Code for SQL Injections  
  
# src/streamlit\_app.py

import streamlit as st

import src.db\_utils as db\_utils  # Import db\_utils from src directory

def display\_products\_streamlit(products, query\_type, sql\_query=None): # Added sql\_query parameter

    """Displays product information in Streamlit, with query type heading and SQL query."""

    st.subheader(f"{query\_type} Results") # Subheader for query type

    if sql\_query: # Display SQL query if provided

        st.code(sql\_query, language="sql") # Use st.code to display SQL nicely

    if products:

        product\_data = []

        for product in products:

            product\_data.append({

                "ID": product[0],

                "Name": product[1],

                "Description": product[2],

                "Price": f"${product[3]:.2f}"

            })

        st.dataframe(product\_data) # Display as a dataframe for better readability

    else:

        st.write("No products found.")

    st.markdown("---") # Separator line

def main():

    st.title("SQL Injection Demo - Product Search")

    st.markdown("This application demonstrates \*\*SQL Injection vulnerability\*\* in the 'Vulnerable Query' vs. the 'Safe Query' using parameterized queries.")

    st.markdown("\*\*Instructions:\*\* Enter a product name or an SQL Injection payload in the text box below to see the difference in results between the vulnerable and safe queries.")

    search\_term = st.text\_input("Enter product name or SQL Injection payload to search:", "")

    if search\_term:

        st.write(f"Searching for: \*\*'{search\_term}'\*\*")

        st.write("### Vulnerable Query Results:")

        st.error("This query is \*\*VULNERABLE to SQL Injection\*\*. User input is directly embedded into the SQL query string.") # Use st.error to highlight vulnerability

        vulnerable\_products = db\_utils.fetch\_product\_by\_name(search\_term)

        # Get the actual query string from db\_utils (you might need to modify db\_utils to return it or capture it)

        # For now, just reconstruct a basic example for display

        example\_vulnerable\_query = f"SELECT \* FROM products WHERE name = '{search\_term}'"

        display\_products\_streamlit(vulnerable\_products, "Vulnerable Query (INJECTED)", sql\_query=example\_vulnerable\_query) # Pass sql\_query

        st.write("### Safe Query Results:")

        st.success("This query is \*\*SAFE from SQL Injection\*\*. It uses parameterized queries, which prevent user input from being interpreted as SQL code.") # Use st.success to highlight safety

        safe\_products = db\_utils.fetch\_product\_by\_name\_safe(search\_term)

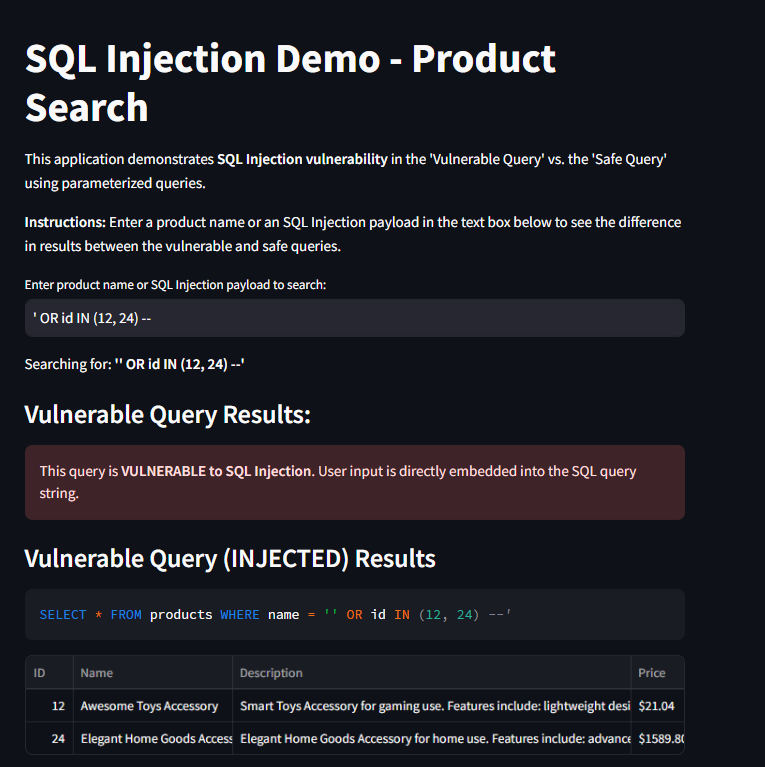
        # Similar to vulnerable query, reconstruct a basic example for display

        example\_safe\_query = "SELECT \* FROM products WHERE name = ?" # Parameterized query structure

        display\_products\_streamlit(safe\_products, "Safe Query (Parameterized)", sql\_query=example\_safe\_query) # Pass sql\_query

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output :

