Source code for langchain_community.vectorstores

```
"""
Pathway Vector Store client.


The Pathway Vector Server is a pipeline written in the Pathway framweork which
all files in a given folder, embeds them, and builds a vector index. The pipel
to changes in source files, automatically updating appropriate index entries.

The PathwayVectorClient implements the LangChain VectorStore interface and que
PathwayVectorServer to retrieve up-to-date documents.

You can use the client with managed instances of Pathway Vector Store, or run
instance as described at https://pathway.com/developers/user-guide/llm-xpack/v

"""

import json
import logging
from typing import Any, Callable, Iterable, List, Optional, Tuple

import requests
from langchain_core.documents import Document
from langchain_core.embeddings import Embeddings
from langchain_core.vectorstores import VectorStore


# Copied from https://github.com/pathwaycom/pathway/blob/main/python/pathway/x
# to remove dependency on Pathway library.
class _VectorStoreClient:
    def __init__(
        self,
        host: Optional[str] = None,
        port: Optional[int] = None,
        url: Optional[str] = None,
    ):
        """
        A client you can use to query :py:class:`VectorStoreServer`.

        Please provide aither the `url`, or `host` and `port`.

        Args:
            - host: host on which `:py:class:`VectorStoreServer` listens
            - port: port on which `:py:class:`VectorStoreServer` listens
            - url: url at which `:py:class:`VectorStoreServer` listens
        """
        err = "Either (`host` and `port`) or `url` must be provided, but not b
        if url is not None:
            if host or port:
                raise ValueError(err)
            self.url = url
        else:
            if host is None:
                raise ValueError(err)
            port = port or 80
            self.url = f"http://{host}:{port}"

    def query(
```

```
        self, query: str, k: int = 3, metadata_filter: Optional[str] = None
    ) -> List[dict]:
        """
        Perform a query to the vector store and fetch results.

        Args:
            - query:
            - k: number of documents to be returned
            - metadata_filter: optional string representing the metadata filte
                in the JMESPath format. The search will happen only for docume
                satisfying this filtering.
        """

        data = {"query": query, "k": k}
        if metadata_filter is not None:
            data["metadata_filter"] = metadata_filter
        url = self.url + "/v1/retrieve"
        response = requests.post(
            url,
            data=json.dumps(data),
            headers={"Content-Type": "application/json"},
            timeout=3,
        )
        responses = response.json()
        return sorted(responses, key=lambda x: x["dist"])

    # Make an alias
    __call__ = query

    def get_vectorstore_statistics(self) -> dict:
        """Fetch basic statistics about the vector store."""

        url = self.url + "/v1/statistics"
        response = requests.post(
            url,
            json={},
            headers={"Content-Type": "application/json"},
        )
        responses = response.json()
        return responses

    def get_input_files(
        self,
        metadata_filter: Optional[str] = None,
        filepath_globpattern: Optional[str] = None,
    ) -> list:
        """
        Fetch information on documents in the vector store.

        Args:
            metadata_filter: optional string representing the metadata filteri
                in the JMESPath format. The search will happen only for docume
                satisfying this filtering.
            filepath_globpattern: optional glob pattern specifying which docum
                will be searched for this query.
        """
        url = self.url + "/v1/inputs"
        response = requests.post(
            url,
```

```
            json={
                "metadata_filter": metadata_filter,
                "filepath_globpattern": filepath_globpattern,
            },
            headers={"Content-Type": "application/json"},
        )
        responses = response.json()
        return responses



[docs]
class PathwayVectorClient(VectorStore):
    """
    VectorStore connecting to Pathway Vector Store.
    """


[docs]
    def __init__(
        self,
        host: Optional[str] = None,
        port: Optional[int] = None,
        url: Optional[str] = None,
    ) -> None:
        """
        A client you can use to query Pathway Vector Store.

        Please provide aither the `url`, or `host` and `port`.

        Args:
            - host: host on which Pathway Vector Store listens
            - port: port on which Pathway Vector Store listens
            - url: url at which Pathway Vector Store listens
        """
        self.client = _VectorStoreClient(host, port, url)




[docs]
    def add_texts(
        self,
        texts: Iterable[str],
        metadatas: Optional[List[dict]] = None,
        **kwargs: Any,
    ) -> List[str]:
        """Pathway is not suitable for this method."""
        raise NotImplementedError(
            "Pathway vector store does not support adding or removing texts"
            " from client."
        )




[docs]
    @classmethod
    def from_texts(
```

```
        cls,
        texts: List[str],
        embedding: Embeddings,
        metadatas: Optional[List[dict]] = None,
        **kwargs: Any,
    ) -> "PathwayVectorClient":
        raise NotImplementedError(
            "Pathway vector store does not support initializing from_texts."
        )



[docs]
    def similarity_search(
        self, query: str, k: int = 4, **kwargs: Any
    ) -> List[Document]:
        metadata_filter = kwargs.pop("metadata_filter", None)
        if kwargs:
            logging.warning(
                "Unknown kwargs passed to PathwayVectorClient.similarity_searc
                kwargs,
            )
        rets = self.client(query=query, k=k, metadata_filter=metadata_filter)

        return [
            Document(page_content=ret["text"], metadata=ret["metadata"]) for r
        ]



[docs]
    def similarity_search_with_score(
```