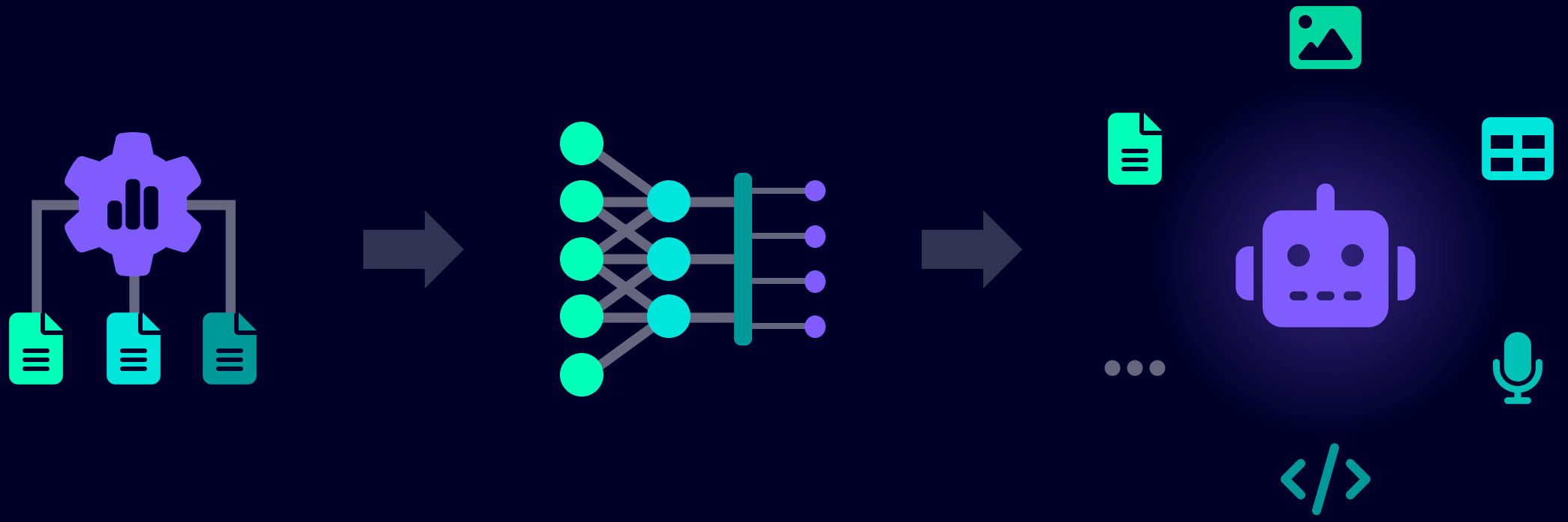
LLM Engineering on GitLab with CI Services

