Industrial Training Project Report
On

INVENTORY MANAGEMENT SYSTEM

Submitted to

HINDALCO Industries Limited,
Muri, Jharkhand - 835101

BY

RISHAV PRASAD
Roll No. - 2105818
Branch - Computer Science and Engineering
College - Kalinga Institute of Industrial Technology
Bhubaneswar, Odisha - 751024

Training Period - 6st May, 2024 - 5th June, 2024

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)
Deemed to be University U/S 3 of UGC Act, 1956

CERTIFICATE

This is certify that the project entitled "**Inventory Management System**" submitted by **Rishav Prasad** of KIIT University, Bhubaneswar is a record of bonafide work carried out by him, in the partial fulfillment of the requirement for the award of Industrial Training Completion Certificate at HINDALCO INDUSTRIES LIMITED, Muri Works. This work is done during 6th May, 2024 - 5th June, 2024, under my guidance.

Under the Guidance of

MAHTAB ALAM
SENIOR OFFICER, SYSTEM
HINDALCO INDUSTRIES LTD.
PROJECT INCHARGE

ATUL KUMAR
HEAD, IT Department
HINDALCO INDUSTRIES LTD.

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to all the individuals who played a crucial role in the successful completion of the Inventory Management System project during my industrial training. Their guidance, support, and contributions were invaluable throughout the entire development process.

First and foremost, I would like to extend my deepest appreciation to **Mr. Atul Kumar**, my mentor, whose expertise and guidance were instrumental in shaping the project. His insightful feedback, constant encouragement, and extensive knowledge of inventory management greatly contributed to the system's development and implementation. His dedication and support have been truly invaluable.

I would also like to extend my heartfelt thanks to **Mr. Mahtab Alam**, **Mr. Kumar Sudhanshu**, **Mr. Timothy Murmu and the** entire staff of Hindalco Industries Ltd's IT department for their continuous assistance, technical insights, and valuable suggestions throughout the project. Their expertise and collaboration significantly enhanced the functionality and efficiency of the Inventory Management System. Their contributions and willingness to share their knowledge have been instrumental in overcoming challenges and achieving the project's objectives.

I would also like to express my gratitude to **Mrs. Angela D'Vaz** for giving me this wonderful opportunity. Their prompt responses to queries and their efforts in organizing the necessary resources and logistics were greatly appreciated.

Lastly, I would like to thank my college and the training coordinators for providing me with the opportunity to undertake this industrial training and gain valuable hands-on experience in software development.

I am truly grateful to all these individuals for their unwavering support, guidance, and encouragement throughout this journey. Their contributions have not only enriched my technical skills but have also shaped me both personally and professionally.

Rishav Prasad
2105818
CSE
KIIT University

ABSTRACT

The Inventory Management System is a web-based application developed for the IT department of Hindalco Industries Ltd, Muri Works. The system aims to streamline and optimize the management of two systems, namely P&B and CapEx, By leveraging technologies such as HTML, CSS, JavaScript, Oracle Application Express, PL/SQL and MySQL, the system provides authorized users with read and write access, authenticated users with read access only, and restricts unauthenticated users from accessing the system.

This project report presents an in-depth analysis of the development and implementation of the Inventory Management System. It outlines the project objectives, provides an overview of the technical stack used, and highlights the key features of the system, including the login feature with differentiated access levels. The report also discusses the system's architecture, implementation details, and the evaluation of the project's outcomes.

Through this project, the IT department at Hindalco Industries Ltd aims to enhance inventory management processes, improve efficiency, and ensure accurate tracking and control of inventory across the P&B and CapEx systems. The project's successful implementation will contribute to streamlined operations, reduced manual efforts, and improved decision-making capabilities.

Keywords: Inventory Management System, P&B, CapEx, HTML, CSS, JavaScript, Oracle Application Express, PL/SQL, MySQL.

CONTENTS

1	Introduction		1
2	Software Requirement Specification (SRS)		2
3	Diagrams		4
	3.1	Use Case Diagram	
	3.2	Entity Relationship (ER) Diagram	
	3.3	Control Flow Diagram	
4	Source Code		6
5	Conclusion		71
6	Future Scope		72
7	References		73

INTRODUCTION

This section provides an overview of the project and sets the context for the development of Inventory Management System for the IT department at Hindalco Industries Ltd. This section also outlines the purpose of the project, the scope of work, and the significance of effective inventory management inthe IT department.

1.1 **Project Overview**: The Inventory Management System project aims to develop a web-based application to streamline the management of two systems within Hindalco Industries Ltd's IT department: P&B (Planning & Budgeting) and CapEx (Capital Expenditure). The system leverages a technology stack consisting of HTML, CSS, JavaScript, PL/SQL, and MySQL to provide efficient tracking, control, and reporting of inventory-related activities.

1.2 **Purpose**: The purpose of this project is to address the challenges faced by the IT department at Hindalco Industries Ltd in managing inventory across various systems. The Inventory Management System will provide authorized users with read and write access, authenticated users with read access only, and restrict unauthenticated users from accessing the system. By implementing this system, the IT department aims to enhance inventory control, reduce manual efforts, and ensure accurate and efficient inventory management.

1.3 **Scope**: The scope of the project is focused on the development of a web application specifically tailored for the IT department's inventory management needs. The system will allow users to perform tasks such as adding and updating inventory items, generating reports, and tracking inventory movement. The project encompasses the design, development, testing, and implementation of the Inventory Management System.

1.4 **Significance**: Effective inventory management is crucial for the smooth functioning of any organization, and it holds particular importance in the IT department. Proper inventory control ensures the availability of necessary equipment, reduces operational costs, minimizes downtime, and enables efficient planning and budgeting. The Inventory Management System will enable the IT department at Hindalco Industries Ltd to optimize their inventory management processes, improve productivity, and make informed decisions based on accurate and up-to-date inventory data.

Software Requirement Specification (SRS)

Introduction

Purpose: The purpose of this Software Requirement Specification (SRS) document is to outline the functional and non-functional requirements for the development of the Inventory Management System. The SRS serves as a reference for the development team and stakeholders to understand the system's capabilities, constraints, and desired behavior.

Scope: The scope of the Inventory Management System includes the design, development, testing, and deployment of a web-based application for the IT department at Hindalco Industries Ltd. The system aims to manage inventory across two systems: P&B (Planning & Budgeting) and CapEx (Capital Expenditure).

Definitions, Acronyms, and Abbreviations:
P&B: Planning & Budgeting
CapEx: Capital Expenditure
SRS: Software Requirement Specification

System Overview

Functional Requirements:
The Inventory Management System shall provide the following functionalities:

- User authentication and authorization for different access levels (admin, authenticated users, unauthenticated users)
- Adding and updating inventory items
- Generating reports for inventory status, transactions, and usage
- Tracking inventory movement and location
- Integration with external systems for data synchronization (if applicable)

Non-functional Requirements:
The non-functional requirements for the Inventory Management System include:

- Usability: The system shall have an intuitive and user-friendly interface.
- Performance: The system shall be responsive and handle a significant number of concurrent users without significant delays.
- Security: The system shall implement appropriate security measures to protect user data and prevent unauthorized access.
- Reliability: The system shall be stable and provide accurate inventory information.

- Scalability: The system should be capable of scaling to accommodate future growth and increasing inventory data.
- Compatibility: The system should be compatible with modern web browsers and mobile devices.

System Architecture

High-Level Architecture: The client-side of the application will be implemented using HTML, CSS, and JavaScript. This includes the web interface for the users to interact with the inventory system, while the server-side will be developed using PL/SQL for business logic and MySQL for data storage and retrieval..

Database Design: The database will include tables to store information such as inventory items, users and transactions. The design will ensure proper normalization and enforce data integrity constraints.

User Interfaces

Login Interface: The login interface shall allow users to enter their credentials and authenticate themselves. Different user roles (admin, authenticated users) will have different levels of access within the system.

Inventory Management Interface: The inventory management interface shall provide functionality for adding and updating inventory items. It should allow users to search for items, view details, and perform bulk actions.

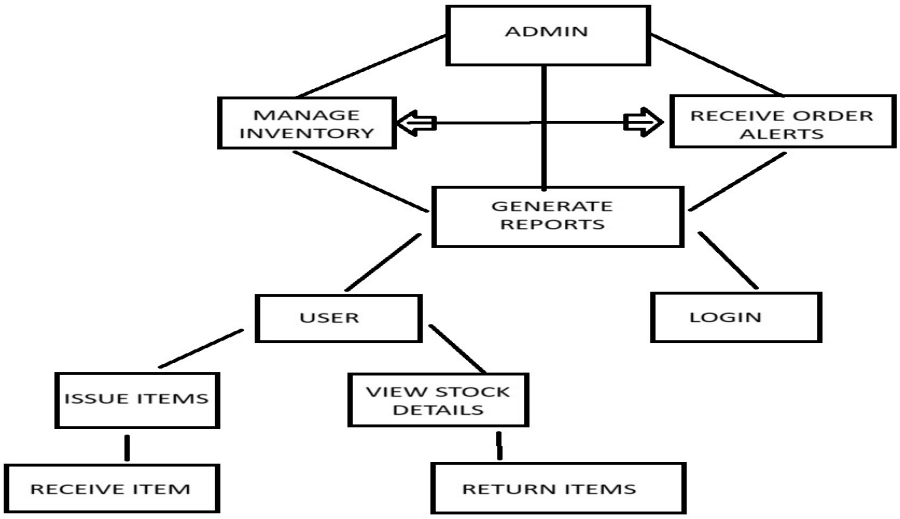
System Constraints

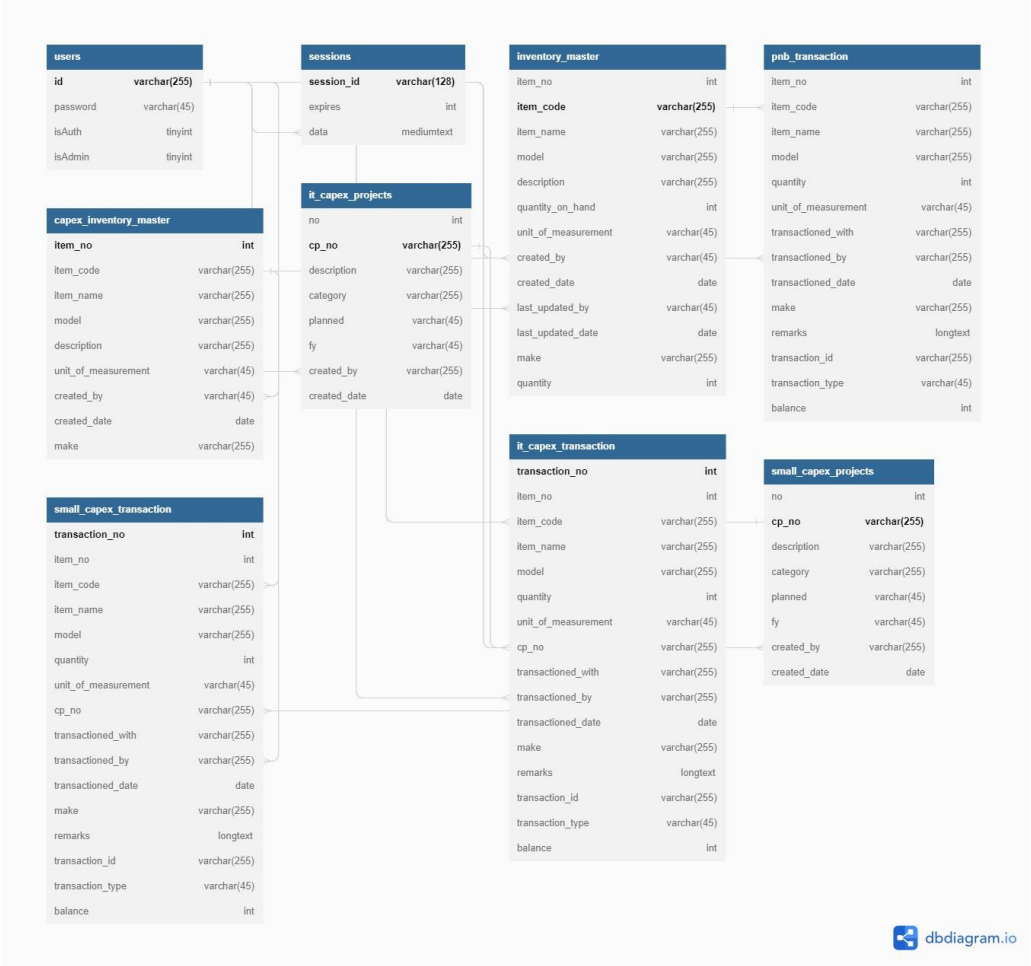
Technology Constraints: The Inventory Management System shall be developed using the following technologies:

- Front-end Technologies: The system must use HTML, CSS, and JavaScript for the client-side.
- Back-end Technologies: The system must use PL/SQL for server-side processing.
- Database: The system must use Oracle Database for data storage and retrieval.

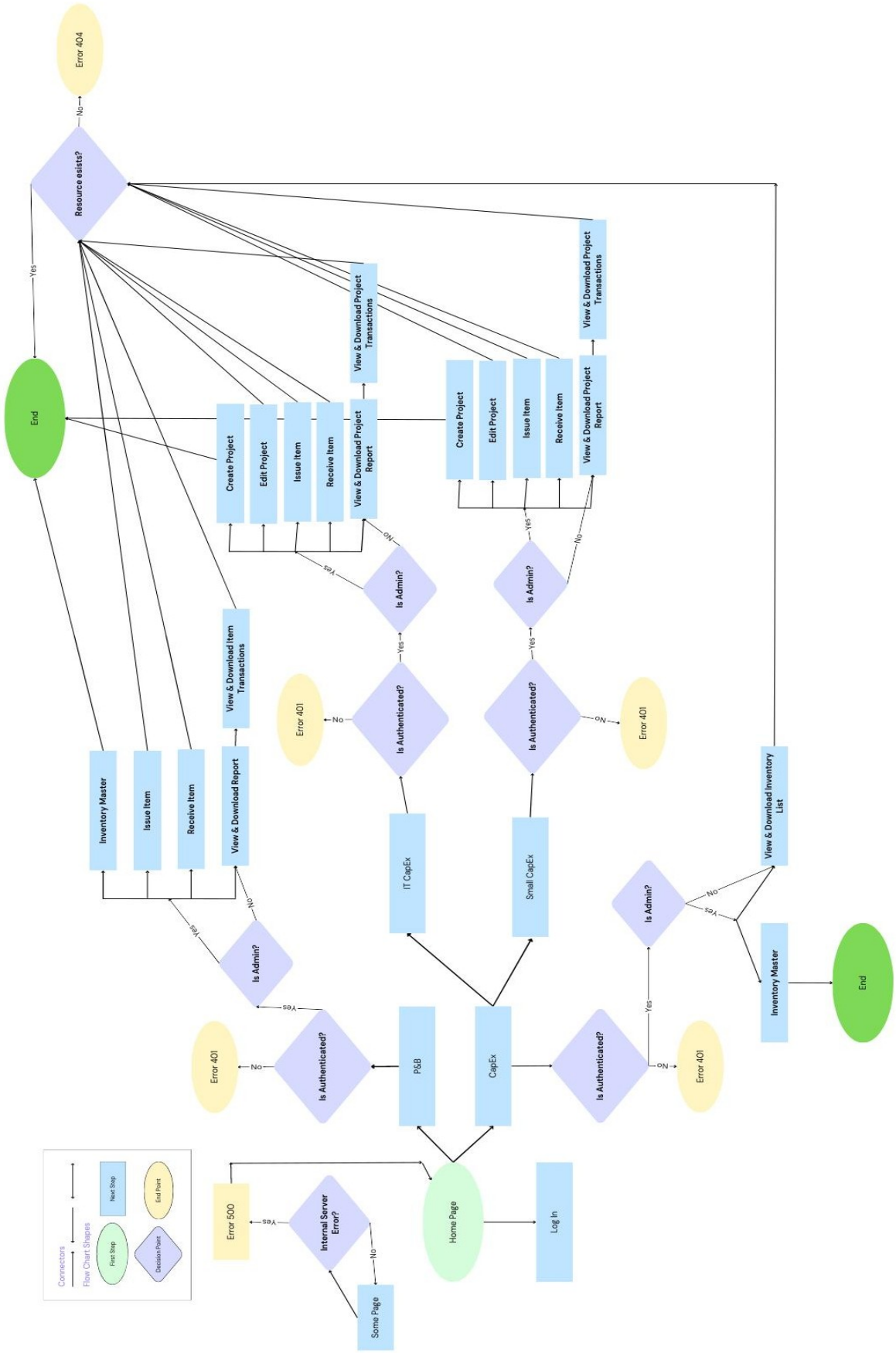
Security Constraints: The system shall implement appropriate security measures, including data encryption, secure user authentication, and access control mechanisms, to ensure the confidentiality and integrity of the system.

Use Case Diagram & Entity Relationship

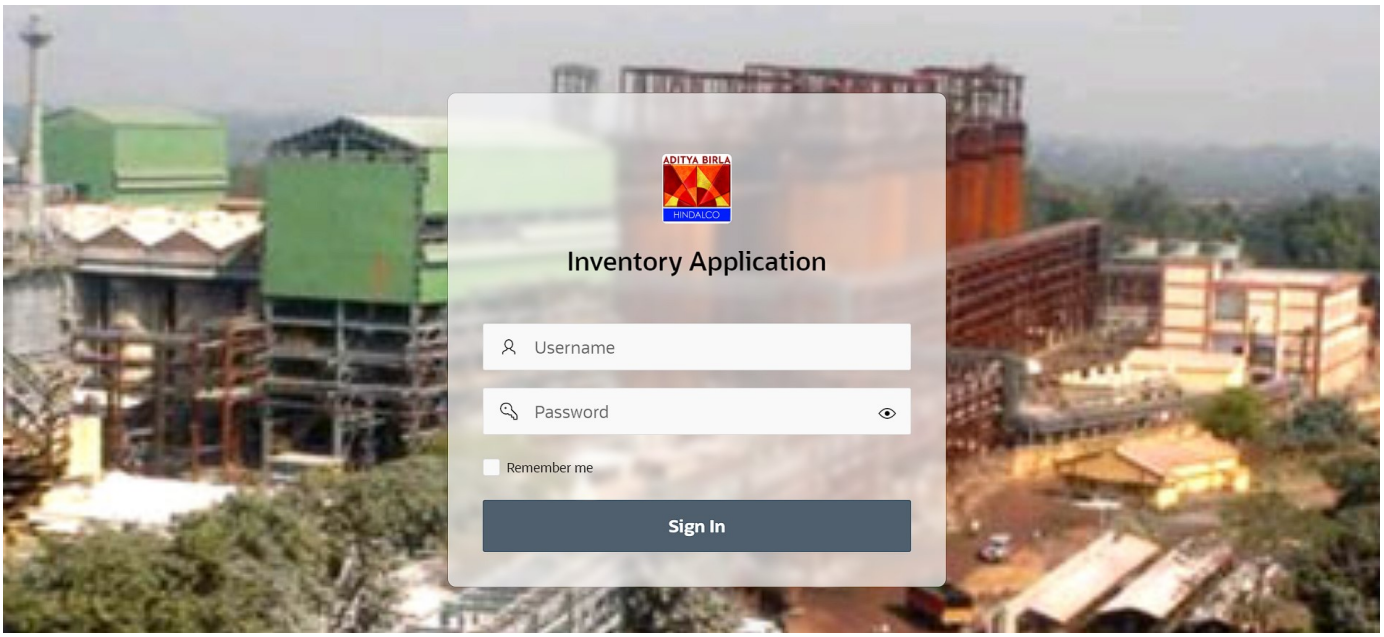




CONTROL FLOW DIAGRAM



Login Page



Login id structure

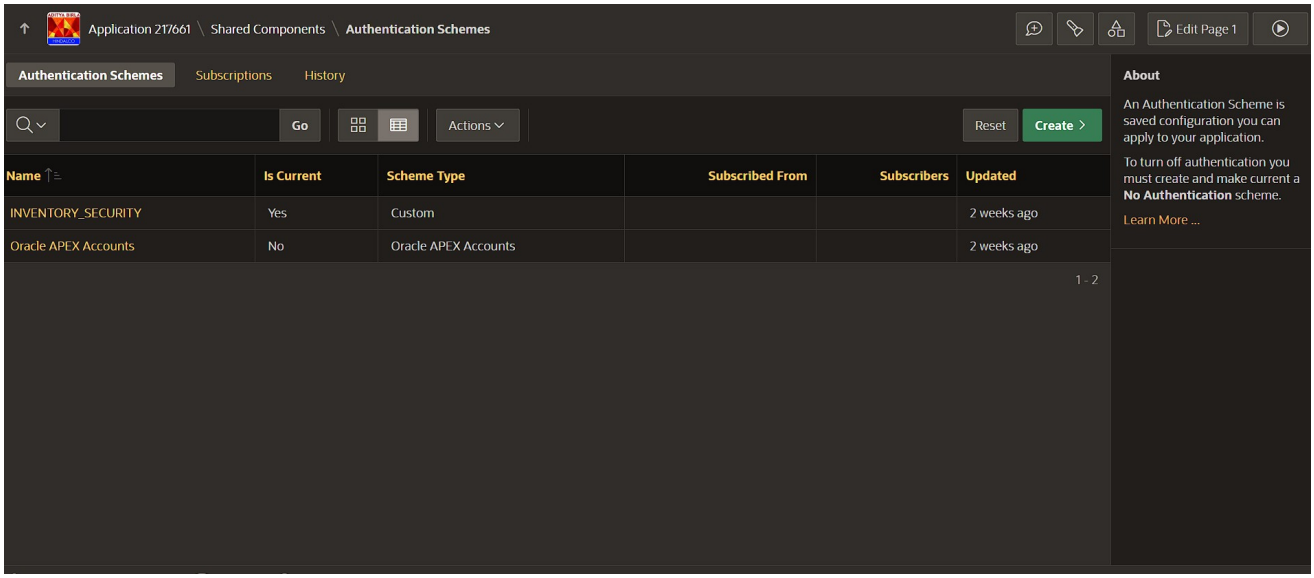
MY_USERS

Columns | Data | Indexes | Constraints | Grants | Statistics | Triggers | Dependencies | DDL | Sample Queries

