```
import os
import re
import sys
import logging
import yaml
import streamlit as st
from datetime import date

current_dir = os.path.dirname(os.path.abspath(__file__))
kit_dir = os.path.abspath(os.path.join(current_dir, ".."))
repo_dir = os.path.abspath(os.path.join(kit_dir, ".."))

sys.path.append(kit_dir)
sys.path.append(repo_dir)

from enterprise_knowledge_retriever.src.document_retrieval import DocumentRetrieval
from utils.visual.env_utils import env_input_fields, initialize_env_variables, are_credentials_set,
save_credentials
from utils.vectordb.vector_db import VectorDb

CONFIG_PATH = os.path.join(kit_dir, 'config.yaml')
PERSIST_DIRECTORY = os.path.join(kit_dir, f"data/my-vector-db")

logging.basicConfig(level=logging.INFO)
logging.info("URL: http://localhost:8501")


def handle_userinput(user_question):
    if user_question:
```

```
    try:
        with st.spinner("Processing..."):
            response = st.session_state.conversation.invoke({"question": user_question})
        st.session_state.chat_history.append(user_question)
        st.session_state.chat_history.append(response["answer"])

        sources = set([
            f'{sd.metadata["filename"]}'
            for sd in response["source_documents"]
        ])
        sources_text = ""
        for index, source in enumerate(sources, start=1):
            source_link = source
            sources_text += (
                f'<font size="2" color="grey">{index}. {source_link}</font>  \n'
            )
        st.session_state.sources_history.append(sources_text)
    except Exception as e:
        st.error(f"An error occurred while processing your question: {str(e)}")

for ques, ans, source in zip(
        st.session_state.chat_history[::2],
        st.session_state.chat_history[1::2],
        st.session_state.sources_history,
):
    with st.chat_message("user"):
        st.write(f"{ques}")

    with st.chat_message(
```

```
            "ai",
            avatar="https://sambanova.ai/hubfs/logotype_sambanova_orange.png",
        ):
            st.write(f"{ans}")
            if st.session_state.show_sources:
                with st.expander("Sources"):
                    st.markdown(
                        f'<font size="2" color="grey">{source}</font>',
                        unsafe_allow_html=True,
                    )



def initialize_document_retrieval():
    if are_credentials_set():
        try:
            return DocumentRetrieval()
        except Exception as e:
            st.error(f"Failed to initialize DocumentRetrieval: {str(e)}")
            return None
    return None



def main():
    with open(CONFIG_PATH, 'r') as yaml_file:
        config = yaml.safe_load(yaml_file)

    prod_mode = config.get('prod_mode', False)
    default_collection = 'ekr_default_collection'
```

```
initialize_env_variables(prod_mode)

st.set_page_config(
    page_title="Reportīquē",
    page_icon="https://sambanova.ai/hubfs/logotype_sambanova_orange.png",
)

if "conversation" not in st.session_state:
    st.session_state.conversation = None
if "chat_history" not in st.session_state:
    st.session_state.chat_history = []
if "show_sources" not in st.session_state:
    st.session_state.show_sources = True
if "sources_history" not in st.session_state:
    st.session_state.sources_history = []
if "vectorstore" not in st.session_state:
    st.session_state.vectorstore = None
if 'input_disabled' not in st.session_state:
    st.session_state.input_disabled = True
if 'document_retrieval' not in st.session_state:
    st.session_state.document_retrieval = None

st.title(":green[Reportīquē]")
st.subheader(":grey[Get your reports summarized]")
st.session_state.solutions_suggestions = ''

with st.sidebar:
    st.session_state.module_response = None
    if not are_credentials_set():
```

```
        url, api_key = env_input_fields()
        if st.button("Save Credentials", key="save_credentials_sidebar"):
            message = save_credentials(url, api_key, prod_mode)
            st.success(message)
            st.rerun()

    if are_credentials_set():
        if st.session_state.document_retrieval is None:
            st.session_state.document_retrieval = initialize_document_retrieval()

    if st.session_state.document_retrieval is not None:
        st.markdown("**Purpose:**")
        st.markdown("Generates detailed reports for IT employees by analyzing and processing the provided code files, utilizing document retrieval and automated content generated technique.")