Siemens Open Source Summit 2025

Sigurd Spieckermann

SIEMENS

The impressive evolution from NLP to advanced AI applications



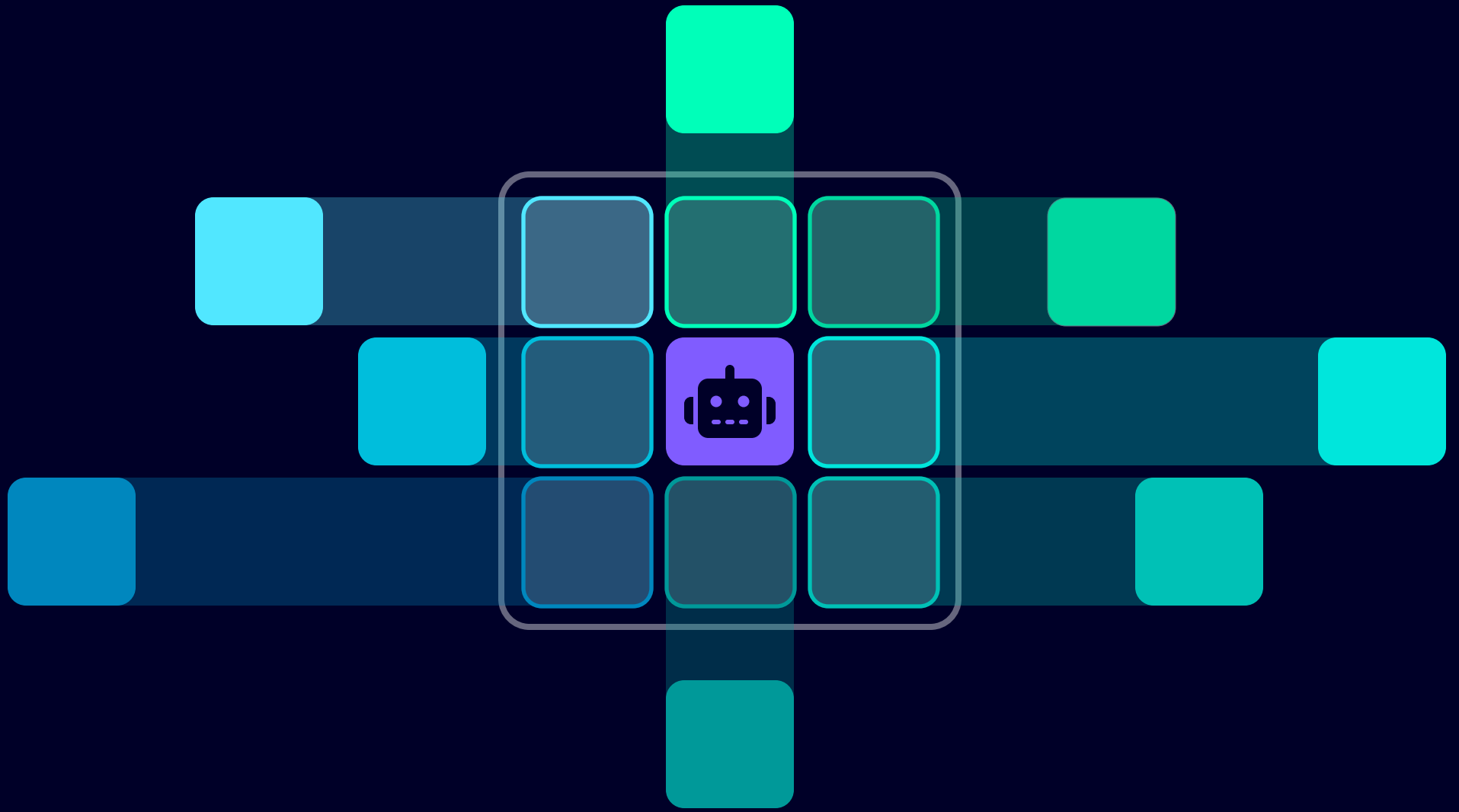
Traditional NLP

Deep Learning for NLP

Chatbots, Agents & more

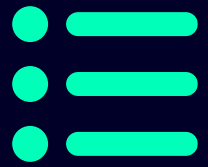
SIEMENS

An LLM application is software with an LLM component



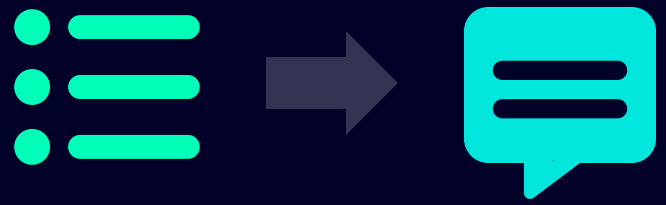
Prompt engineering guides LLM responses to desired outcomes without (re)training