Download | Refresh

```
1  CREATE TABLE "MY_USERS"
2   (    "ID" NUMBER,
3    "UNAME" VARCHAR2(60),
4    "LOG_IN" VARCHAR2(50),
5    "LOG_PASS" VARCHAR2(60),
6    "STATUS" VARCHAR2(1),
7    "USER_ROLL" VARCHAR2(20),
8    "PASSWORD_HASH" VARCHAR2(100),
9     PRIMARY KEY ("ID")
10 USING INDEX  ENABLE,
11    UNIQUE ("LOG_IN")
12 USING INDEX  ENABLE,
13    UNIQUE ("LOG_PASS")
14 USING INDEX  ENABLE
15  ) ;
```

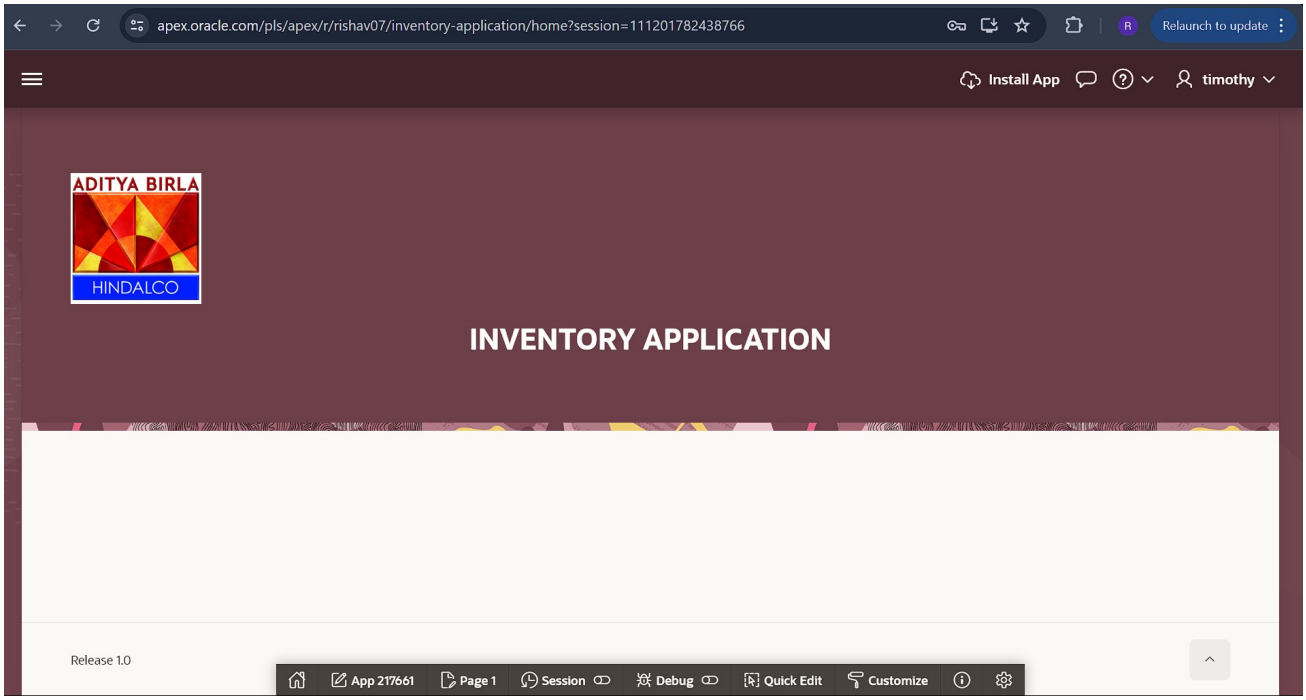
Custom Authentication for login Details



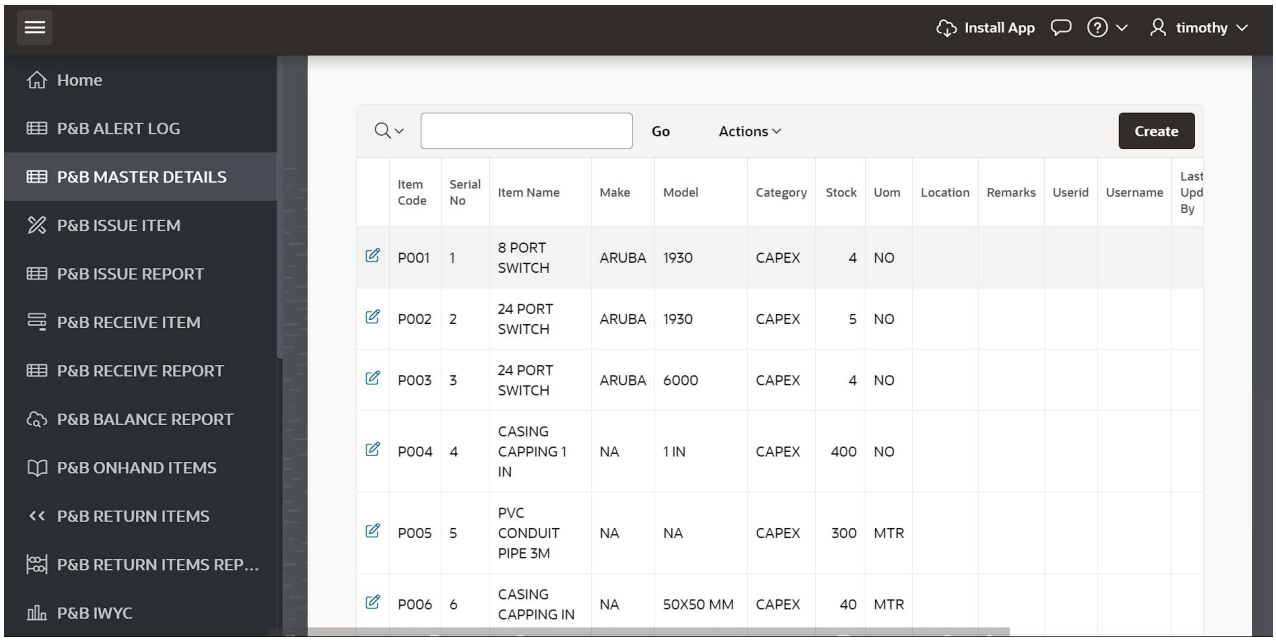
Custom Authentication PL/SQL Code

```
FUNCTION user_aut    (

 p_username IN VARCHAR2,
 p_password VARCHAR2
)
 RETURN BOOLEAN
AS
 lc_pwd_exit VARCHAR2 (1);
BEGIN
 SELECT 'Y'
 INTO lc_pwd_exit
 FROM MY_USERS
 WHERE upper(LOG_IN) = UPPER (p_username) AND LOG_PASS = p_password and
status='Y'
;
RETURN TRUE;
EXCEPTION
 WHEN NO_DATA_FOUND
 THEN
 RETURN FALSE;
END user_aut;
```

INVENTORY MANAGEMENT SYSTEM HOME PAGE



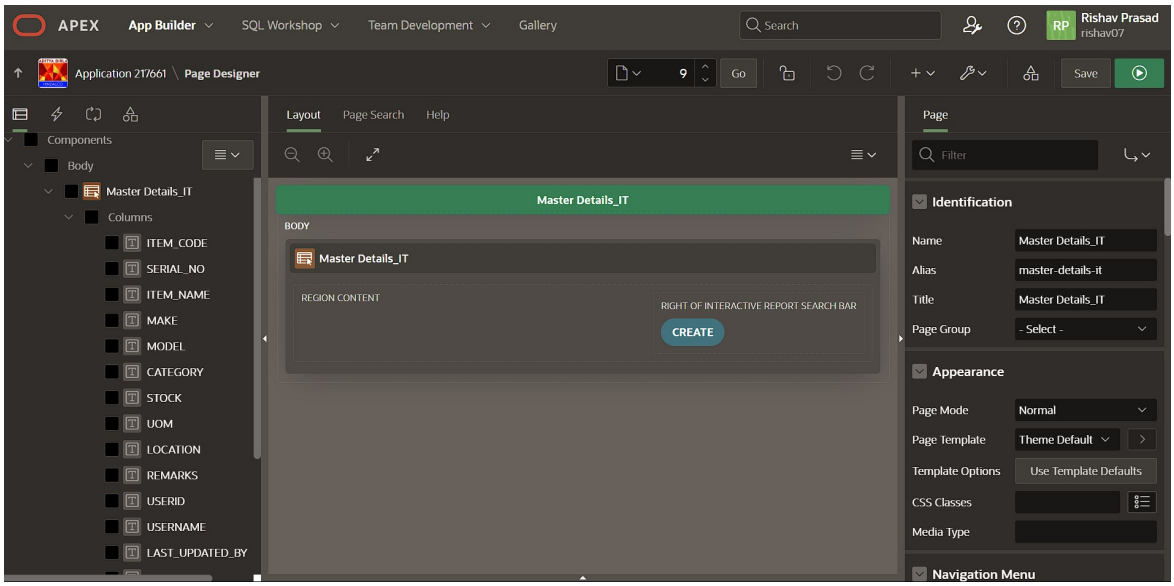
P&B Master Table



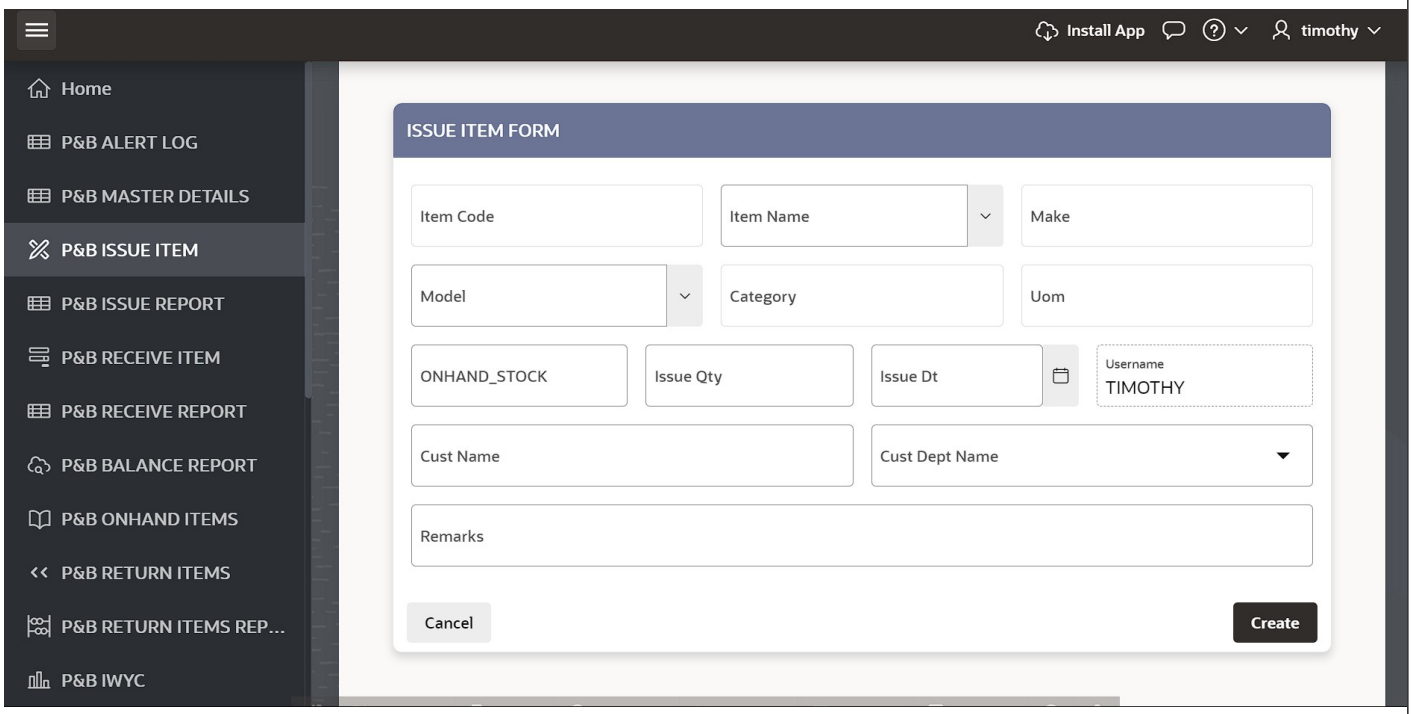
P&B Master Table Structure

```
  CREATE TABLE "IT_ITEMS"
   (  "SERIAL_NO" VARCHAR2(15),
      "ITEM_CODE" VARCHAR2(6),
      "ITEM_NAME" VARCHAR2(50),
      "MAKE" VARCHAR2(15),
      "MODEL" VARCHAR2(30),
      "CATEGORY" VARCHAR2(30),
      "STOCK" NUMBER,
      "UOM" VARCHAR2(6),
      "LOCATION" VARCHAR2(15),
      "REMARKS" VARCHAR2(100),
      "USERID" VARCHAR2(10),
      "USERNAME" VARCHAR2(30),
      "LAST_UPDATED_BY" VARCHAR2(30),
      "LAST_UPDATE_DT" DATE,
      "REORDER_LEVEL" NUMBER DEFAULT 1,
       CONSTRAINT "PK_IT_ITEMS" PRIMARY KEY ("ITEM_CODE")
  USING INDEX  ENABLE
   ) ;

  CREATE OR REPLACE EDITIONABLE TRIGGER "TR_IT_ITEMS_INSERT_SERIAL_NO"
BEFORE INSERT ON IT_ITEMS
FOR EACH ROW
BEGIN
    :NEW.SERIAL_NO := SEQ_IT_ITEMS_SERIAL_NO.NEXTVAL;
END;
/
ALTER TRIGGER "TR_IT_ITEMS_INSERT_SERIAL_NO" ENABLE;
```

P&B Master Table Page



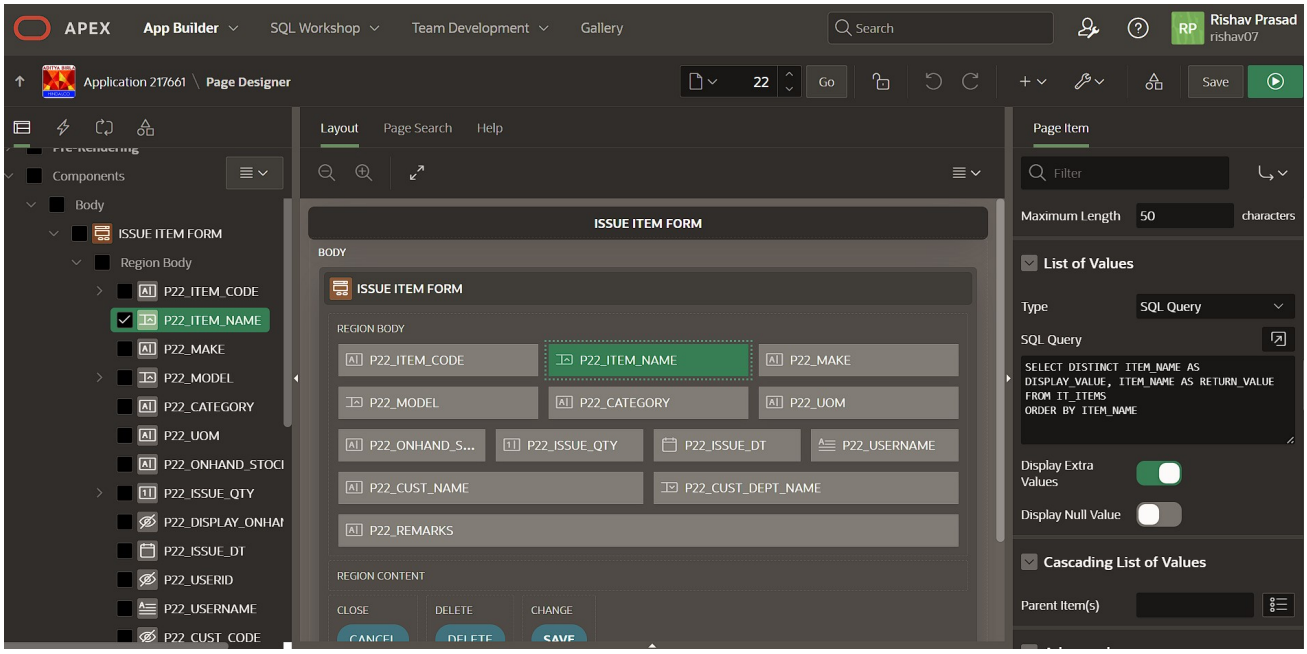
P&B Issue Item Form Page



P&B Issue Item Table Structure

```
CREATE TABLE "ISSUE_ITEMS"
  (  "ITEM_CODE" VARCHAR2(20),
     "ITEM_NAME" VARCHAR2(50),
     "MAKE" VARCHAR2(50),
     "MODEL" VARCHAR2(30),
     "CATEGORY" VARCHAR2(30),
     "UOM" VARCHAR2(50),
     "ISSUE_QTY" NUMBER,
     "ISSUE_DT" DATE,
     "USERID" VARCHAR2(10),
     "USERNAME" VARCHAR2(30),
     "CUST_CODE" VARCHAR2(10),
     "CUST_NAME" VARCHAR2(30),
     "CUST_DEPT_CODE" VARCHAR2(10),
     "CUST_DEPT_NAME" VARCHAR2(30),
     "REMARKS" VARCHAR2(250)
  ) ;
```

