<u>Tkinter GUI:</u>

- `employee_functionality()` and `retrieve_employee_bookings()` functions use Tkinter for creating dialog boxes and displaying messages.

- The Treeview widget in `retrieve_employee_bookings()` displays the booking data in a tabular format within a popup window.

<u>Summary:</u>

- This script allows an employee to input their Employee ID.

- It then fetches their booking data from an API.

- If bookings are found, it displays them in a new window using a table.

- It handles various errors and displays appropriate messages to the user.

Configuration Settings and Options

The Oracle APEX TechInnovators Inc. Equipment Tracking System provides a range of configuration settings and options that can be adjusted to suit the application's requirements. These settings are typically configured through the APEX user interface or the Oracle Database.

1. Application Properties

- Application Name: TechInnovators Inc. Equipment Tracking System.
- Application ID: 230221.
- Schema: The database schema, Equipment Tracking System.
- Authentication Scheme: Defines how admins can authenticate
- Authorization Scheme: Determines access control for different parts of the application.
- Globalization: Language and date format settings for internationalisation.

2. User Interface Settings

- Theme: Configuration options for customizing the visual appearance of the application, including themes, colors, and branding elements to align with business preferences or branding guidelines.
- Templates: HTML templates for page layouts, regions, and navigation.
- Navigation: Configuration for menus, tabs, and breadcrumb navigation.
- Custom CSS and JavaScript: For adding custom styling and functionality.

3. Security Settings

- Session State Protection: Prevents URL tampering by checking the validity of session state.
- Page Access Protection: Controls which users can access specific pages.
- HTTPS Enforcement: Ensures that the application is accessed over a secure connection.

4. Database Settings

- SQL Queries and PL/SQL: Custom SQL and PL/SQL code used within the application.
- Database Links: Connections to other databases for data integration.

5. Performance Settings

- Caching: Configures caching of static resources and query results to improve performance.

6. Monitoring and Logging

- Activity Monitoring: Tracks user activity and application usage.
- Error Logging: Captures and logs application errors for debugging purposes.
- Performance Metrics: Monitors application performance and identifies bottlenecks.

7.Scalability and Extensibility

- Ability to handle growing inventory and user base
- The overall system takes into account modular design for future enhancements

The Python-based application requires configuration settings for the application environment, API interactions, and database connections. These settings can be managed using environment variables, or directly within the code.

1. Environment Configuration

- Environment Variables: Used to store sensitive information like API keys, database credentials, and configuration settings.

2. API Settings

- Endpoints: Specific API endpoints for different operations (e.g., booking, equipment) can be modified for future modifications.
- Headers: Custom headers for API requests (e.g., Content-Type, Authorization).
- Timeouts: Request timeout settings to handle slow responses or timeouts.

3. Logging and Monitoring

- Error Handling: Configuration for capturing and reporting errors, including error message popups.

4. Security Settings

- Encryption Keys: Keys for encrypting sensitive data.
- Session Management: Configuration for managing user sessions securely.

User Manual and Guides

End User Documentation

(Available in documentation folder (Zuhar Regal 4227160 Individual Assignment\TechInnovators User Manual Guide.pdf))

Administrator Guides

Disaster Recovery and Backup

Database Backup:
- Schedule regular backups of the database to ensure data is consistently saved.
- Ensure backup files are securely stored and can be easily restored when needed.

Python Backup:
- Ensure backups of code are readily available.

Monitoring and Performance

- Oracle Enterprise Manager:Employ Oracle Enterprise Manager to monitor and manage Oracle environments efficiently.

Monitoring and Troubleshooting Steps

- Perform regular checks on application and database records to identify any anomalies or errors.
- Perform regular system health checks, including CPU, memory, disk usage, and network performance, to proactively identify and address potential issues.

Troubleshooting Guide

Network Connectivity and Firewall:

- Ensure that your network connection is active and stable.