Prompt engineering guides LLM responses to desired outcomes without (re)training



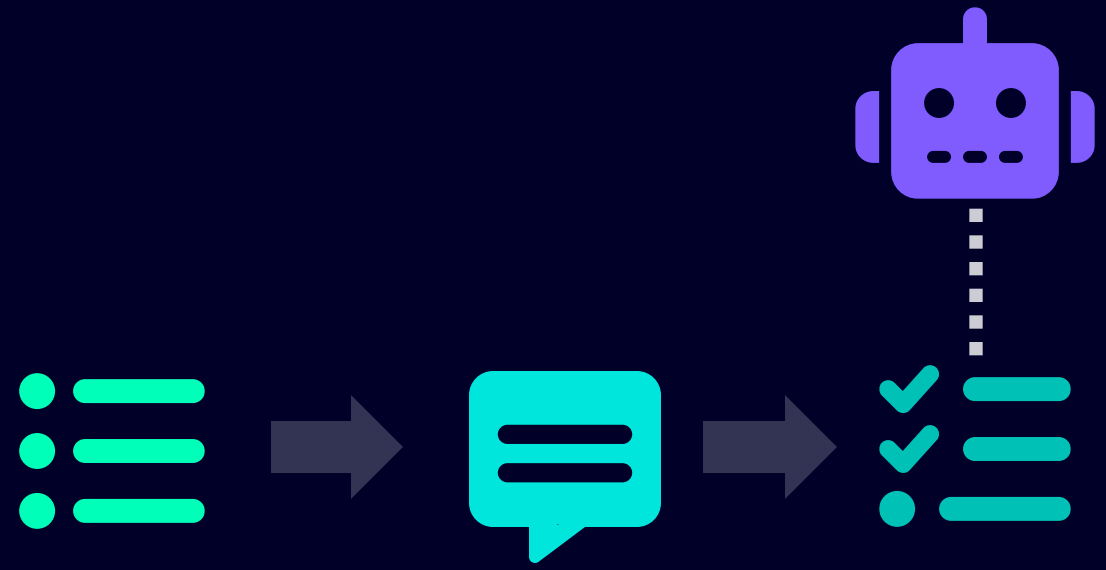
Create test cases

Prompt engineering guides LLM responses to desired outcomes without (re)training



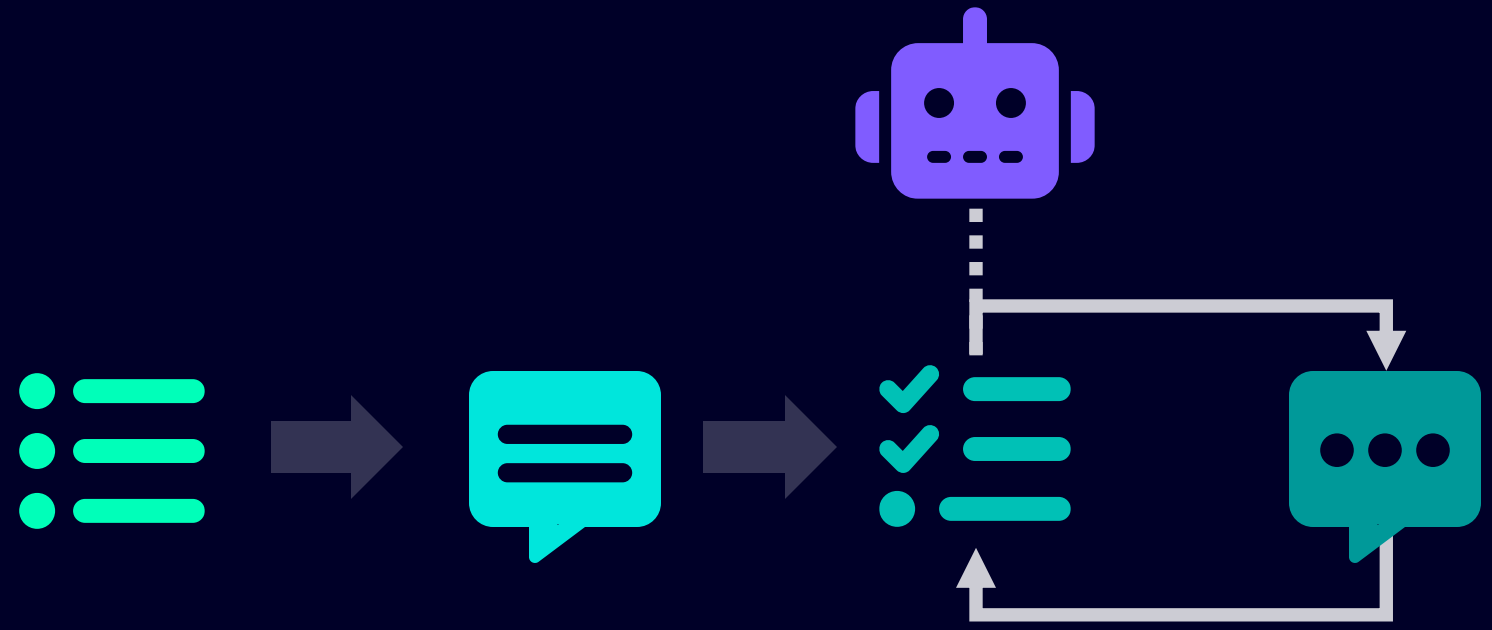
Create test cases Engineer prompt

Prompt engineering guides LLM responses to desired outcomes without (re)training

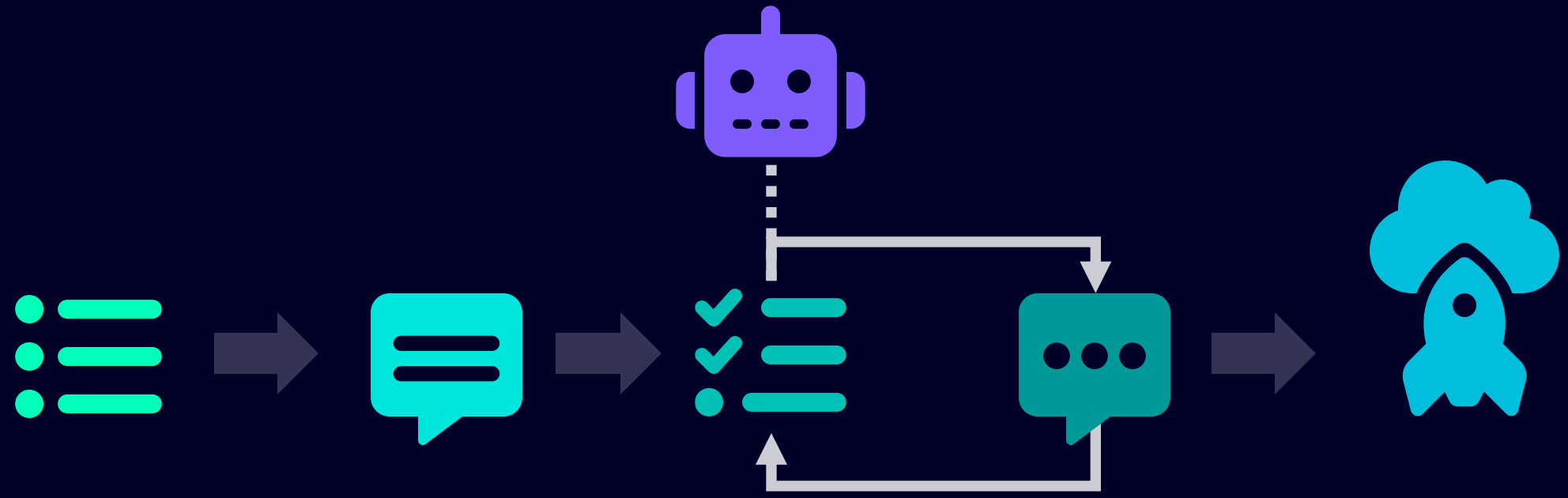


Create test cases Engineer prompt Evaluate

Prompt engineering guides LLM responses to desired outcomes without (re)training



Prompt engineering guides LLM responses to desired outcomes without (re)training



SIEMENS

Implications of external LLM inference services on DevOps

SIEMENS