P&B Issue Item Page View



P&B Issue Item Dynamic actions Code For fetch details from master table

ITEM_NAME SQL QUERY FOR FETCH DETAILS

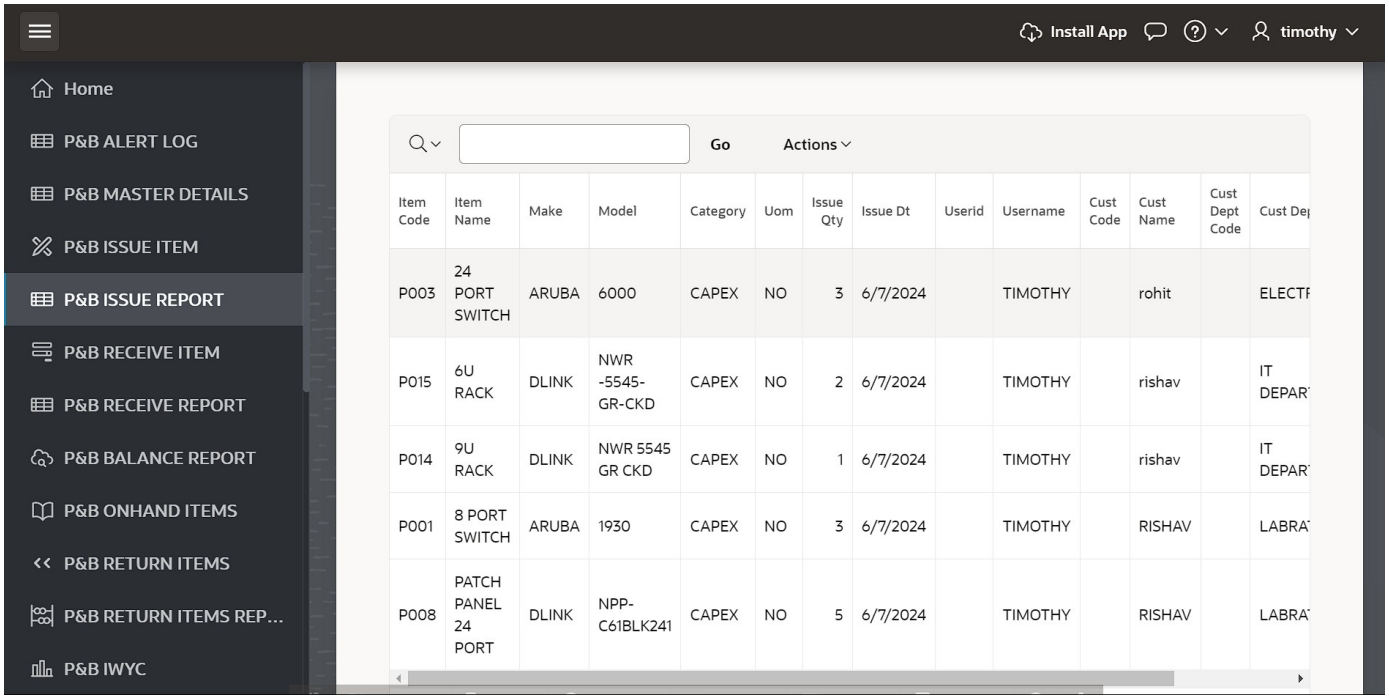
```
SELECT DISTINCT ITEM_NAME AS DISPLAY_VALUE, ITEM_NAME AS RETURN_VALUE
FROM IT_ITEMS
ORDER BY ITEM_NAME
```

ITEM_MODEL SQL QUERY FOR FETCH MODEL DETAILS

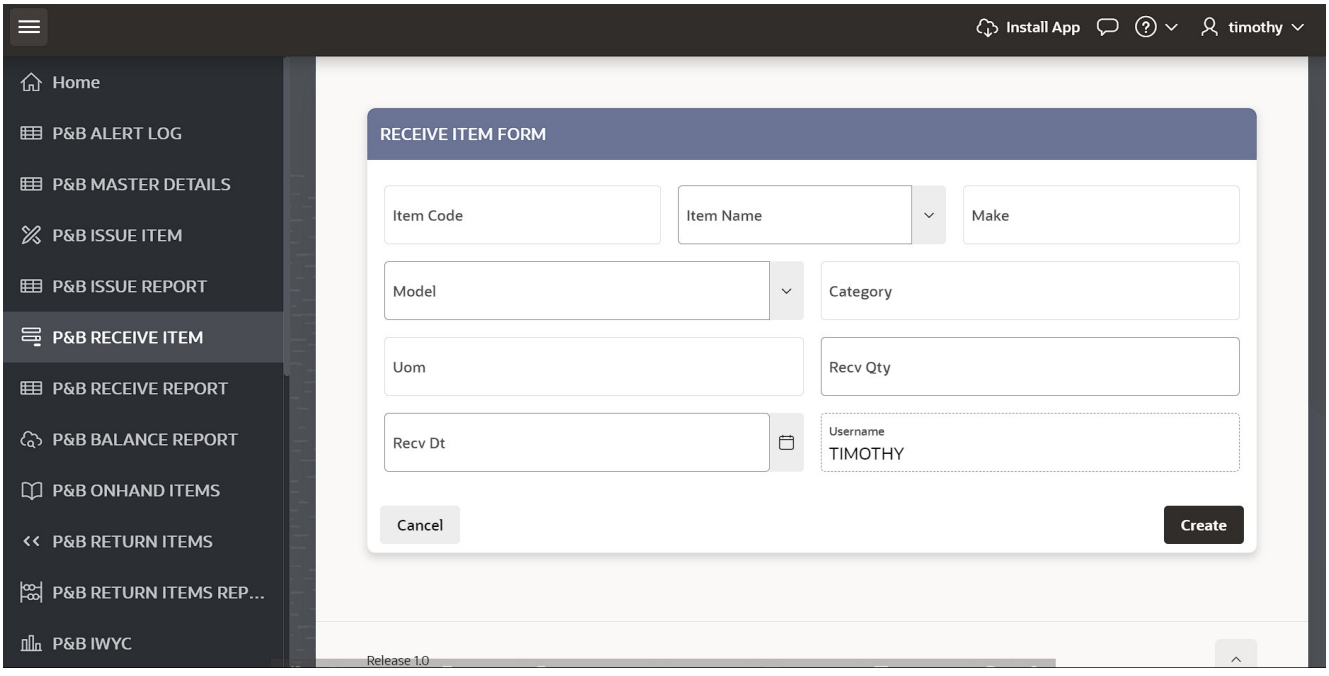
```
SELECT MODEL AS DISPLAY_VALUE, MODEL AS RETURN_VALUE
FROM IT_ITEMS
WHERE ITEM_NAME = :P22_ITEM_NAME
ORDER BY MODEL
```

BY FOLLOWING THIS STEP IT CAN SHOW ITEM CODE, MAKE, CATEGORY, UOM

P&B Issue Item Report Page



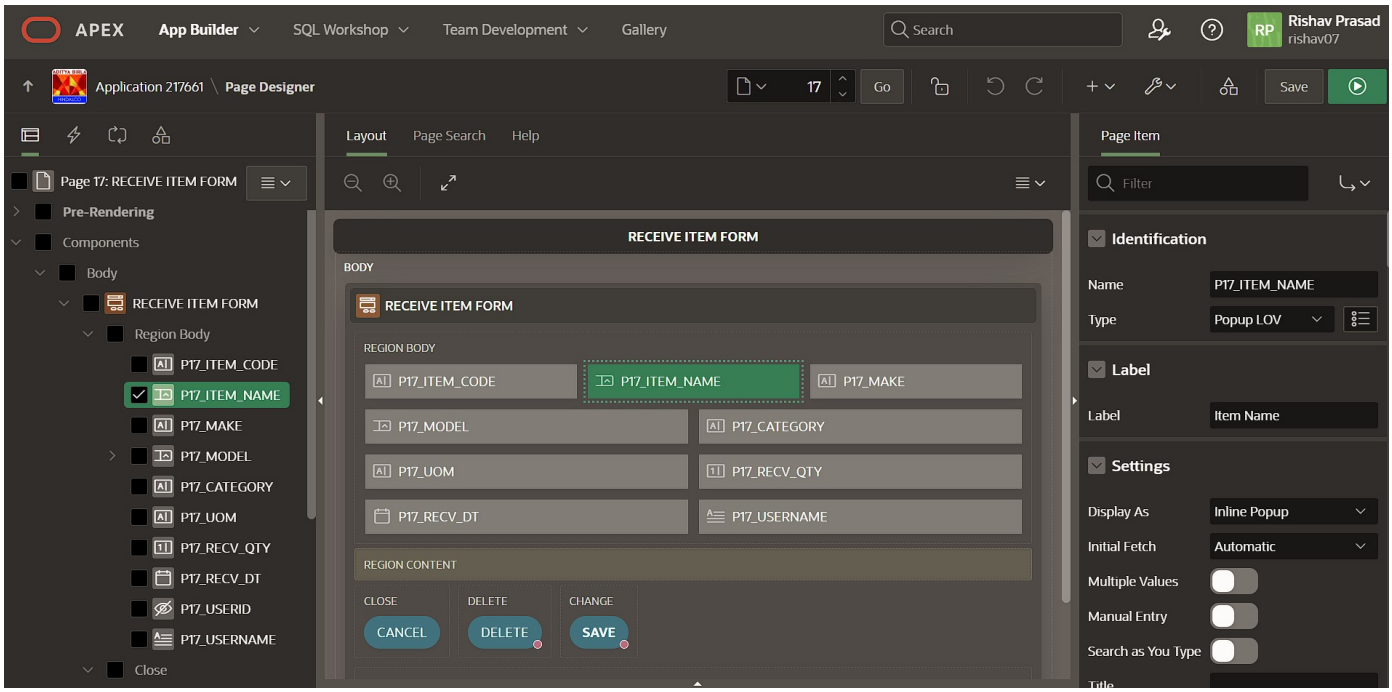
P&B Receive Item Form Page



P&B Receive Item Table Structure

```
CREATE TABLE "RECEIVE_ITEMS"
  (  "ITEM_CODE" VARCHAR2(6),
     "ITEM_NAME" VARCHAR2(50),
     "MAKE" VARCHAR2(15),
     "MODEL" VARCHAR2(30),
     "CATEGORY" VARCHAR2(30),
     "UOM" VARCHAR2(6),
     "RECV_QTY" NUMBER,
     "RECV_DT" DATE,
     "USERID" VARCHAR2(10),
     "USERNAME" VARCHAR2(30)
  ) ;
```

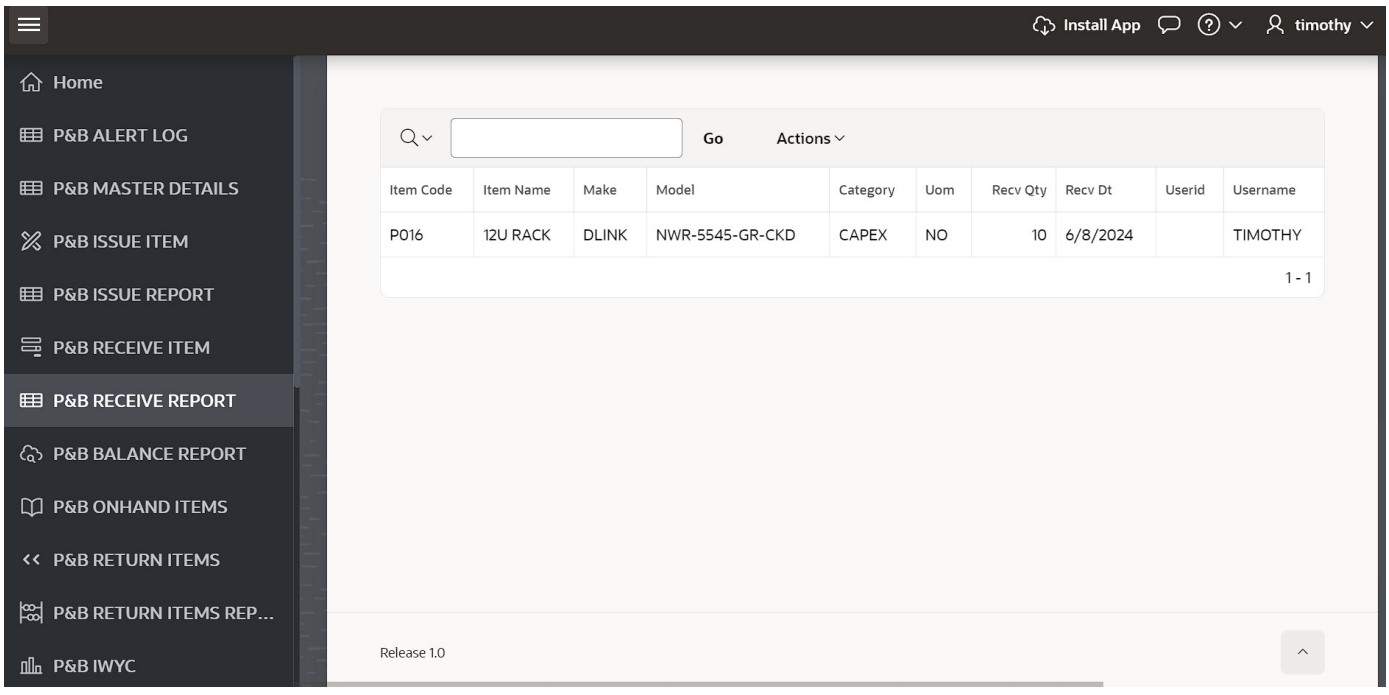
P&B Receive Item Page View



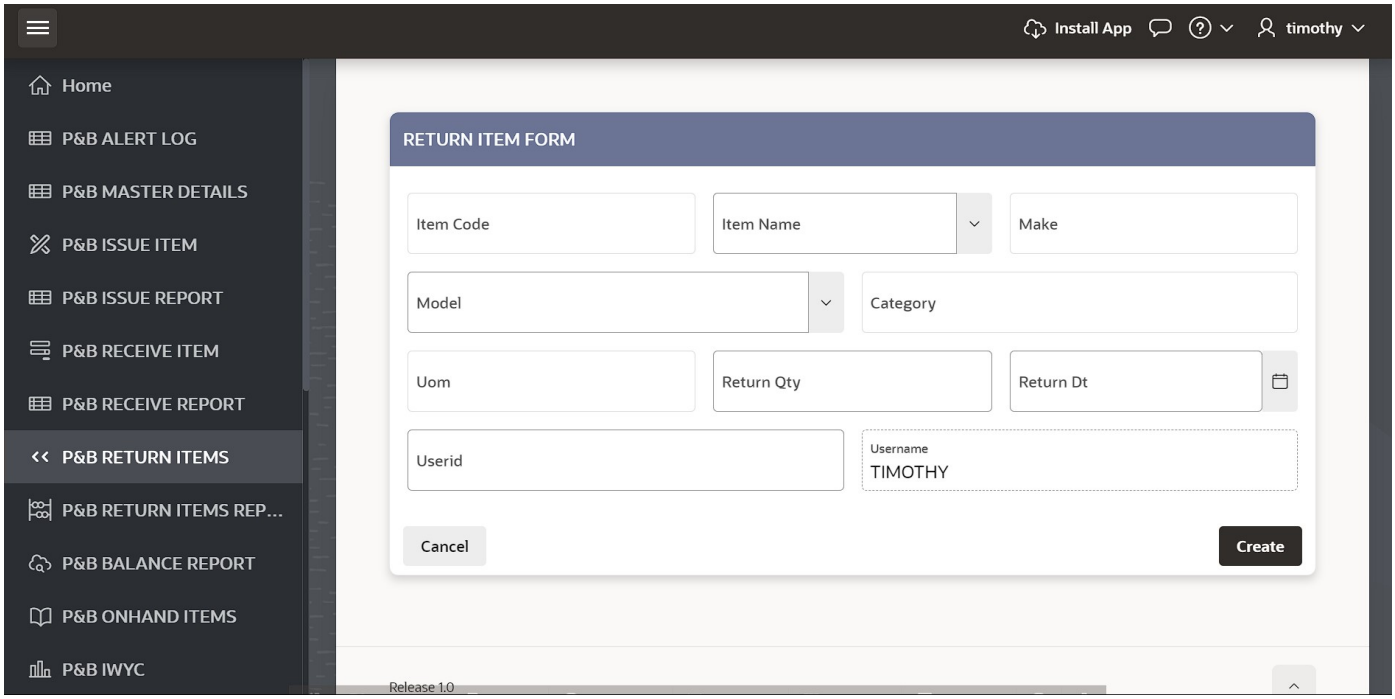
P&B Receive Item Dynamic actions Code For fetch details from master table

```
ITEM_NAME SQL QUERY FOR FETCH DETAILS

SELECT DISTINCT ITEM_NAME AS DISPLAY_VALUE, ITEM_NAME AS RETURN_VALUE
FROM IT_ITEMS
ORDER BY ITEM_NAME



ITEM MODEL SQL QUERY FOR FETCH MODEL DETAILS

SELECT MODEL AS DISPLAY_VALUE, MODEL AS RETURN_VALUE
FROM IT_ITEMS
WHERE ITEM_NAME = :P22_ITEM_NAME
ORDER BY MODEL


BY FOLLOWING THIS STEP IT CAN SHOW ITEM CODE, MAKE, CATEGORY, UOM
```

P&B Receive Item Report Page



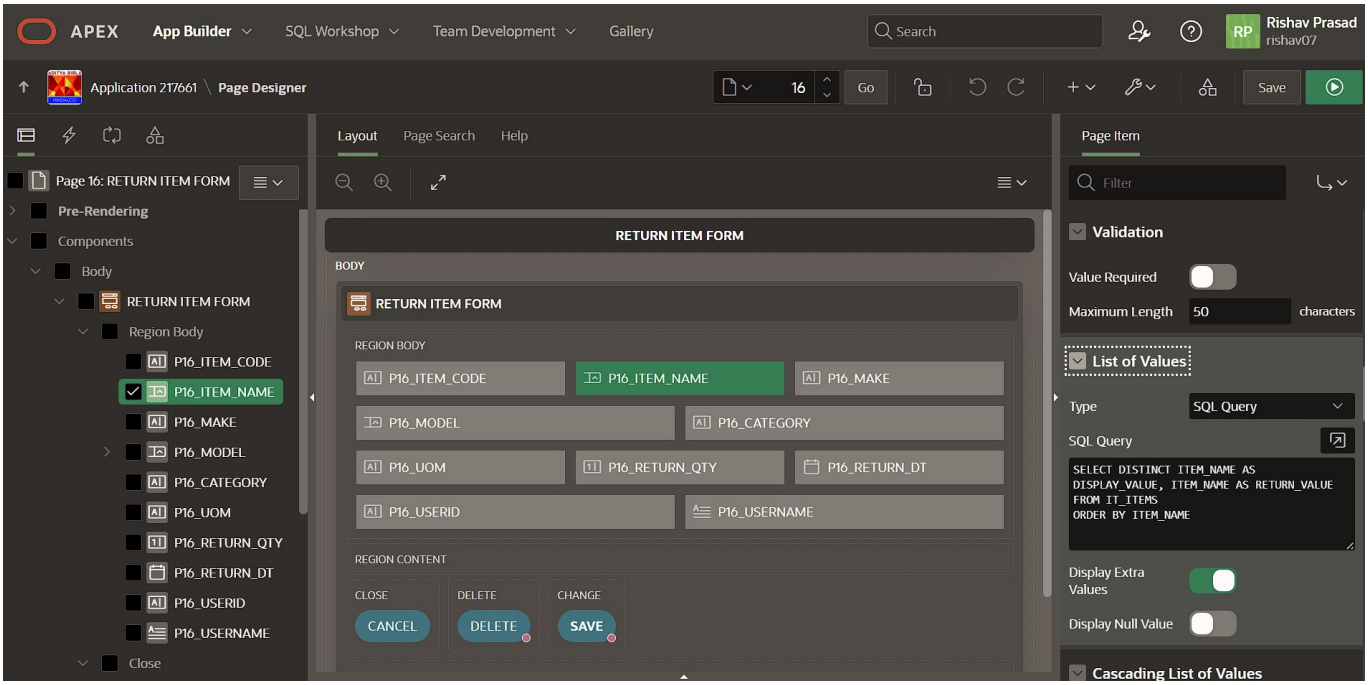
P&B Return Item Form Page



P&B Return Item Table Structure

```
CREATE TABLE "RETURN_ITEMS"
   (  "ITEM_CODE" VARCHAR2(6),
      "ITEM_NAME" VARCHAR2(50),
      "MAKE" VARCHAR2(15),
      "MODEL" VARCHAR2(30),
      "CATEGORY" VARCHAR2(30),
      "UOM" VARCHAR2(6),
      "RETURN_QTY" NUMBER,
      "RETURN_DT" DATE,
      "USERID" VARCHAR2(10),
      "USERNAME" VARCHAR2(30)
   ) ;
```