        st.markdown("**Follow the instructions.**")
        st.markdown("**1. Upload the files**")
        datasource_options = ["Upload files (create new vector db)"]
        if not prod_mode:
            datasource_options.append("Use existing vector db")

        datasource = datasource_options[0]

        if "Upload" in datasource:
            if config.get('pdf_only_mode', False):
                docs = st.file_uploader(
                    "Upload PDF files", accept_multiple_files=True, type=["pdf"]
                )
            else:
                docs = st.file_uploader(
```

```
            "Add files", accept_multiple_files=True,
            type=[".eml", ".html", ".json", ".md", ".msg", ".rst", ".rtf", ".txt", ".xml", ".png", ".jpg",
                ".jpeg", ".tiff", ".bmp", ".heic", ".csv", ".doc", ".docx", ".epub", ".odt", ".pdf",
                ".ppt", ".pptx", ".tsv", ".xlsx", ".py", ".c", ".h", ".ino", ".cpp", ".java", ".javac", ".pyc"]
        )
    st.markdown("**2. Process your documents**")
    st.markdown(
        "**Note:** Depending on the size and number of your documents, this could take several
minutes"
    )
    st.markdown("Create database")
    if st.button("Process"):
        with st.spinner("Processing"):
            try:
                text_chunks = st.session_state.document_retrieval.parse_doc(docs)
                embeddings = st.session_state.document_retrieval.load_embedding_model()
                collection_name = default_collection if not prod_mode else None
                vectorstore = st.session_state.document_retrieval.create_vector_store(text_chunks,
                                                                                      embeddings,
                                                                                      output_db=None,
                                                                                      collection_name=collection_name)
                st.session_state.vectorstore = vectorstore
                st.session_state.document_retrieval.init_retriever(vectorstore)
                st.session_state.conversation =
st.session_state.document_retrieval.get_qa_retrieval_chain()
                st.toast(f"File uploaded! Now the Task completed will be automatically filled",
icon='🎉')
                st.session_state.input_disabled = False
```

```
            prompt = "suggest DISTINCT titles for all the programs that you have found and also for those programs that are not explicitly mentioned. NUMBER THEM"
            response = st.session_state.conversation.invoke({"question": prompt})
            st.session_state.tasks = []
            response = response['answer']

            response = response.split('\n')
            for i in response:
               x = re.search("(\d).(\s)(.+)", i)
               if x != None:
                  __task__ = x.group(3)
                  if __task__ not in st.session_state.tasks:
                     st.session_state.tasks.append(__task__)

            module_prompt = "Give me the detailed analysis of the context without mentioning the coding"
            response = st.session_state.conversation.invoke({"question": module_prompt})

            response_processed = ''
            for i in response['answer'].split('\n')[3:-1]:
               response_processed += i

            st.session_state.module_response = response_processed

            prompt_solutions = "based on your understanding gimme solutions to improve the coding, DO NOT INCLUDE CODING"
            response = st.session_state.conversation.invoke({"question": prompt_solutions})
            response = response['answer'].split('\n')[1:]

            st.session_state.solutions_suggestions = ''
```

```
            for i in response:
                x = re.search("(\d).(\s)(.+)", i)
                if x != None:
                    __task__ = x.group(0)
                    st.session_state.solutions_suggestions += (__task__ + '\n')



        except Exception as e:
            st.error(f"An error occurred while processing: {str(e)}")

st.markdown("**3. Ask questions about your data!**")

with st.expander("Additional settings", expanded=False):
    st.markdown("**Interaction options**")
    st.markdown(
        "**Note:** Toggle these at any time to change your interaction experience"
    )
    show_sources = st.checkbox("Show sources", value=True, key="show_sources")

    st.markdown("**Reset chat**")
    st.markdown(
        "**Note:** Resetting the chat will clear all conversation history"
    )
    if st.button("Reset conversation"):
        st.session_state.chat_history = []