API Endpoints and Credentials:

- Confirm that the API endpoints specified in python file are correct and accessible.
- Verify that the API credentials provided are accurate and have the necessary permissions.
- Confirm that your internet connection is working properly. Try accessing other websites or services to verify connectivity.
- Double-check the API endpoints and access credentials in your configuration file to ensure they are correct and up-to-date, using postman

Database Server Status:

- Check that the database server is running and accessible.
- Ensure that there are no connectivity issues between your application and the database server.

<u>Application Launch Issues</u>

Python and Libraries:

- Verify that all required libraries and dependencies are installed before running scripts.

<u>Webcam Functionality</u>

Webcam Installation:

- Make sure your webcam is properly installed and connected to your computer with python.

<u>Persistent Issues</u>

- If you continue to experience problems after following these steps, please contact support at [4227160@myuwc.ac.za] for further assistance. Provide as much detail as possible about the issue and any troubleshooting steps you have already taken.

<u>Overall Solution Benefits</u>

<u>Scalability and Extensibility</u>

Scalability:

Database Scalability: Oracle Database can handle large volumes of data and can be scaled vertically or horizontally.
Application Scalability: Oracle APEX and Python applications can be deployed on cloud infrastructure with auto-scaling capabilities.
API Scalability: RESTful APIs can be hosted on scalable cloud services, managed by API gateways.

Extensibility:

Modular Design:
- The system is designed with modular components (Oracle APEX for admin and inventory management, Python for employee interactions and complex logic). This allows for easy addition of new features or modifications without affecting the entire system.
- New modules or functionalities (e.g., new types of reports, integration with third-party services) can be added as separate components that interact with existing ones through well-defined APIs.

API-Driven Architecture: New applications and systems can integrate seamlessly through RESTful APIs.

- The use of RESTful APIs facilitates integration with other systems and applications. New applications (e.g., mobile apps, third-party systems) can easily interact with the existing system via these APIs.
- Future enhancements, such as adding new endpoints or modifying existing ones, can be done independently without disrupting current functionalities.

<u>Data Integrity and Validation</u>

Database Constraints:
- Primary Keys: Ensure unique identification of records in tables (e.g., booking_id in BOOKING table, equipment_id in EQUIPMENT table).
- Foreign Keys: Maintain referential integrity between related tables (e.g., employee_id in BOOKING references EMPLOYEE, equipment_id in LOCATION references EQUIPMENT).
- Not Null Constraints: Ensure that essential fields (e.g., booking_id, employee_id, equipment_id) always have valid data.

Validation in Oracle APEX:
- Built-in validation features in Oracle APEX ensure that data entered by users is correct and consistent. This includes form validation, data type checks, and custom validation rules.
- Triggers and stored procedures can enforce complex business rules and ensure data consistency across multiple tables.

Validation in Python Application:
- Input validation: Ensure that user inputs (e.g., Booking ID, Employee ID) are correct and prevent SQL injection attacks.
- API validation: The RESTful API endpoints can perform additional checks to ensure data integrity before processing requests (e.g., checking if the equipment is available before allowing a booking).

Efficient Workflows

Optimized Data Flow:

- Real-Time Updates: Changes in equipment status (e.g., check-out/check-in) are updated in real-time in the Oracle APEX database, ensuring that data is always current and accurate.
- Efficient API Calls: The RESTful API ensures that data retrieval and updates between the Python application and the Oracle APEX backend are efficient and secure. Batch processing can be employed for bulk operations to reduce the number of API calls.

User-Friendly Interfaces:

- Oracle APEX: Provides intuitive and responsive web interfaces for admin tasks such as adding equipment, managing bookings, and generating reports.
- Python Application: Offers simple and effective interfaces for employees to view their equipment usage and perform check-in/check-out operations using QR codes for quick and accurate input.

The proposed solution ensures scalability by using cloud infrastructure, modular design, and API-driven architecture. Extensibility is achieved through the separation of concerns and the ability to add new components without disrupting existing ones. Data integrity and validation are maintained through database constraints, application-level validations, and secure API interactions. Efficient workflows are ensured through real-time updates, user-friendly interfaces, and automated processes. This comprehensive approach provides a functional and future-proof equipment tracking system for TechInnovators Inc.

END