P&B Return Item Page View



P&B Return Item Dynamic actions Code For fetch details from master table

ITEM_NAME SQL QUERY FOR FETCH DETAILS

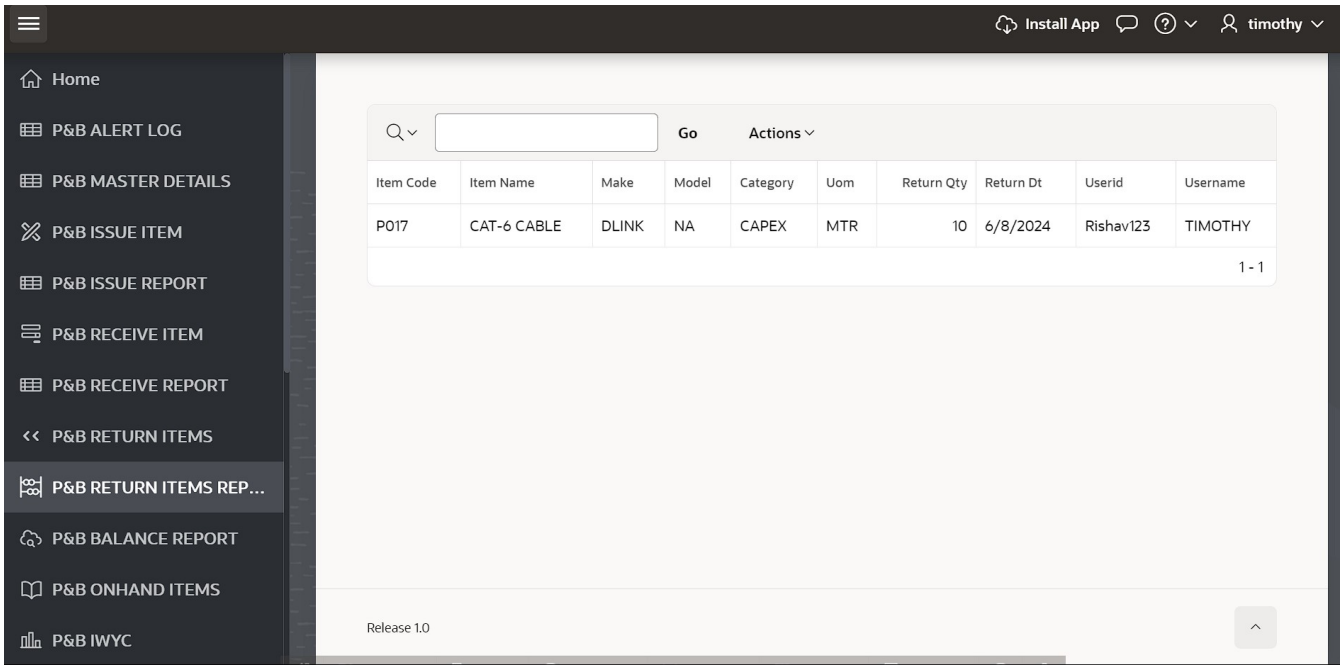
```
SELECT DISTINCT ITEM_NAME AS DISPLAY_VALUE, ITEM_NAME AS RETURN_VALUE
FROM IT_ITEMS
ORDER BY ITEM_NAME
```

ITEM_MODEL SQL QUERY FOR FETCH MODEL DETAILS

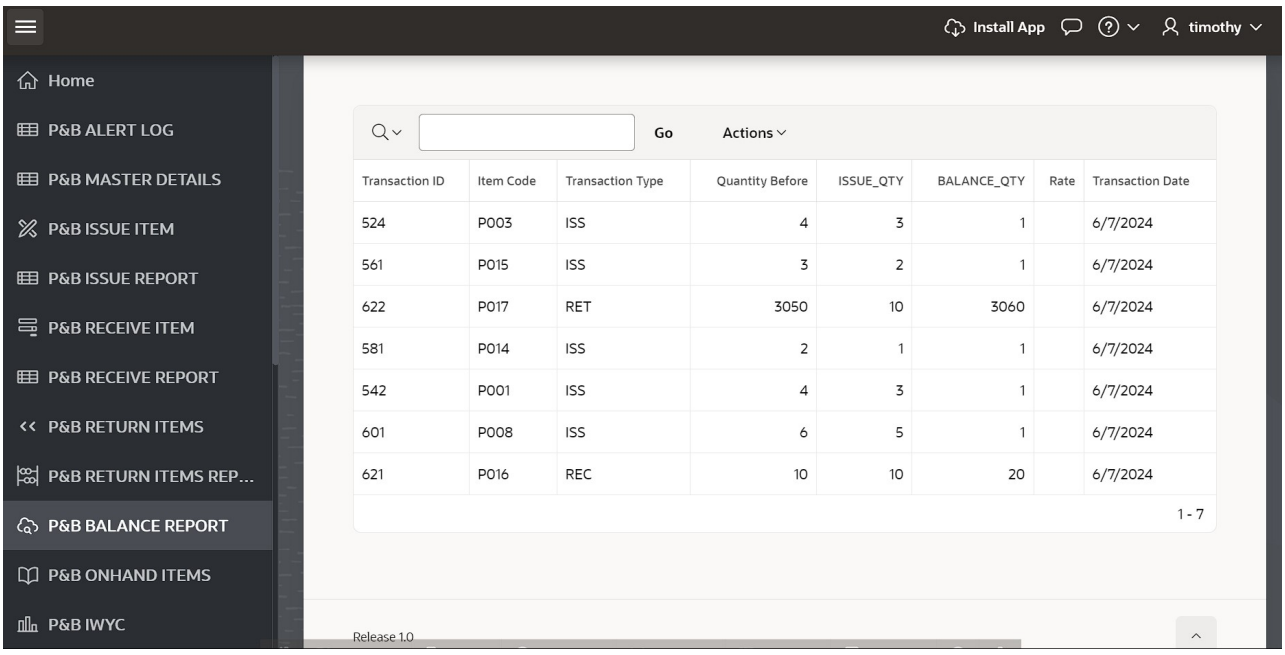
```
SELECT MODEL AS DISPLAY_VALUE, MODEL AS RETURN_VALUE
FROM IT_ITEMS
WHERE ITEM_NAME = :P22_ITEM_NAME
ORDER BY MODEL
```

BY FOLLOWING THIS STEP IT CAN SHOW ITEM CODE, MAKE, CATEGORY, UOM

P&B Return Item Report Page



P&B Balance Report



P&B Triggers for ISS,REC,RET

Issue Item Trigger

```
create or replace TRIGGER TR_ISSUE_ITEMS_UPDATE_OB
AFTER INSERT ON ISSUE_ITEMS
FOR EACH ROW
DECLARE
    l_quantity_before NUMBER;
    l_quantity_after  NUMBER;
BEGIN
    -- Fetch the last quantity after from the OPENING_BALANCE for this item
code
    -- If there is no previous record, fetch the stock from IT_ITEMS
    SELECT COALESCE(
        (SELECT QUANTITY_AFTER
         FROM OPENING_BALANCE
         WHERE ITEM_CODE = :NEW.ITEM_CODE
         ORDER BY TRANSACTION_ID DESC FETCH FIRST 1 ROWS ONLY),
        (SELECT STOCK
         FROM IT_ITEMS
         WHERE ITEM_CODE = :NEW.ITEM_CODE)
    ) INTO l_quantity_before
    FROM DUAL;

    -- Calculate the quantity after the transaction
    l_quantity_after := l_quantity_before - :NEW.ISSUE_QTY;

    -- Insert into OPENING_BALANCE
    INSERT INTO OPENING_BALANCE (
        ITEM_CODE, TRANSACTION_TYPE, QUANTITY_BEFORE, QUANTITY,
QUANTITY_AFTER, RATE
    ) VALUES (
        :NEW.ITEM_CODE, 'ISS', l_quantity_before, :NEW.ISSUE_QTY,
l_quantity_after, NULL
    );
END;
/
```

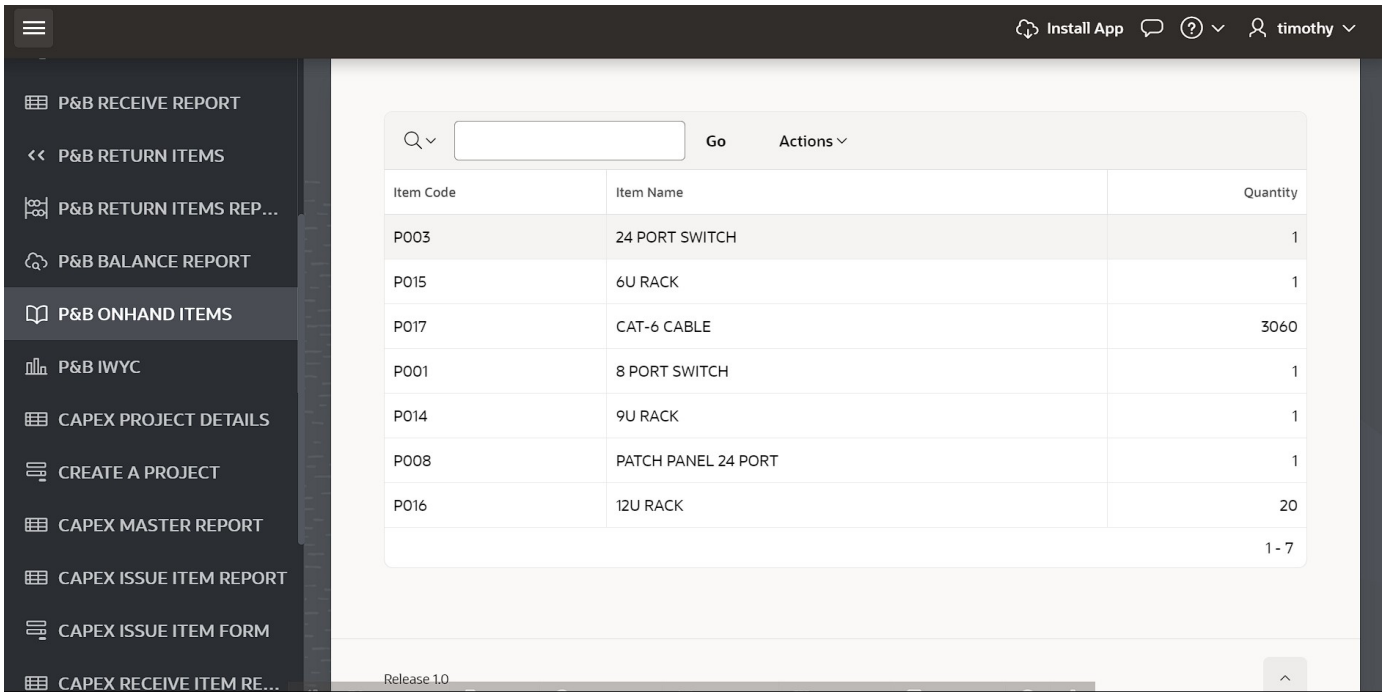
Receive Item Trigger

```
create or replace TRIGGER TR_RECEIVE_ITEMS_UPDATE_OB
AFTER INSERT ON RECEIVE_ITEMS
FOR EACH ROW
DECLARE
    l_quantity_before NUMBER;
    l_quantity_after  NUMBER;
BEGIN
    -- Fetch the last quantity after from the OPENING_BALANCE for this
item code
    -- If there is no previous record, fetch the stock from IT_ITEMS
    SELECT COALESCE(
        (SELECT QUANTITY_AFTER
         FROM OPENING_BALANCE
         WHERE ITEM_CODE = :NEW.ITEM_CODE
         ORDER BY TRANSACTION_ID DESC FETCH FIRST 1 ROWS ONLY),
        (SELECT STOCK
         FROM IT_ITEMS
         WHERE ITEM_CODE = :NEW.ITEM_CODE)
    ) INTO l_quantity_before
    FROM DUAL;

    -- Calculate the quantity after the transaction
    l_quantity_after := l_quantity_before + :NEW.RECV_QTY;

    -- Insert into OPENING_BALANCE
    INSERT INTO OPENING_BALANCE (
        ITEM_CODE, TRANSACTION_TYPE, QUANTITY_BEFORE, QUANTITY,
QUANTITY_AFTER, RATE
    ) VALUES (
        :NEW.ITEM_CODE, 'REC', l_quantity_before, :NEW.RECV_QTY,
l_quantity_after, NULL
    );
END;
/
```

RETURN ITEM TRIGGER

```
create or replace TRIGGER TR_RETURN_ITEMS_UPDATE_OB
AFTER INSERT ON RETURN_ITEMS
FOR EACH ROW
DECLARE
    l_quantity_before NUMBER;
    l_quantity_after  NUMBER;
BEGIN
    -- Fetch the last quantity after from the OPENING_BALANCE for this
item code
    -- If there is no previous record, fetch the stock from IT_ITEMS
    SELECT COALESCE(
        (SELECT QUANTITY_AFTER
         FROM OPENING_BALANCE
         WHERE ITEM_CODE = :NEW.ITEM_CODE
         ORDER BY TRANSACTION_ID DESC FETCH FIRST 1 ROWS ONLY),
        (SELECT STOCK
         FROM IT_ITEMS
         WHERE ITEM_CODE = :NEW.ITEM_CODE)
    ) INTO l_quantity_before
    FROM DUAL;

    -- Calculate the quantity after the return
    l_quantity_after := l_quantity_before + :NEW.RETURN_QTY;

    -- Insert into OPENING_BALANCE
    INSERT INTO OPENING_BALANCE (
        ITEM_CODE, TRANSACTION_TYPE, QUANTITY_BEFORE, QUANTITY,
QUANTITY_AFTER, RATE
    ) VALUES (
        :NEW.ITEM_CODE, 'RET', l_quantity_before, :NEW.RETURN_QTY,
l_quantity_after, NULL
    );

    -- Update the ONHAND_QUANTITY table
    UPDATE ONHAND_QUANTITY
    SET QUANTITY = l_quantity_after
    WHERE ITEM_CODE = :NEW.ITEM_CODE;
END;
/
```

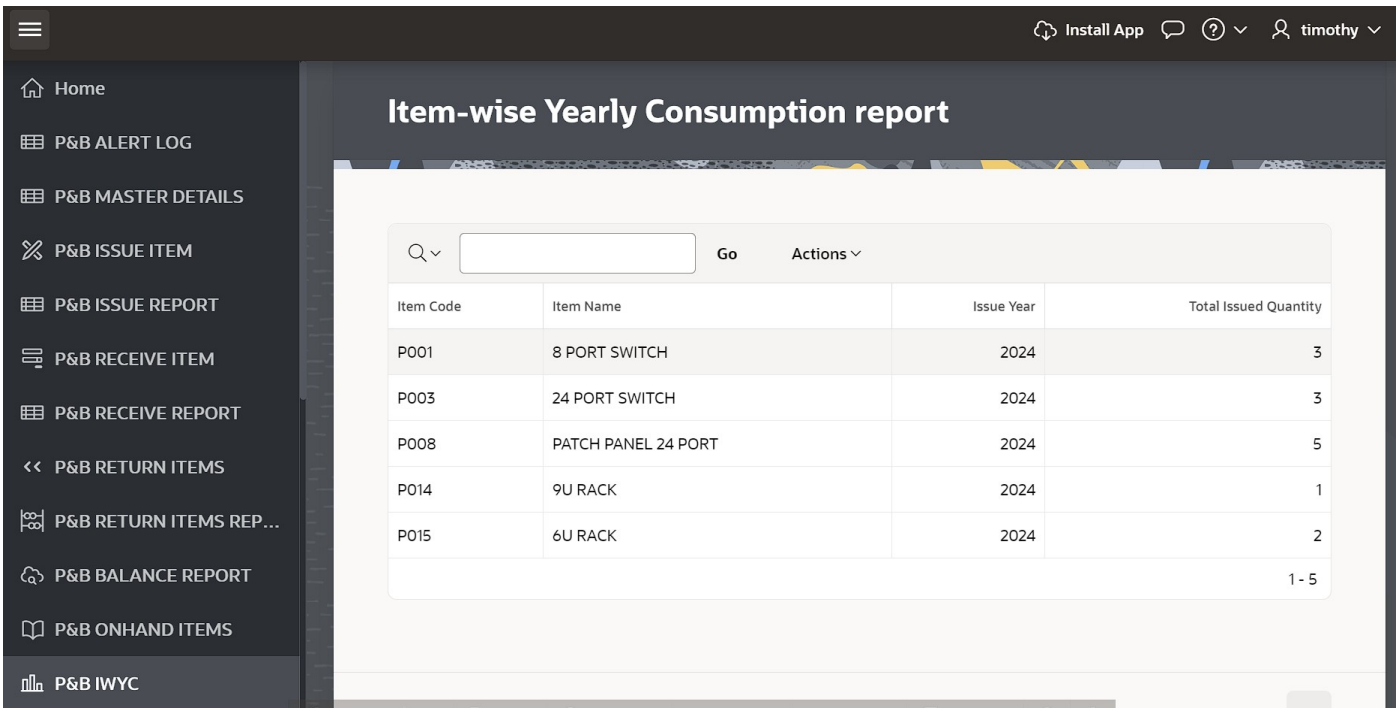
P&B Onhand Items REPORT



P&B Trigger for Onhand Quanity

```
create or replace TRIGGER TR_UPDATE_ONHAND_QUANTITY
AFTER INSERT OR UPDATE ON OPENING_BALANCE
FOR EACH ROW
BEGIN
    -- Ensure the `ITEM_CODE` exists and `QUANTITY_AFTER` is provided
    IF :NEW.ITEM_CODE IS NOT NULL AND :NEW.QUANTITY_AFTER IS NOT NULL THEN
        MERGE INTO ONHAND_QUANTITY oq
        USING (
            SELECT :NEW.ITEM_CODE AS ITEM_CODE,
                   :NEW.QUANTITY_AFTER AS QUANTITY,
                   (SELECT ITEM_NAME FROM IT_ITEMS WHERE ITEM_CODE
= :NEW.ITEM_CODE) AS ITEM_NAME
            FROM DUAL
        ) src
        ON (oq.ITEM_CODE = src.ITEM_CODE)
        WHEN MATCHED THEN
            UPDATE SET oq.QUANTITY = src.QUANTITY, oq.ITEM_NAME =
src.ITEM_NAME
        WHEN NOT MATCHED THEN
            INSERT (oq.ITEM_CODE, oq.ITEM_NAME, oq.QUANTITY)
            VALUES (src.ITEM_CODE, src.ITEM_NAME, src.QUANTITY);
    END IF;
END;
/
```