        st.session_state.sources_history = []
        st.toast(
```

```
                "Conversation reset. The next response will clear the history on the screen"
            )
st.markdown("<hr style='background-color:green;color:green;height:2px'>", unsafe_allow_html=True)

col1, col2 = st.columns(2)
with col1:
    name = st.text_input("Name")

with col2:
    user_id = st.text_input("ID")

col5, col6 = st.columns(2)

roles = [
"Software Development",
"Project & Product Management",
"Data & Analytics",
"Cloud & Infrastructure",
"Cybersecurity",
"Quality Assurance (QA)",
"UI/UX & Design",
"IT Support",
"Enterprise Solutions & Consulting",
"Emerging Technologies",
"Leadership & Executive Roles"
]

sub_roles = {
    "Software Development": [
```

```
        "Software Engineer", "Full Stack Developer", "Backend Developer",
        "Frontend Developer", "Mobile App Developer", "DevOps Engineer", "Game Developer", "Other"
    ],
    "Project & Product Management": [
        "Project Manager", "Product Manager", "Scrum Master",
        "Technical Program Manager", "Business Analyst", "Other"
    ],
    "Data & Analytics": [
        "Data Scientist", "Data Engineer", "Data Analyst",
        "Machine Learning Engineer", "AI Engineer", "Business Intelligence Analyst", "Database Administrator (DBA)", "Other"
    ],
    "Cloud & Infrastructure": [
        "Cloud Engineer", "Cloud Architect", "System Administrator", "Network Engineer", "Security Engineer", "Other"
    ],
    "Cybersecurity": [
        "Cybersecurity Analyst", "Penetration Tester (Ethical Hacker)",
        "Security Architect", "Incident Response Analyst", "Security Consultant", "Other"
    ],
    "Quality Assurance (QA)": [
        "QA Engineer", "Automation Tester", "Manual Tester", "Other"
    ],
    "UI/UX & Design": [
        "UI/UX Designer", "User Experience Researcher", "Product Designer", "Interaction Designer", "Other"
    ],
    "IT Support": [
        "Technical Support Engineer", "Help Desk Technician", "IT Support Specialist", "Other"
    ],
```

```
    "Enterprise Solutions & Consulting": [
        "ERP Consultant (e.g., SAP, Oracle)", "Salesforce Administrator/Developer",
        "IT Consultant", "Solutions Architect", "Other"
    ],
    "Emerging Technologies": [
        "Blockchain Developer", "AR/VR Developer", "IoT Engineer", "Robotics Engineer", "Other"
    ],
    "Leadership & Executive Roles": [
        "Chief Information Officer (CIO)", "Chief Technology Officer (CTO)",
        "IT Director", "Technical Lead", "Other"
    ]
}

with col5:
    selected_role = st.selectbox("Select Role", roles)

with col6:
    selected_sub_role = st.selectbox("Select Sub-Role", sub_roles[selected_role])

col3, col4 = st.columns([3, 1])

with col3:
    project_name = st.text_input("Project Name")

with col4:
    start_date = st.date_input("Report Date", date.today())

st.markdown("<hr style='background-color:green;color:green;height:2px'>", unsafe_allow_html=True)
```

```
if 'tasks' not in st.session_state:
    st.session_state.tasks = []

def delete_task(task_to_remove):
    if task_to_remove in st.session_state.tasks:
        st.session_state.tasks.remove(task_to_remove)

st.subheader("Tasks Completed")
task_selected = {}

if len(st.session_state.tasks) == 0:
    st.write("No tasks has been completed.")

for task in st.session_state.tasks:
    col7, col8 = st.columns(2)

    with col7:
        st.markdown(f"- {task}", unsafe_allow_html=True)

    with col8:
        if st.button("Delete", key=f"delete_{task}"):
            delete_task(task)
            st.experimental_rerun()

col9, col10 = st.columns(2)
with col9:
    new_task = st.text_input("Enter a new task")

with col10:
```

```
        st.markdown(" ")
        st.markdown(" ")
        if st.button("Add Task"):
            if new_task and new_task not in st.session_state.tasks:
                st.session_state.tasks.append(new_task)
                st.success(f"Task '{new_task}' added!")
                st.experimental_rerun()

    st.write("### Detailed Analysis")
    st.text_area("Your detailed analysis will be shown here.", value=st.session_state.module_response, height=250, disabled=True)

    with st.expander("Recommended Solutions"):
        st.text_area("Some recommendations to improve the programs will be displayed here.", height=250, value=st.session_state.solutions_suggestions, disabled=True)

    with st.expander("Challenges"):
        challenges = st.text_area("Describe any challenges faced:", height=150)

if __name__ == "__main__":
    os.environ['SAMBANOVA_API_KEY'] = "d18422d8-b21a-4db9-a5e5-55344715bd4a"
    main()
```