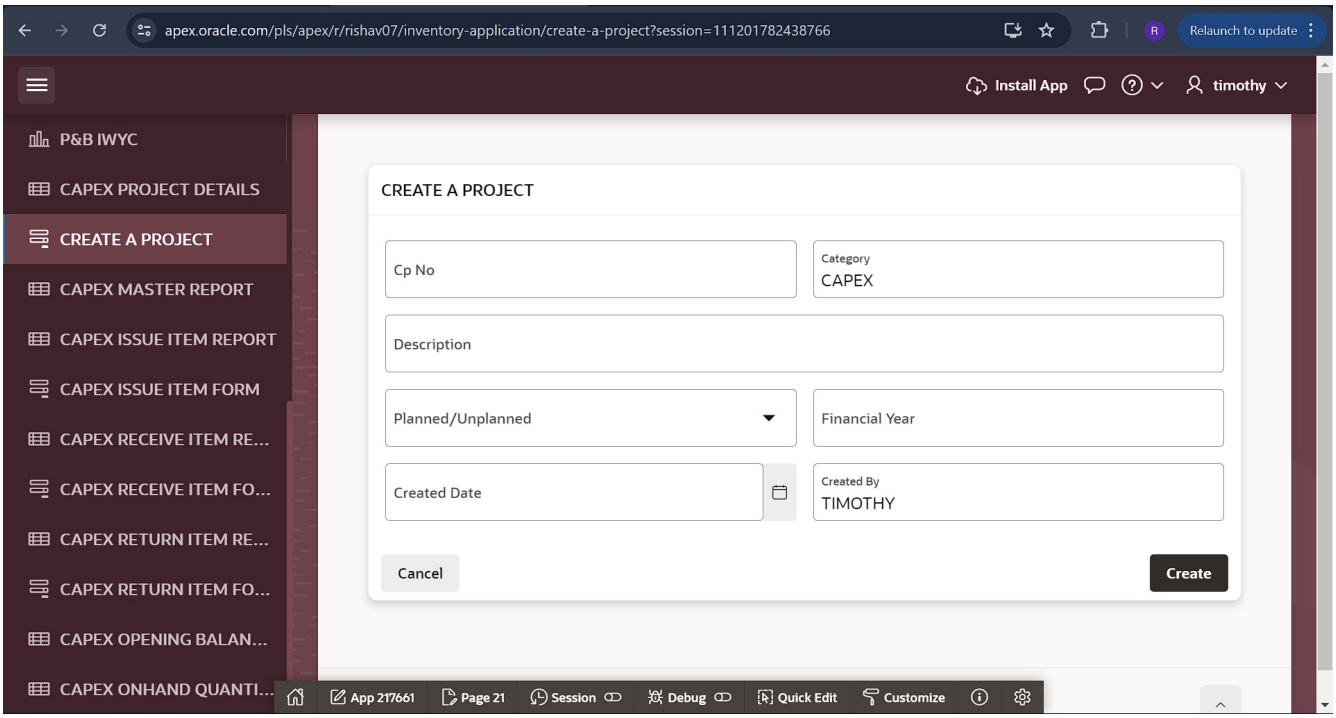
P&B Item wise Yearly Cosumption Report



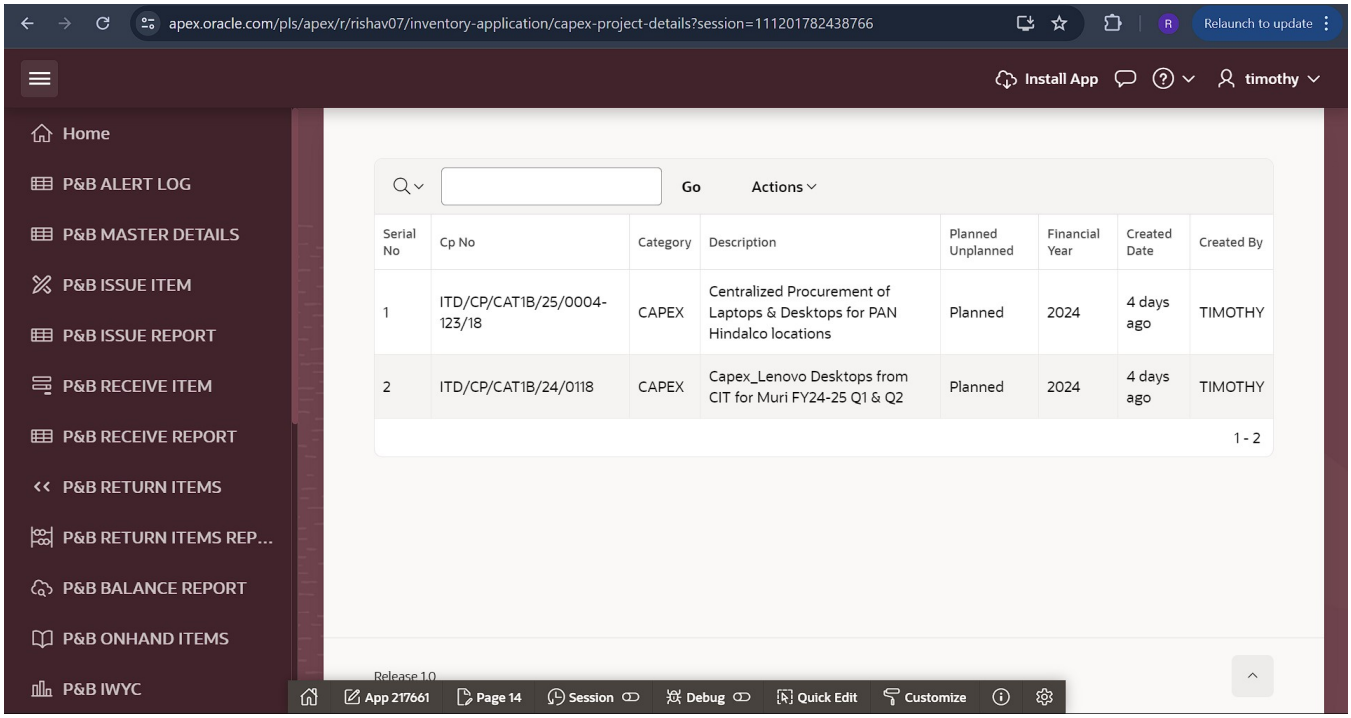
P&B Item wise Yearly report View Created

```
  CREATE OR REPLACE FORCE EDITIONABLE VIEW
"ITEMWISE_YEARLY_CONSUMPTION" ("ITEM_CODE", "ITEM_NAME",
"ISSUE_YEAR", "TOTAL_ISSUED_QUANTITY") AS
  SELECT
    ITEM_CODE,
    ITEM_NAME,
    EXTRACT(YEAR FROM ISSUE_DT) AS ISSUE_YEAR,
    SUM(ISSUE_QTY) AS TOTAL_ISSUED_QUANTITY
FROM
    ISSUE_ITEMS
GROUP BY
    ITEM_CODE, ITEM_NAME, EXTRACT(YEAR FROM ISSUE_DT);
```

CREATED A PROJECT FORM PAGE



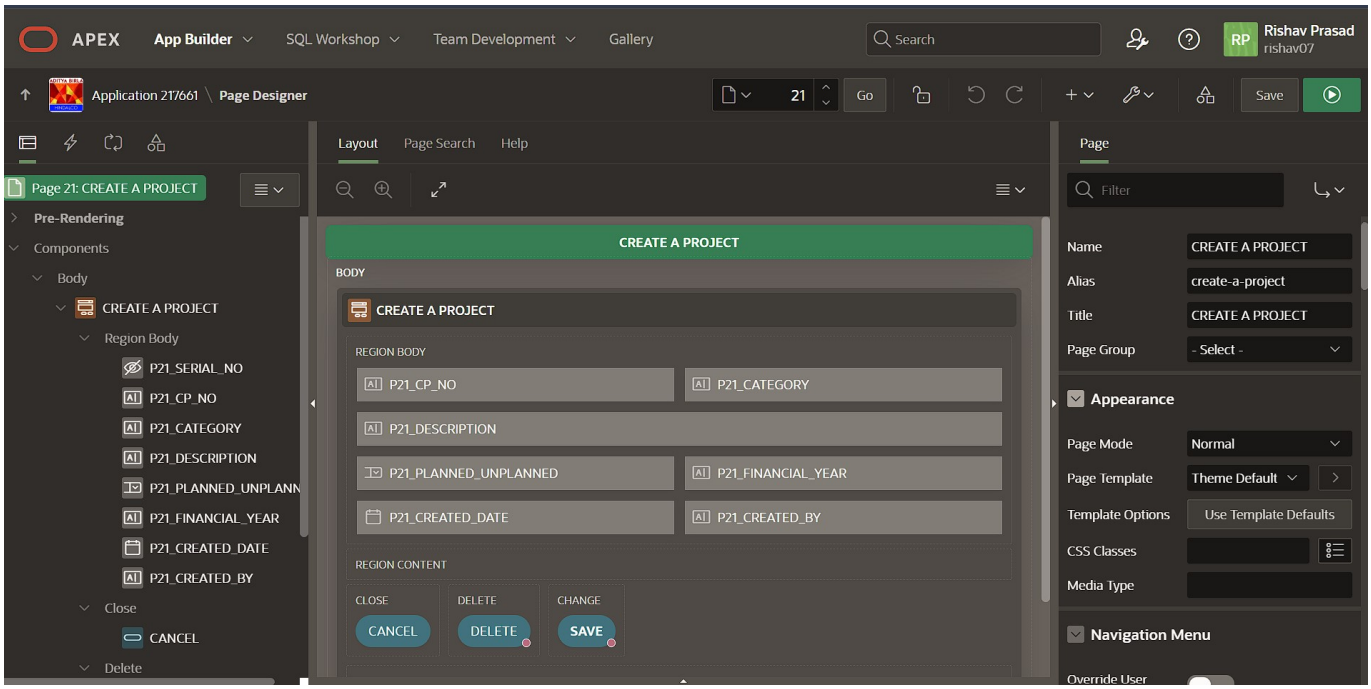
CAPEX PROJECT DETAILS PAGE



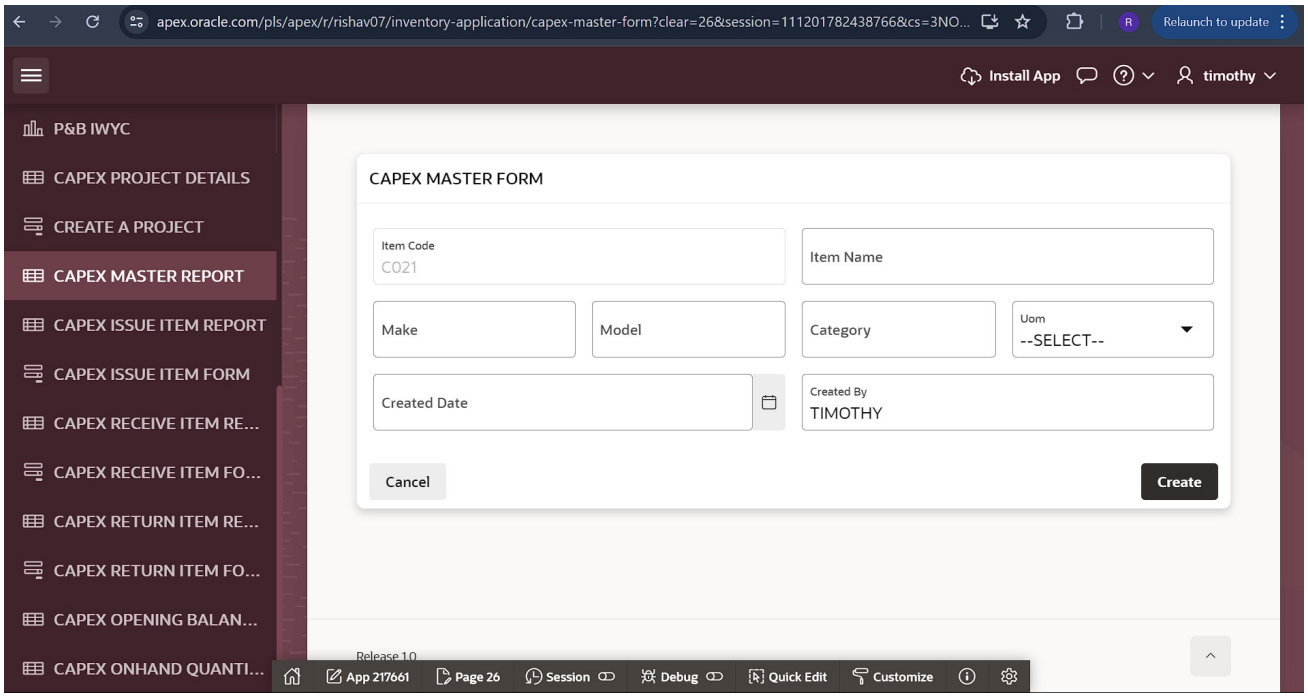
STRUCTURE OF CAPEX PROJECT DETAILS

```
CREATE TABLE "CAPEX_PROJECTS"
   (  "SERIAL_NO" NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START
WITH 1 CACHE 20 NOORDER  NOCYCLE  NOKEEP  NOSCALE  NOT NULL ENABLE,
      "CP_NO" VARCHAR2(50) NOT NULL ENABLE,
      "CATEGORY" VARCHAR2(30),
      "DESCRIPTION" VARCHAR2(100),
      "PLANNED_UNPLANNED" VARCHAR2(20),
      "FINANCIAL_YEAR" VARCHAR2(20),
      "CREATED_DATE" DATE DEFAULT SYSDATE,
      "CREATED_BY" VARCHAR2(20),
       CONSTRAINT "PK_CAPEX_PROJECTS" PRIMARY KEY ("SERIAL_NO")
  USING INDEX  ENABLE,
       CONSTRAINT "UQ_CP_NO" UNIQUE ("CP_NO")
  USING INDEX  ENABLE
   ) ;
```

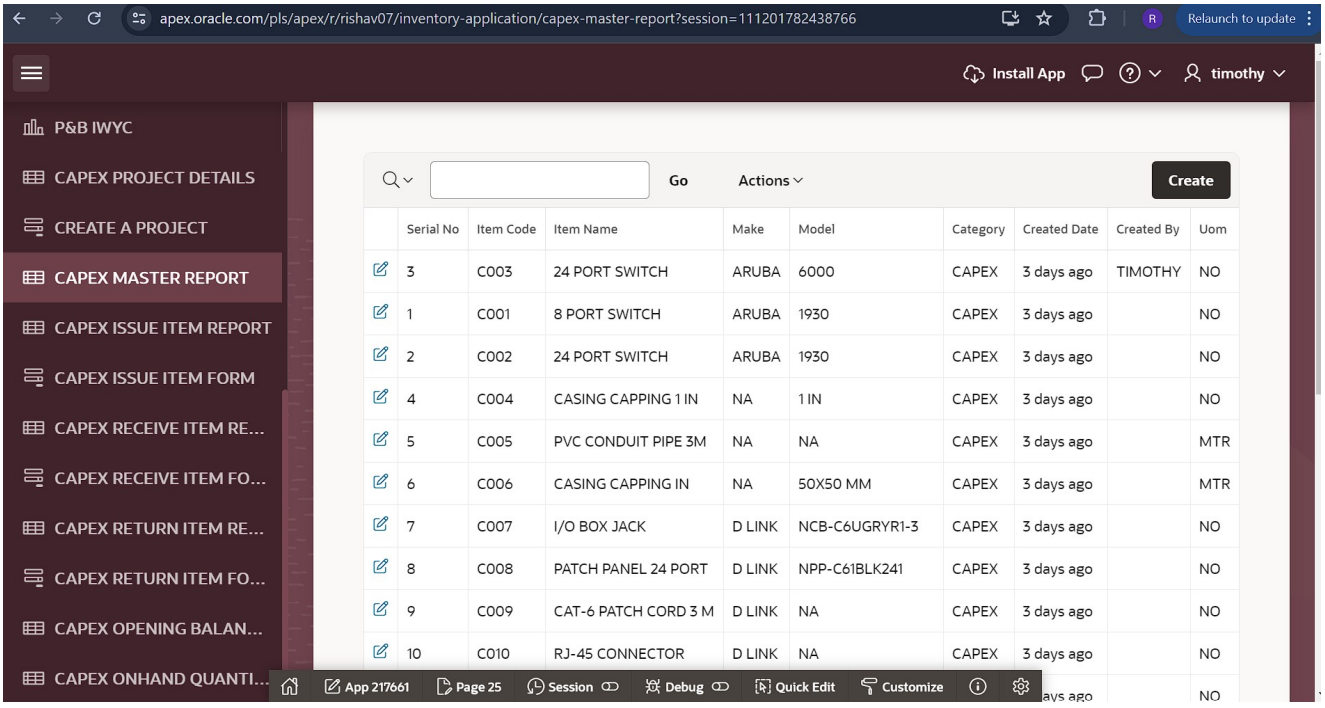
CAPEX PROJECT DETAILS PAGE



CaPeX MASTER TABLE FORM PAGE



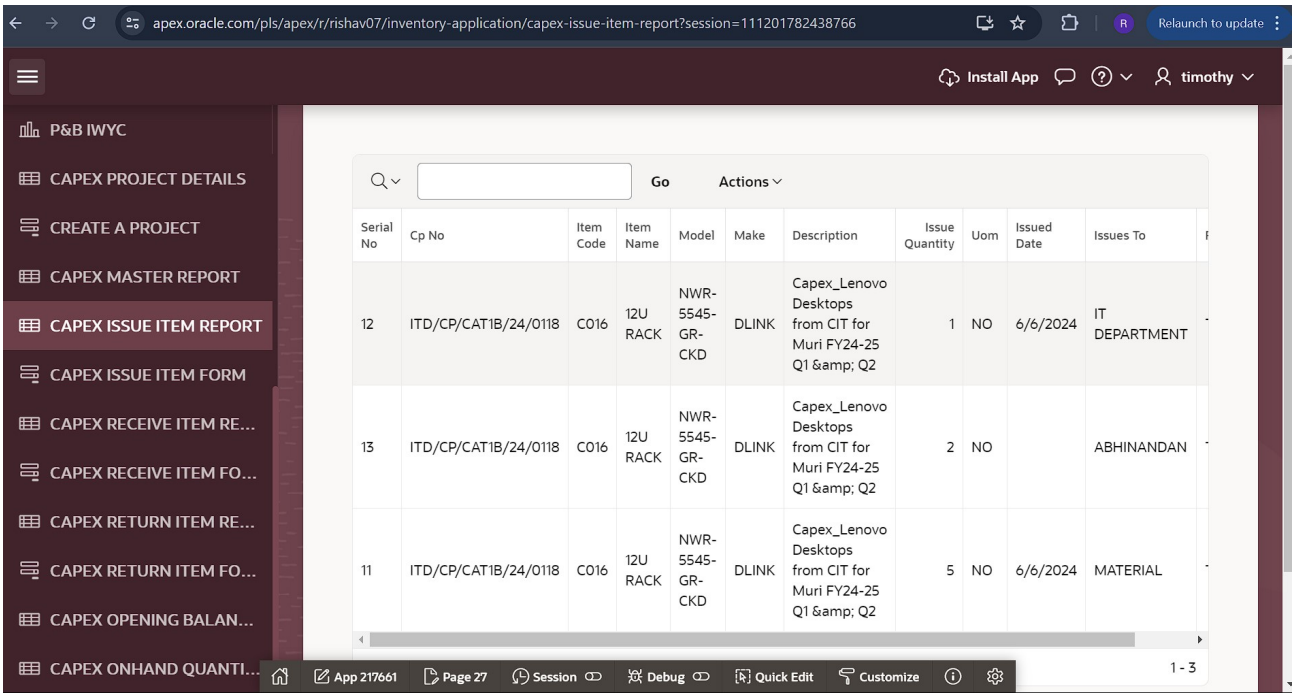
CaPeX MASTER TABLE REPORT PAGE



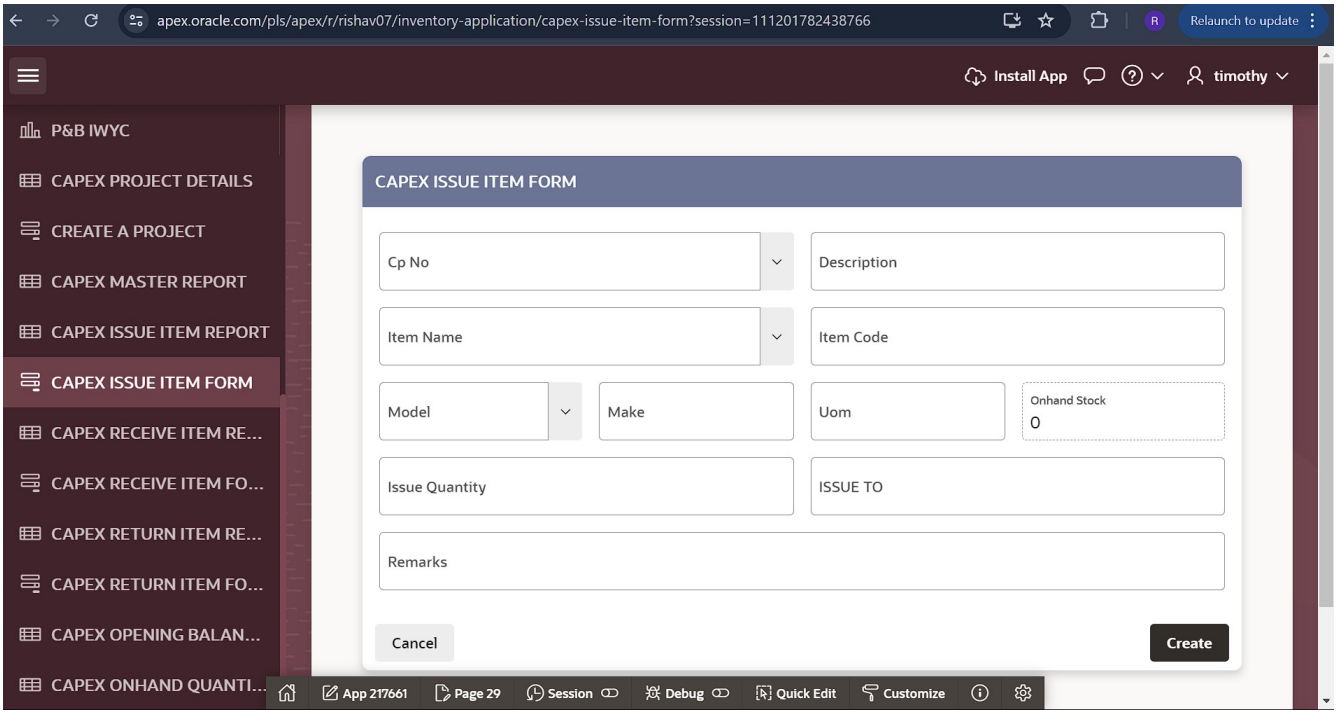
CaPeX MASTER TABLE STRUCTURE

```
CREATE TABLE "CAPEX_MASTER"
   (  "SERIAL_NO" NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1
START WITH 1 CACHE 20 NOORDER  NOCYCLE  NOKEEP  NOSCALE  NOT NULL
ENABLE,
      "ITEM_CODE" VARCHAR2(50) NOT NULL ENABLE,
      "ITEM_NAME" VARCHAR2(100) NOT NULL ENABLE,
      "MAKE" VARCHAR2(50),
      "MODEL" VARCHAR2(50),
      "CATEGORY" VARCHAR2(50),
      "CREATED_DATE" DATE DEFAULT SYSDATE,
      "CREATED_BY" VARCHAR2(50),
      "UOM" VARCHAR2(30),
       CONSTRAINT "PK_CAPEX_MASTER" PRIMARY KEY ("SERIAL_NO")
  USING INDEX  ENABLE,
       CONSTRAINT "UQ_ITEM_CODE" UNIQUE ("ITEM_CODE")
  USING INDEX  ENABLE
   ) ;
```

CaPeX ISSUE ITEM REPORT PAGE



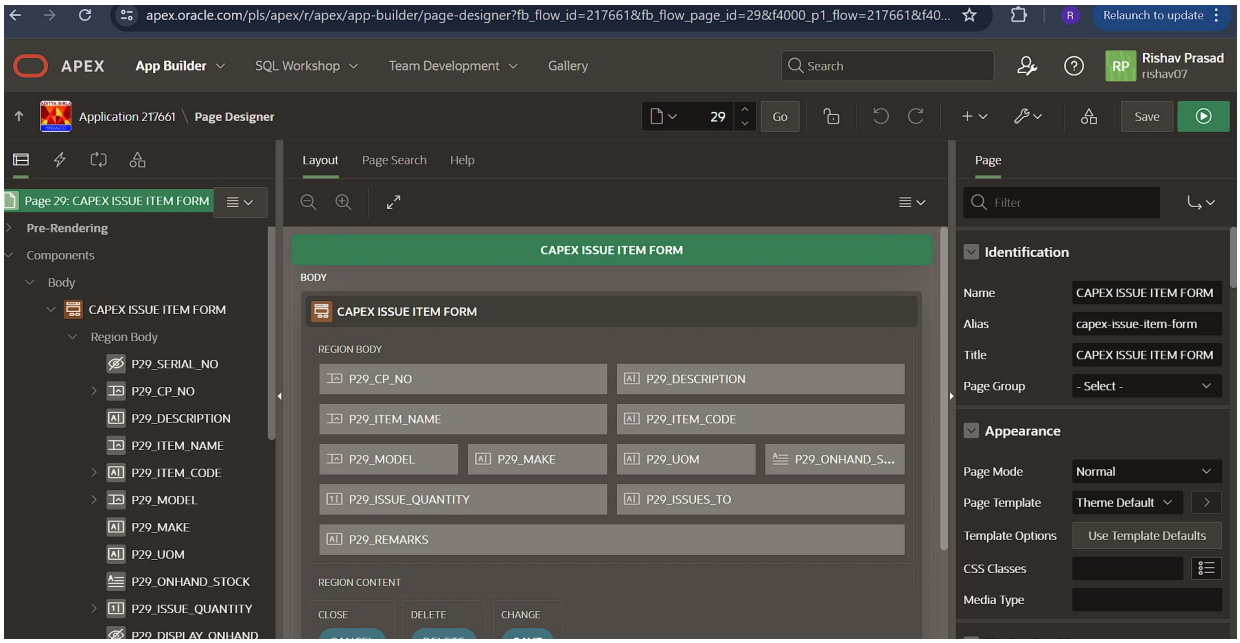
CaPeX ISSUE ITEM FORM PAGE



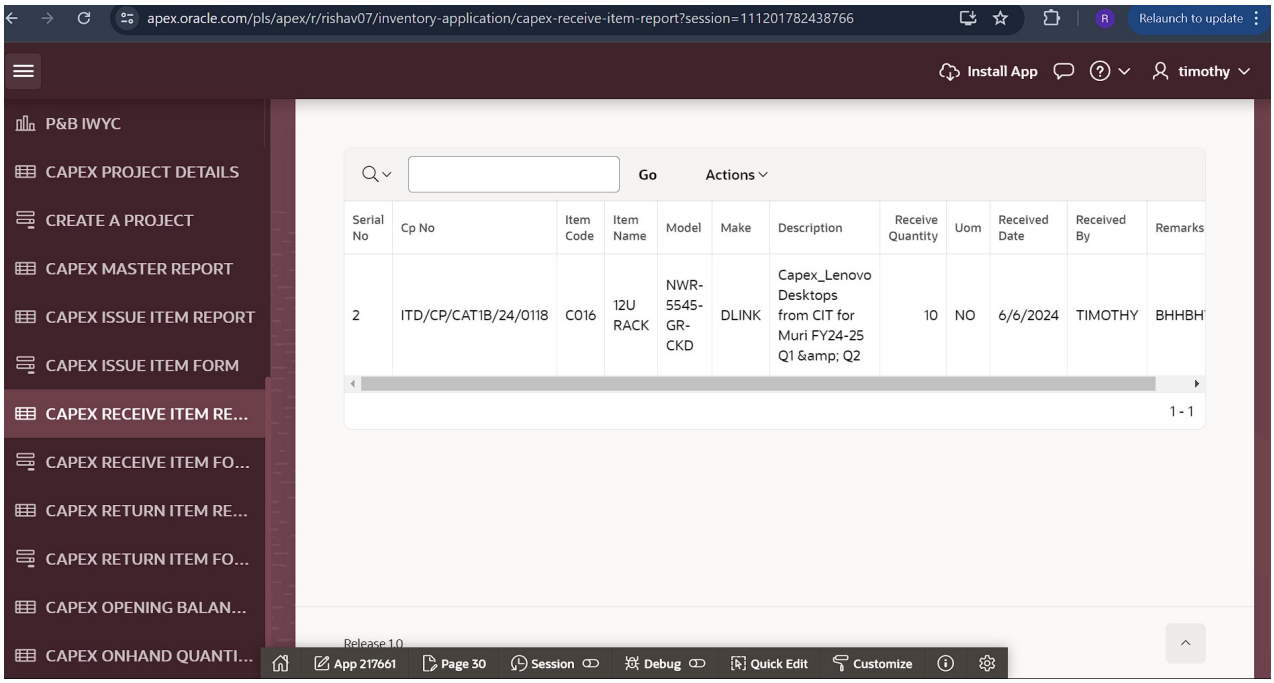
CaPeX ISSUE ITEM TABLE STRUCTURE

```
CREATE TABLE "CAPEX_ISSUE_ITEM"
  (  "SERIAL_NO" NUMBER NOT NULL ENABLE,
     "CP_NO" VARCHAR2(50) NOT NULL ENABLE,
     "ITEM_CODE" VARCHAR2(10) NOT NULL ENABLE,
     "ITEM_NAME" VARCHAR2(50),
     "MODEL" VARCHAR2(30),
     "MAKE" VARCHAR2(15),
     "DESCRIPTION" VARCHAR2(100),
     "ISSUE_QUANTITY" NUMBER NOT NULL ENABLE,
     "UOM" VARCHAR2(6),
     "ISSUED_DATE" DATE DEFAULT SYSDATE,
     "ISSUES_TO" VARCHAR2(100) NOT NULL ENABLE,
     "REMARKS" VARCHAR2(100),
      CONSTRAINT "PK_CAPEX_ISSUE_ITEM" PRIMARY KEY ("SERIAL_NO")
 USING INDEX  ENABLE
  ) ;

 ALTER TABLE "CAPEX_ISSUE_ITEM" ADD CONSTRAINT
"FK_CAPEX_ISSUE_ITEM_CP_NO" FOREIGN KEY ("CP_NO")
       REFERENCES "CAPEX_PROJECTS" ("CP_NO") ENABLE;
 ALTER TABLE "CAPEX_ISSUE_ITEM" ADD CONSTRAINT
"FK_CAPEX_ISSUE_ITEM_ITEM_CODE" FOREIGN KEY ("ITEM_CODE")
       REFERENCES "CAPEX_MASTER" ("ITEM_CODE") ENABLE;
```

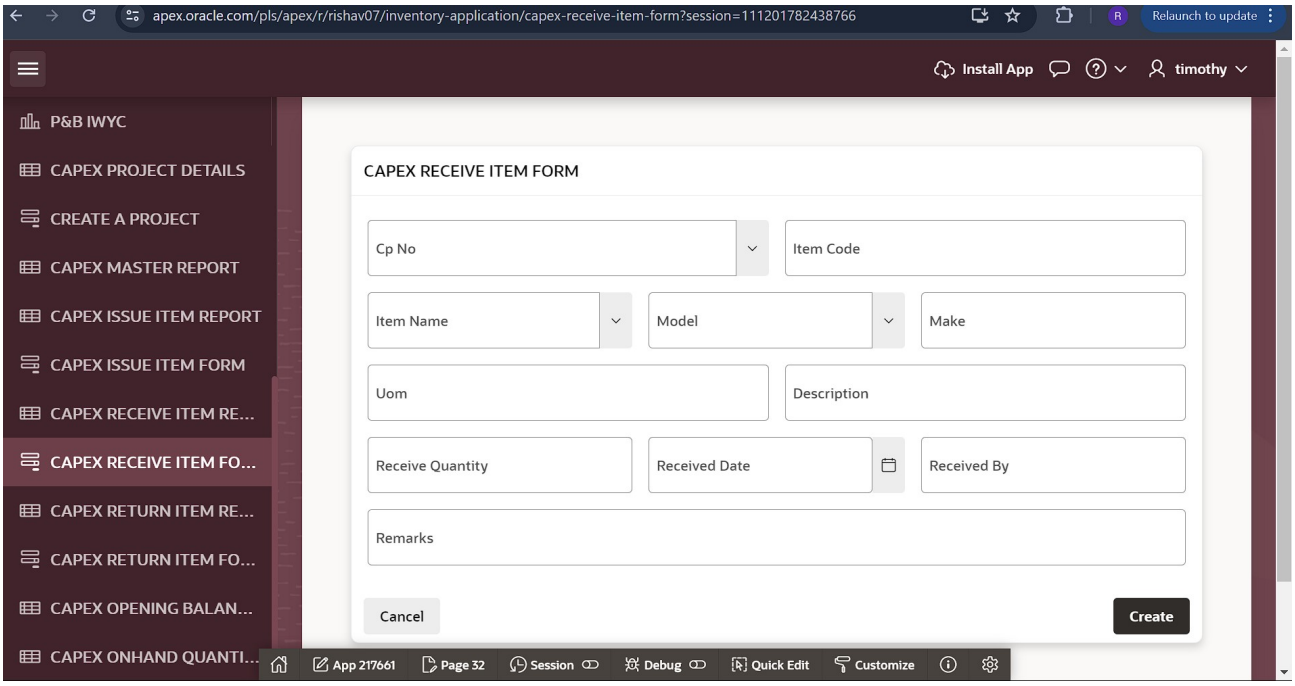
CaPeX ISSUE ITEM PAGE



CaPeX RECEIVE ITEM REPORT PAGE



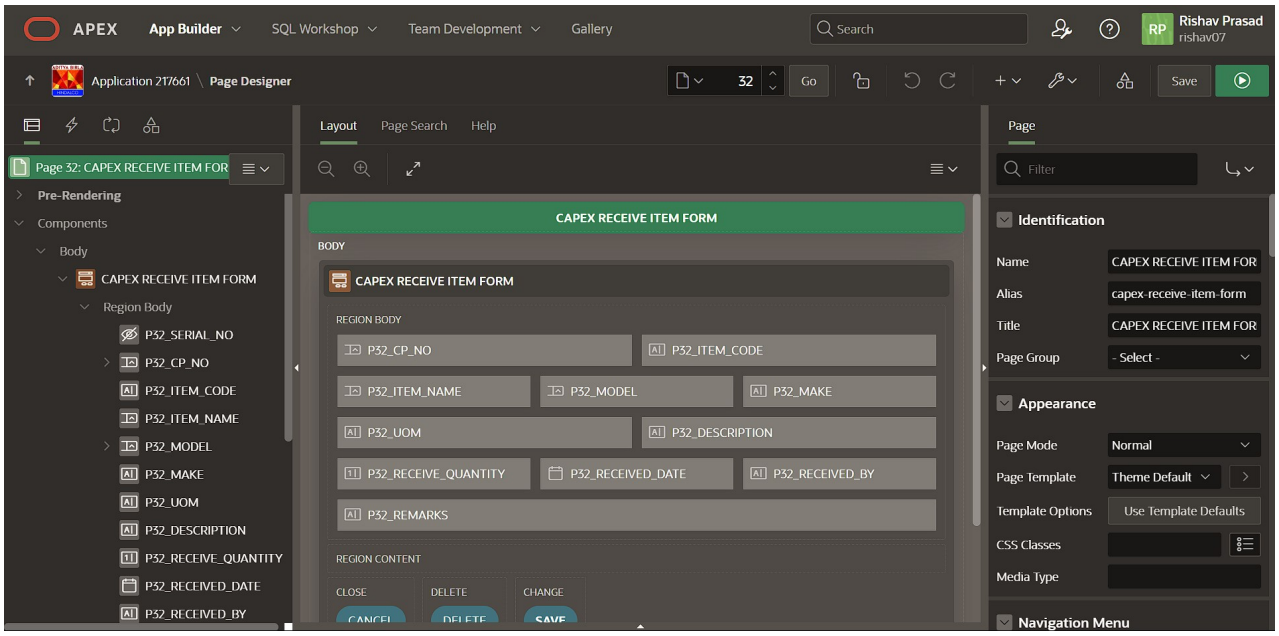
CaPeX RECEIVE ITEM FORM PAGE



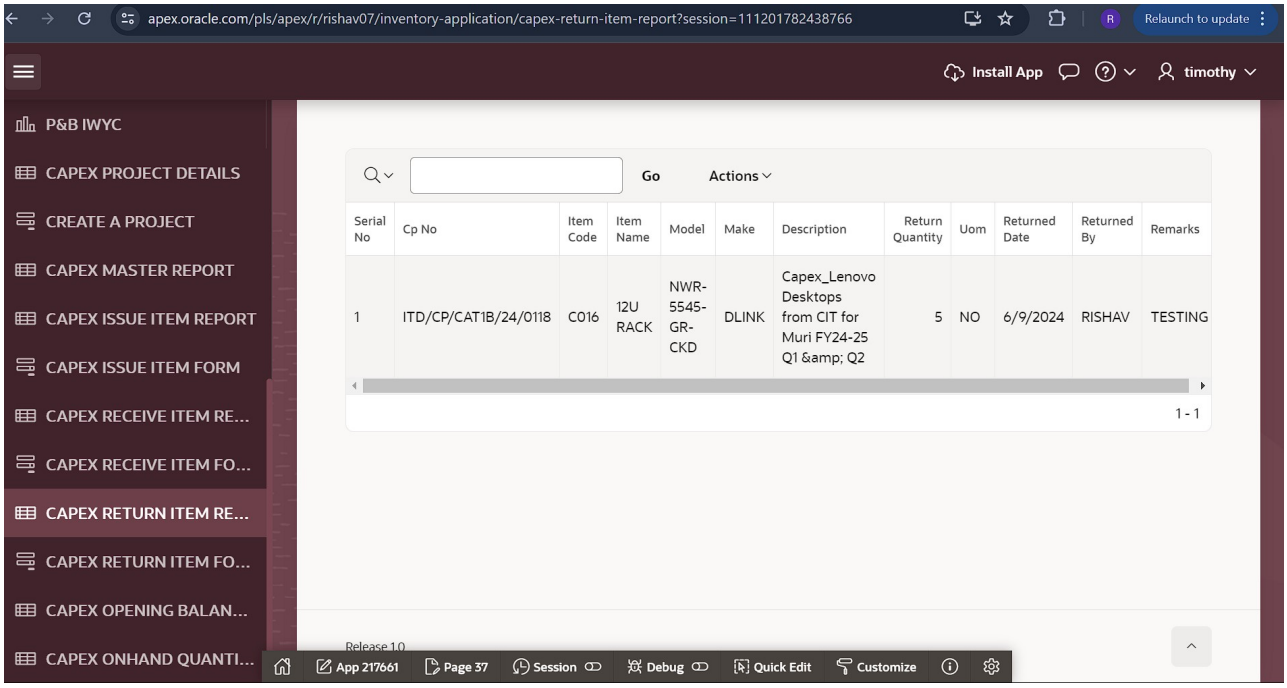
CaPeX RECEIVE ITEM TABLE STRUCTURE

```
CREATE TABLE "CAPEX_RECEIVE_ITEM"
   (  "SERIAL_NO" NUMBER NOT NULL ENABLE,
      "CP_NO" VARCHAR2(50) NOT NULL ENABLE,
      "ITEM_CODE" VARCHAR2(10) NOT NULL ENABLE,
      "ITEM_NAME" VARCHAR2(50),
      "MODEL" VARCHAR2(30),
      "MAKE" VARCHAR2(15),
      "DESCRIPTION" VARCHAR2(100),
      "RECEIVE_QUANTITY" NUMBER NOT NULL ENABLE,
      "UOM" VARCHAR2(6),
      "RECEIVED_DATE" DATE DEFAULT SYSDATE,
      "RECEIVED_BY" VARCHAR2(100) NOT NULL ENABLE,
      "REMARKS" VARCHAR2(100),
       CONSTRAINT "PK_CAPEX_RECEIVE_ITEM" PRIMARY KEY ("SERIAL_NO")
  USING INDEX  ENABLE
   ) ;

  ALTER TABLE "CAPEX_RECEIVE_ITEM" ADD CONSTRAINT
"FK_CAPEX_RECEIVE_ITEM_CP_NO" FOREIGN KEY ("CP_NO")
        REFERENCES "CAPEX_PROJECTS" ("CP_NO") ENABLE;
  ALTER TABLE "CAPEX_RECEIVE_ITEM" ADD CONSTRAINT
"FK_CAPEX_RECEIVE_ITEM_ITEM_CODE" FOREIGN KEY ("ITEM_CODE")
        REFERENCES "CAPEX_MASTER" ("ITEM_CODE") ENABLE;
```

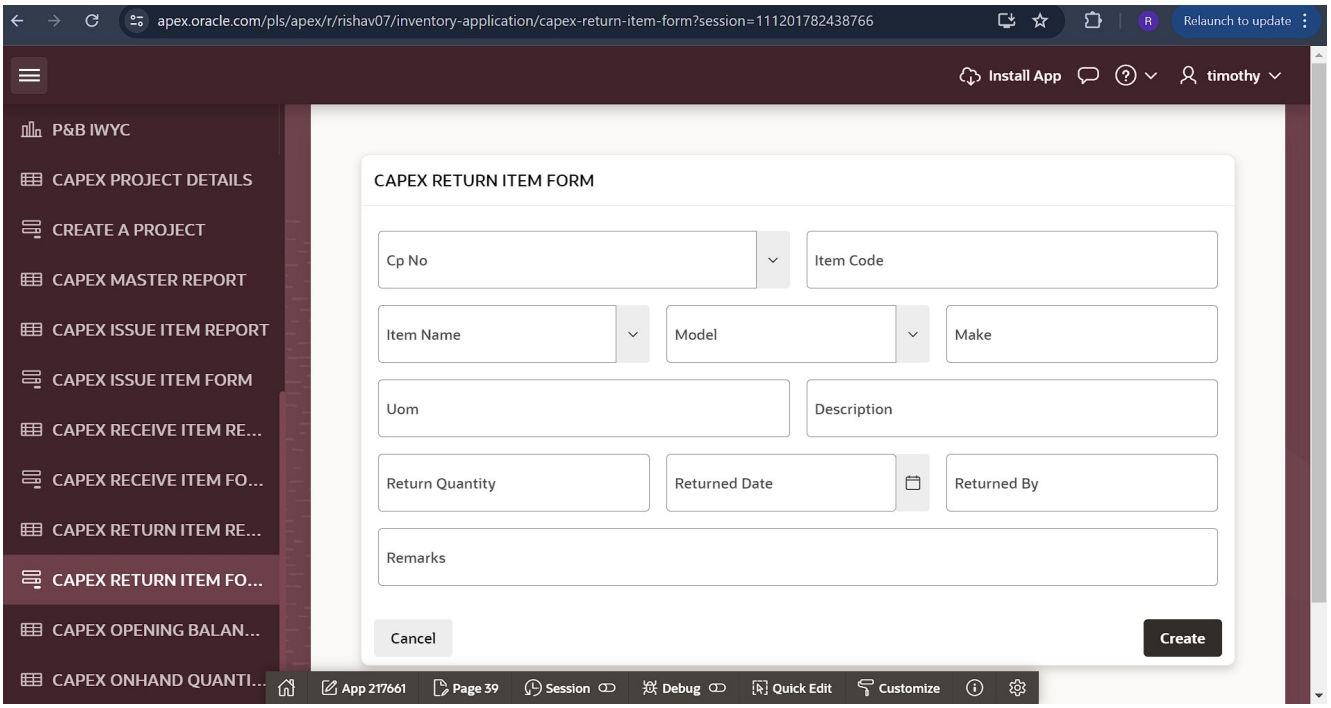
CaPeX RECEIVE ITEM PAGE



CaPeX RETURN ITEM REPORT PAGE



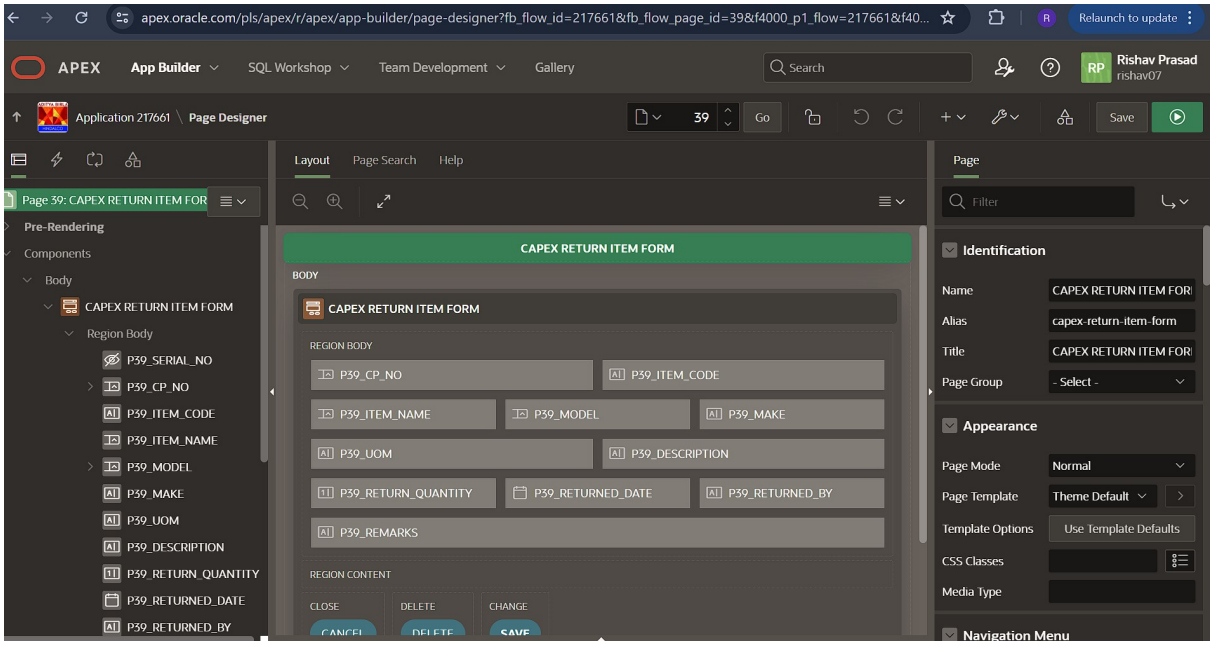
CaPeX RETURN ITEM FORM PAGE



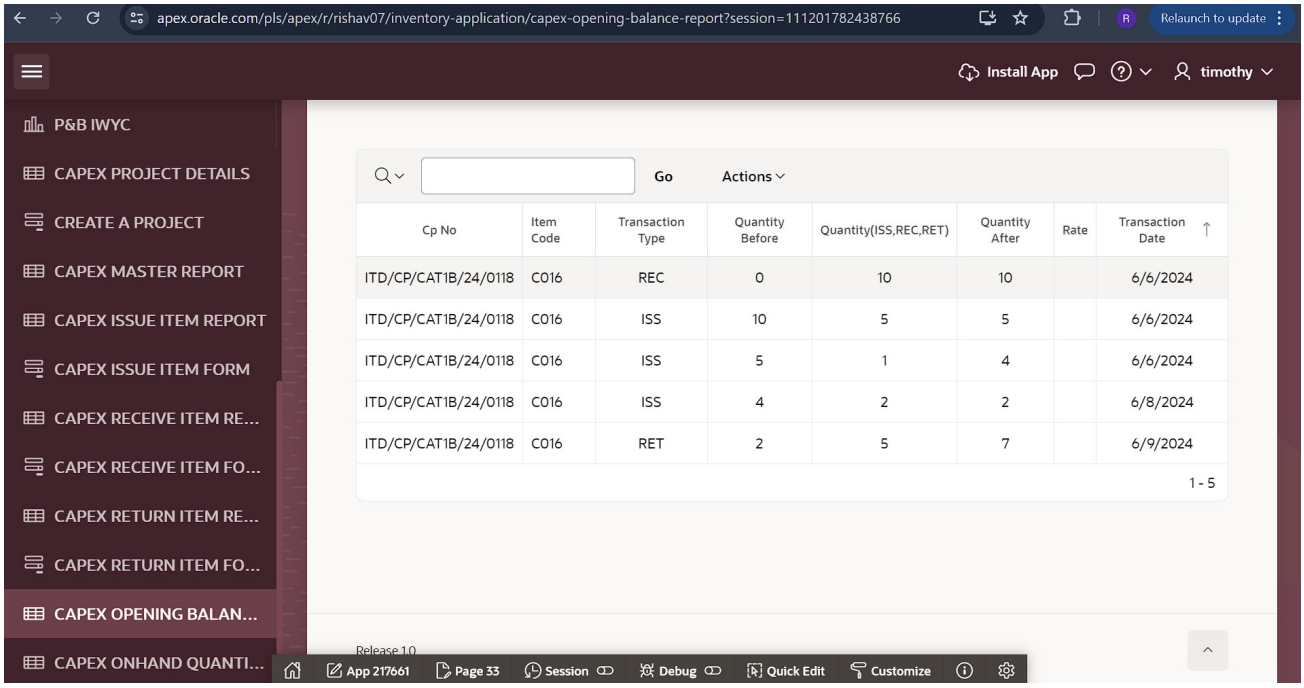
CaPeX RETURN ITEM TABLE STRUCTURE

```
  CREATE TABLE "CAPEX_RETURN_ITEM"
   (  "SERIAL_NO" NUMBER NOT NULL ENABLE,
      "CP_NO" VARCHAR2(50) NOT NULL ENABLE,
      "ITEM_CODE" VARCHAR2(10) NOT NULL ENABLE,
      "ITEM_NAME" VARCHAR2(50),
      "MODEL" VARCHAR2(30),
      "MAKE" VARCHAR2(15),
      "DESCRIPTION" VARCHAR2(100),
      "RETURN_QUANTITY" NUMBER NOT NULL ENABLE,
      "UOM" VARCHAR2(6),
      "RETURNED_DATE" DATE DEFAULT SYSDATE,
      "RETURNED_BY" VARCHAR2(100) NOT NULL ENABLE,
      "REMARKS" VARCHAR2(100),
       CONSTRAINT "PK_CAPEX_RETURN_ITEM" PRIMARY KEY ("SERIAL_NO")
  USING INDEX  ENABLE
   ) ;

  ALTER TABLE "CAPEX_RETURN_ITEM" ADD CONSTRAINT
"FK_CAPEX_RETURN_ITEM_CP_NO" FOREIGN KEY ("CP_NO")
        REFERENCES "CAPEX_PROJECTS" ("CP_NO") ENABLE;
  ALTER TABLE "CAPEX_RETURN_ITEM" ADD CONSTRAINT
"FK_CAPEX_RETURN_ITEM_ITEM_CODE" FOREIGN KEY ("ITEM_CODE")
        REFERENCES "CAPEX_MASTER" ("ITEM_CODE") ENABLE;
```

CaPeX RETURN ITEM PAGE



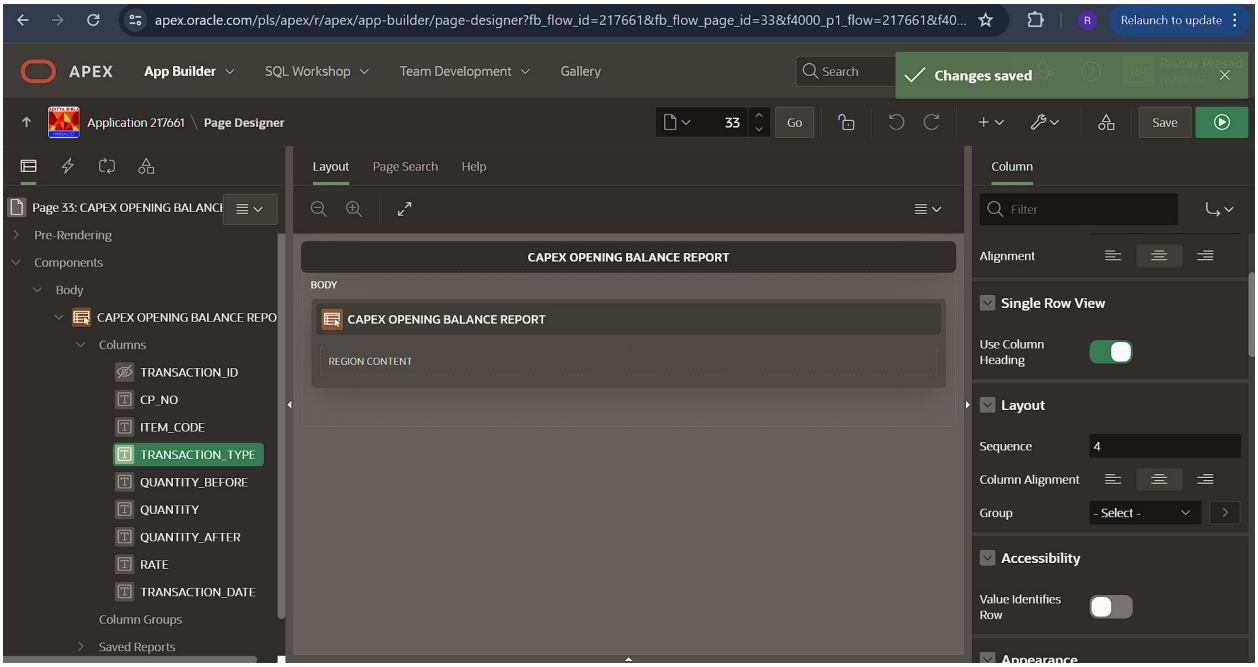
CaPeX OPENING BALANCE REPORT PAGE



CaPeX OPENING BALANCE TABLE STRUCTURE

```
CREATE TABLE "CAPEX_OPENING_BALANCE"
   (  "TRANSACTION_ID" NUMBER GENERATED BY DEFAULT AS IDENTITY
MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1
START WITH 1 CACHE 20 NOORDER  NOCYCLE  NOKEEP  NOSCALE  NOT NULL
ENABLE,
      "CP_NO" VARCHAR2(50) NOT NULL ENABLE,
      "ITEM_CODE" VARCHAR2(6) NOT NULL ENABLE,
      "TRANSACTION_TYPE" VARCHAR2(3),
      "QUANTITY_BEFORE" NUMBER,
      "QUANTITY" NUMBER NOT NULL ENABLE,
      "QUANTITY_AFTER" NUMBER,
      "RATE" NUMBER,
      "TRANSACTION_DATE" DATE DEFAULT SYSDATE,
       CONSTRAINT "CHK_CAPEX_TRANSACTION_TYPE" CHECK
(TRANSACTION_TYPE IN ('ISS', 'REC', 'RET')) ENABLE,
       PRIMARY KEY ("TRANSACTION_ID")
  USING INDEX  ENABLE
   ) ;
```

CaPeX OPENING BALANCE PAGE



CaPeX OPENING BALANCE TRIGGERS

CAPEX ISSUE ITEM TRIGGER

```
CREATE OR REPLACE EDITIONABLE TRIGGER "CAPEX_ISSUE_ITEM_TRG"
BEFORE INSERT ON CAPEX_ISSUE_ITEM
FOR EACH ROW
BEGIN
  IF :NEW.SERIAL_NO IS NULL THEN
    SELECT CAPEX_ISSUE_ITEM_SEQ.NEXTVAL INTO :NEW.SERIAL_NO FROM DUAL;
  END IF;
END;
/
ALTER TRIGGER "CAPEX_ISSUE_ITEM_TRG" ENABLE;
  CREATE OR REPLACE EDITIONABLE TRIGGER "TR_CAPEX_ISSUE_ITEM_UPDATE_OB"
FOR INSERT ON CAPEX_ISSUE_ITEM
COMPOUND TRIGGER

  TYPE t_issue_item_rec IS RECORD (
    item_code VARCHAR2(10),
    cp_no VARCHAR2(50),
    issue_quantity NUMBER
  );

  TYPE t_issue_item_tab IS TABLE OF t_issue_item_rec;
  g_issue_items t_issue_item_tab := t_issue_item_tab();

  AFTER EACH ROW IS
  BEGIN
    g_issue_items.EXTEND;
    g_issue_items(g_issue_items.COUNT) :=
t_issue_item_rec(:NEW.ITEM_CODE, :NEW.CP_NO, :NEW.ISSUE_QUANTITY);
  END AFTER EACH ROW;

  AFTER STATEMENT IS
  BEGIN
    FOR i IN 1 .. g_issue_items.COUNT LOOP
      DECLARE
        l_quantity_before NUMBER;
        l_quantity_after  NUMBER;
      BEGIN
        -- Fetch the last quantity after from the CAPEX_OPENING_BALANCE for this item code
and CP_NO
        SELECT COALESCE(
            (SELECT QUANTITY_AFTER
             FROM (SELECT QUANTITY_AFTER
                   FROM CAPEX_OPENING_BALANCE
                   WHERE ITEM_CODE = g_issue_items(i).item_code
                     AND CP_NO = g_issue_items(i).cp_no
                   ORDER BY TRANSACTION_ID DESC)
             WHERE ROWNUM = 1),
            (SELECT ISSUE_QUANTITY
             FROM CAPEX_ISSUE_ITEM
             WHERE ITEM_CODE = g_issue_items(i).item_code AND ROWNUM = 1)
        ) INTO l_quantity_before
        FROM DUAL;

        -- Calculate the quantity after the issue
        l_quantity_after := l_quantity_before - g_issue_items(i).issue_quantity;

        -- Insert into CAPEX_OPENING_BALANCE
        INSERT INTO CAPEX_OPENING_BALANCE (
            ITEM_CODE, CP_NO, TRANSACTION_TYPE, QUANTITY_BEFORE, QUANTITY, QUANTITY_AFTER,
TRANSACTION_DATE
        ) VALUES (
            g_issue_items(i).item_code, g_issue_items(i).cp_no, 'ISS', l_quantity_before,
g_issue_items(i).issue_quantity, l_quantity_after, SYSDATE
        );
```

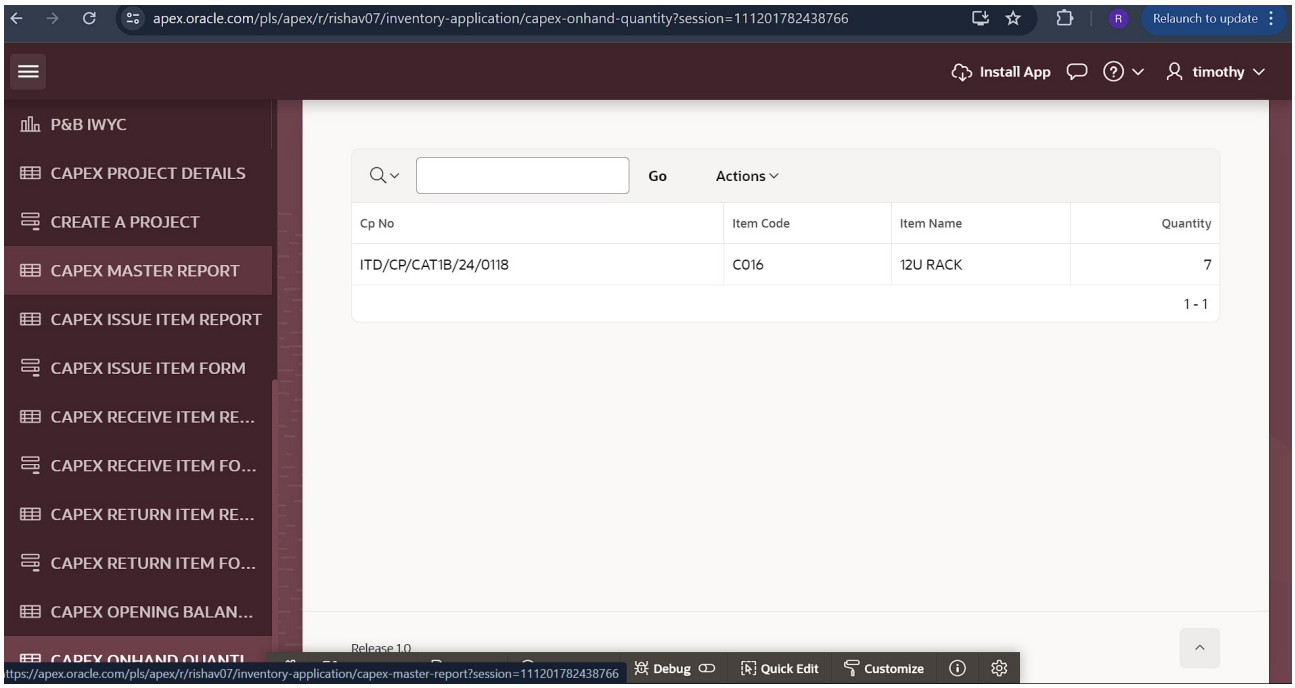
CAPEX RECEIVE ITEM TRIGGER

```
 CREATE OR REPLACE EDITIONABLE TRIGGER "CAPEX_RECEIVE_ITEM_TRG"
BEFORE INSERT ON CAPEX_RECEIVE_ITEM
FOR EACH ROW
BEGIN
  IF :NEW.SERIAL_NO IS NULL THEN
    SELECT CAPEX_RECEIVE_ITEM_SEQ.NEXTVAL INTO :NEW.SERIAL_NO FROM DUAL;
  END IF;
END;
/
ALTER TRIGGER "CAPEX_RECEIVE_ITEM_TRG" ENABLE;
  CREATE OR REPLACE EDITIONABLE TRIGGER "TR_CAPEX_RECEIVE_ITEM_UPDATE_OB"
FOR INSERT ON CAPEX_RECEIVE_ITEM
COMPOUND TRIGGER

  TYPE t_receive_item_rec IS RECORD (
    item_code VARCHAR2(10),
    cp_no VARCHAR2(50),
    receive_quantity NUMBER
  );

  TYPE t_receive_item_tab IS TABLE OF t_receive_item_rec;
  g_receive_items t_receive_item_tab := t_receive_item_tab();

  AFTER EACH ROW IS
  BEGIN
    g_receive_items.EXTEND;
    g_receive_items(g_receive_items.COUNT) :=
t_receive_item_rec(:NEW.ITEM_CODE, :NEW.CP_NO, :NEW.RECEIVE_QUANTITY);
  END AFTER EACH ROW;

  AFTER STATEMENT IS
  BEGIN
    FOR i IN 1 .. g_receive_items.COUNT LOOP
      DECLARE
        l_quantity_before NUMBER;
        l_quantity_after  NUMBER;
      BEGIN
        -- Fetch the last quantity after from the CAPEX_OPENING_BALANCE for this item code
and CP_NO
        SELECT COALESCE(
            (SELECT QUANTITY_AFTER
             FROM (SELECT QUANTITY_AFTER
                   FROM CAPEX_OPENING_BALANCE
                   WHERE ITEM_CODE = g_receive_items(i).item_code
                     AND CP_NO = g_receive_items(i).cp_no
                   ORDER BY TRANSACTION_ID DESC)
             WHERE ROWNUM = 1),
            0 -- Assuming 0 as default initial stock if no previous record exists
        ) INTO l_quantity_before
        FROM DUAL;

        -- Calculate the quantity after the receive
        l_quantity_after := l_quantity_before + g_receive_items(i).receive_quantity;

        -- Insert into CAPEX_OPENING_BALANCE
        INSERT INTO CAPEX_OPENING_BALANCE (
            ITEM_CODE, CP_NO, TRANSACTION_TYPE, QUANTITY_BEFORE, QUANTITY, QUANTITY_AFTER,
TRANSACTION_DATE
        ) VALUES (
            g_receive_items(i).item_code, g_receive_items(i).cp_no, 'REC',
l_quantity_before, g_receive_items(i).receive_quantity, l_quantity_after, SYSDATE
        );
```

CAPEX RETURN ITEM TRIGGER

```
CREATE OR REPLACE EDITIONABLE TRIGGER "CAPEX_RETURN_ITEM_TRG"
BEFORE INSERT ON CAPEX_RETURN_ITEM
FOR EACH ROW
BEGIN
    IF :NEW.SERIAL_NO IS NULL THEN
        SELECT CAPEX_RETURN_ITEM_SEQ.NEXTVAL INTO :NEW.SERIAL_NO FROM DUAL;
    END IF;
END;
/
ALTER TRIGGER "CAPEX_RETURN_ITEM_TRG" ENABLE;
  CREATE OR REPLACE EDITIONABLE TRIGGER "TR_CAPEX_RETURN_ITEM_UPDATE_OB"
FOR INSERT ON CAPEX_RETURN_ITEM
COMPOUND TRIGGER

  TYPE t_return_item_rec IS RECORD (
    item_code VARCHAR2(10),
    cp_no VARCHAR2(50),
    return_quantity NUMBER
  );

  TYPE t_return_item_tab IS TABLE OF t_return_item_rec;
  g_return_items t_return_item_tab := t_return_item_tab();

  AFTER EACH ROW IS
  BEGIN
    g_return_items.EXTEND;
    g_return_items(g_return_items.COUNT) :=
t_return_item_rec(:NEW.ITEM_CODE, :NEW.CP_NO, :NEW.RETURN_QUANTITY);
  END AFTER EACH ROW;

  AFTER STATEMENT IS
  BEGIN
    FOR i IN 1 .. g_return_items.COUNT LOOP
      DECLARE
        l_quantity_before NUMBER;
        l_quantity_after  NUMBER;
      BEGIN
        -- Fetch the last quantity after from the CAPEX_OPENING_BALANCE for this item code
and CP_NO
        SELECT COALESCE(
            (SELECT QUANTITY_AFTER
             FROM (SELECT QUANTITY_AFTER
                   FROM CAPEX_OPENING_BALANCE
                   WHERE ITEM_CODE = g_return_items(i).item_code
                     AND CP_NO = g_return_items(i).cp_no
                   ORDER BY TRANSACTION_ID DESC)
             WHERE ROWNUM = 1),
            0 -- Assuming 0 as default initial stock if no previous record exists
        ) INTO l_quantity_before
        FROM DUAL;

        -- Calculate the quantity after the return
        l_quantity_after := l_quantity_before + g_return_items(i).return_quantity;

        -- Insert into CAPEX_OPENING_BALANCE
        INSERT INTO CAPEX_OPENING_BALANCE (
            ITEM_CODE, CP_NO, TRANSACTION_TYPE, QUANTITY_BEFORE, QUANTITY, QUANTITY_AFTER,
TRANSACTION_DATE
        ) VALUES (
            g_return_items(i).item_code, g_return_items(i).cp_no, 'RET',
l_quantity_before, g_return_items(i).return_quantity, l_quantity_after, SYSDATE
        );
```

CaPeX ONHAND REPORT



CaPeX ONHAND TABLE STRUCTURE

```
CREATE TABLE "CAPEX_ONHAND_QUANTITY"
 (  "CP_NO" VARCHAR2(50) NOT NULL ENABLE,
    "ITEM_CODE" VARCHAR2(6) NOT NULL ENABLE,
    "ITEM_NAME" VARCHAR2(50),
    "QUANTITY" NUMBER,
     PRIMARY KEY ("CP_NO", "ITEM_CODE")
USING INDEX  ENABLE
 ) ;
```

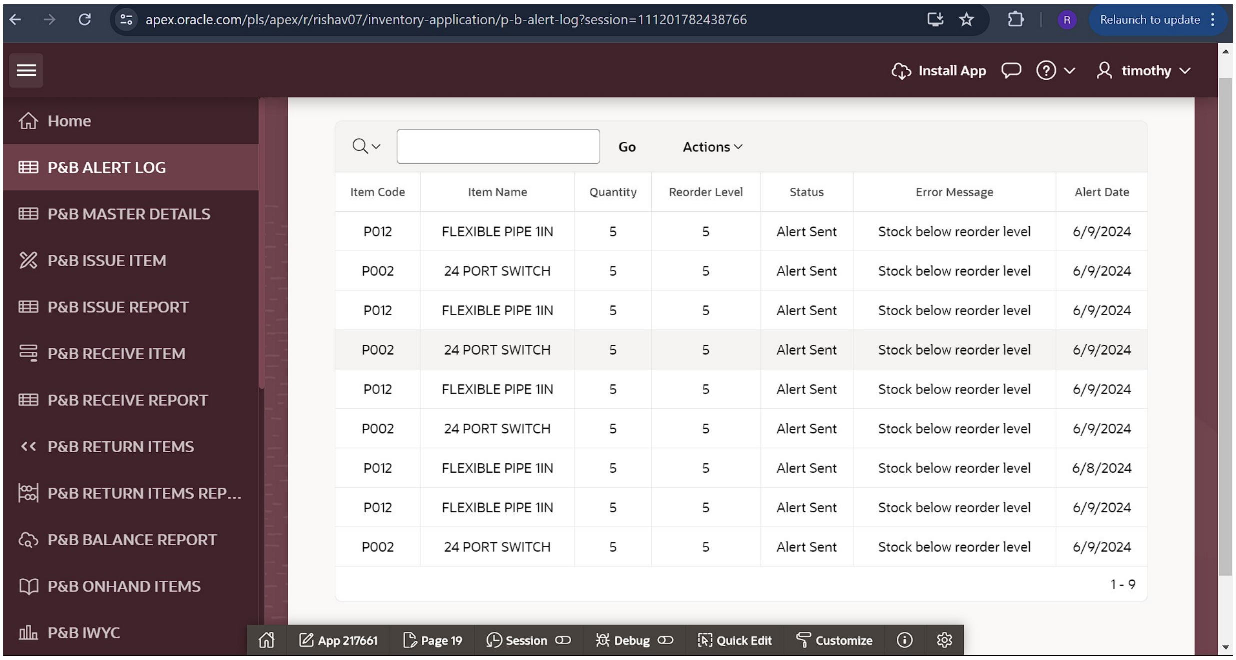
CaPeX ONHAND TRIGGER

```
create or replace TRIGGER TR_UPDATE_CAPEX_ONHAND_QUANTITY
AFTER INSERT OR UPDATE ON CAPEX_OPENING_BALANCE
FOR EACH ROW
BEGIN
    MERGE INTO CAPEX_ONHAND_QUANTITY coq
    USING (SELECT :NEW.CP_NO AS CP_NO,
                  :NEW.ITEM_CODE AS ITEM_CODE,
                  :NEW.QUANTITY_AFTER AS QUANTITY,
                  (SELECT ITEM_NAME FROM CAPEX_MASTER WHERE ITEM_CODE
= :NEW.ITEM_CODE) AS ITEM_NAME
           FROM DUAL) src
    ON (coq.CP_NO = src.CP_NO AND coq.ITEM_CODE = src.ITEM_CODE)
    WHEN MATCHED THEN
        UPDATE SET coq.QUANTITY = src.QUANTITY, coq.ITEM_NAME =
src.ITEM_NAME
    WHEN NOT MATCHED THEN
        INSERT (coq.CP_NO, coq.ITEM_CODE, coq.ITEM_NAME,
coq.QUANTITY)
        VALUES (src.CP_NO, src.ITEM_CODE, src.ITEM_NAME,
src.QUANTITY);
END;
/
```

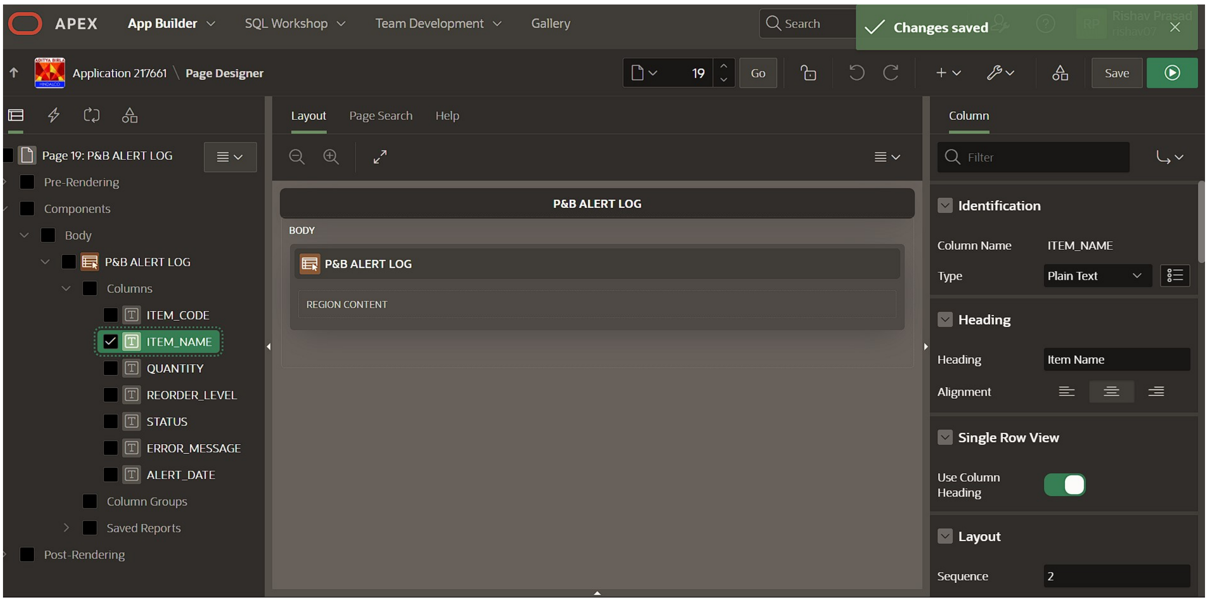
CaPeX ONHAND STOCK PROCEDURE

```
create or replace FUNCTION get_capex_onhand_stock (
    p_item_code IN VARCHAR2,
    p_cp_no IN VARCHAR2
) RETURN NUMBER IS
    l_onhand_stock NUMBER;
BEGIN
    SELECT COALESCE(
        (SELECT QUANTITY_AFTER
         FROM CAPEX_OPENING_BALANCE
         WHERE ITEM_CODE = p_item_code
           AND CP_NO = p_cp_no
         ORDER BY TRANSACTION_ID DESC FETCH FIRST 1 ROWS
ONLY),
        0 -- Assuming 0 as default initial stock if no
previous record exists
    ) INTO l_onhand_stock
    FROM DUAL;

    RETURN l_onhand_stock;
END get_capex_onhand_stock;
/
```

P&B ALERT LOG PAGE



P&B ALERT LOG



P&B ALERT LOG PROCEDURE

```
CREATE OR REPLACE PROCEDURE CHECK_REORDER_LEVEL IS
    CURSOR reorder_cursor IS
        SELECT *
        FROM (
            SELECT I.ITEM_CODE, I.ITEM_NAME, I.REORDER_LEVEL,
                   NVL(I.STOCK, 0) + NVL(R.RECV_QTY, 0) - NVL(ISS.ISSUE_QTY, 0) + NVL(RT.RETURN_QTY, 0) AS
QUANTITY
            FROM IT_ITEMS I
            LEFT JOIN (
                SELECT ITEM_CODE, SUM(RECV_QTY) AS RECV_QTY
                FROM RECEIVE_ITEMS
                GROUP BY ITEM_CODE
            ) R ON I.ITEM_CODE = R.ITEM_CODE
            LEFT JOIN (
                SELECT ITEM_CODE, SUM(ISSUE_QTY) AS ISSUE_QTY
                FROM ISSUE_ITEMS
                GROUP BY ITEM_CODE
            ) ISS ON I.ITEM_CODE = ISS.ITEM_CODE
            LEFT JOIN (
                SELECT ITEM_CODE, SUM(RETURN_QTY) AS RETURN_QTY
                FROM RETURN_ITEMS
                GROUP BY ITEM_CODE
            ) RT ON I.ITEM_CODE = RT.ITEM_CODE
            WHERE I.REORDER_LEVEL > 0
        )
        WHERE QUANTITY <= REORDER_LEVEL;

    l_item_code IT_ITEMS.ITEM_CODE%TYPE;
    l_item_name IT_ITEMS.ITEM_NAME%TYPE;
    l_reorder_level IT_ITEMS.REORDER_LEVEL%TYPE;
    l_quantity NUMBER;

    -- Add a variable to hold the list of email addresses
    l_email_list VARCHAR2(4000);
BEGIN
    -- Fetch the email list from the ADMIN_EMAILS table
    SELECT LISTAGG(EMAIL, ',') WITHIN GROUP (ORDER BY EMAIL)
    INTO l_email_list
    FROM ADMIN_EMAILS;

    OPEN reorder_cursor;
    LOOP
        FETCH reorder_cursor INTO l_item_code, l_item_name, l_reorder_level, l_quantity;
        EXIT WHEN reorder_cursor%NOTFOUND;

        IF l_quantity <= l_reorder_level THEN
            -- Insert initial log entry with status "Scheduled at 8 AM"
            INSERT INTO REORDER_ALERT_LOG (ITEM_CODE, ITEM_NAME, QUANTITY, REORDER_LEVEL, STATUS,
ERROR_MESSAGE, ALERT_DATE)
            VALUES (l_item_code, l_item_name, l_quantity, l_reorder_level, 'Scheduled at 8 AM', 'Stock
below reorder level', SYSDATE);

            -- Send the email
            APEX_MAIL.SEND(
                p_to       => l_email_list, -- Use the dynamic email list
                p_from     => 'inventory@example.com',
                p_subj     => 'Reorder Alert',
                p_body     => 'Item Code: ' || l_item_code ||
                              ', Item Name: ' || l_item_name ||
                              ', Quantity: ' || l_quantity ||
                              ', Reorder Level: ' || l_reorder_level ||
                              '. Please reorder the item as soon as possible.'
            );

            -- Update the log entry status to "Alert Sent"
            UPDATE REORDER_ALERT_LOG
            SET STATUS = 'Alert Sent'
            WHERE ITEM_CODE = l_item_code
            AND ITEM_NAME = l_item_name
            AND ALERT_DATE = (SELECT MAX(ALERT_DATE) FROM REORDER_ALERT_LOG WHERE ITEM_CODE = l_item_code);
        END IF;
    END LOOP;
    CLOSE reorder_cursor;
END;
/
```

CONCLUSION

The Development of the Inventory Management System for the IT Department At Hindalco Industries Ltd. Has Successfully addressed the challenges associated with managing Inventory across multiple systems. The system provides a user-friendly interface for authorized users to efficiently track, control, and report Inventory Data. By leveraging technologies such as HTML, CSS, Oracle APEX, MySQL, PL/SQL, the system has streamlined inventory management processes and improved overall efficiency within the department.
Through the development and implementation of the Inventory Management System, several key achievements have been relalized. This include:

- Improved Inventory Control and Accuracy: The system allows for Real Time Tracking of Inventory Items, ensuring better control and reducing instances of stockouts or excess inventory.
- Enhanced Reporting Capabilities: With the system's reporting functionality, users can generate detailed reports on Inventory status, transactions, and usage enabling informed decision-making.
- Increased Productivity : Automation of Inventory Management process has reduced manual efforts, enabling IT Department personnel to focus on other critical tasks.

The Successful deployement and utilization of the Inventory Management System have brought tangible benefits to Hindalco Industries Ltd's IT Department, fostering operational efficiency and cost optimization.

FUTURE SCOPE

While the Current Version of the Inventory Management System fulfills the Initial requirements, there is a scope for further enhancements and expansions. Some potential areas for future development include:

Integration with external System:
Integration with Procurement systems to automate the inventory replenishment process.
Integration with financial systems for better cost Tracking and financial analysis.

Advanced Analytics and Forecasting:
Incorporating data analysis and forecasting techniques to predictt inventory demand, optimize reorder points, and improve overall inventory Planning.

Mobile Application Development:
Creating mobile application for inventory management, providing on-the-go access and real time updates to authorized users.

Supplier Collaboration:
Implementing Features that enables better collaboration with Suppliers, facilitating efficient communication and streamlining the procurement process.

These Future enhancements would further strengthen the inventory management system, allowing Hindalco Industries Ltd's Department to maximize efficiency, reduce costs, and adapt to evolving business needs.