Implications of external LLM inference services on DevOps

Infra & Services
Management

Implications of external LLM inference services on DevOps

Infra & Services Management

IAM & Security for Developers

Implications of external LLM inference services on DevOps

Infra & Services Management

IAM & Security for Developers

IAM & Security for CI/CD

Implications of external LLM inference services on DevOps

Infra & Services Management

IAM & Security for Developers

IAM & Security for CI/CD

Reproducibility

Implications of external LLM inference services on DevOps

Infra & Services Management

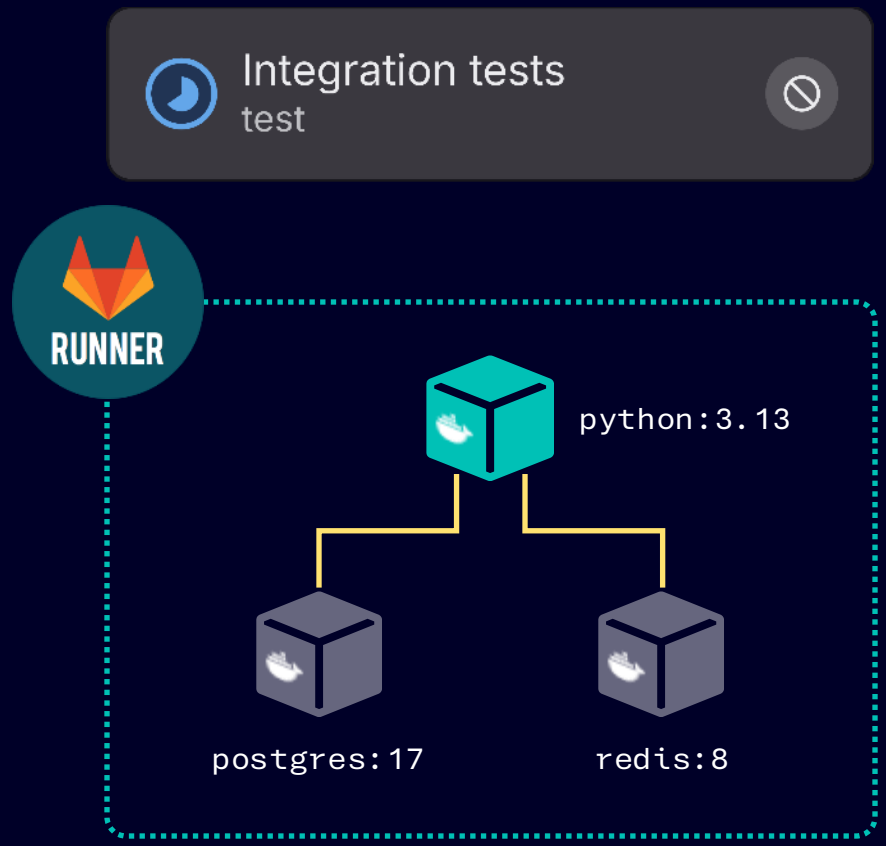
IAM & Security for Developers

IAM & Security for CI/CD

Reproducibility

Flexibility

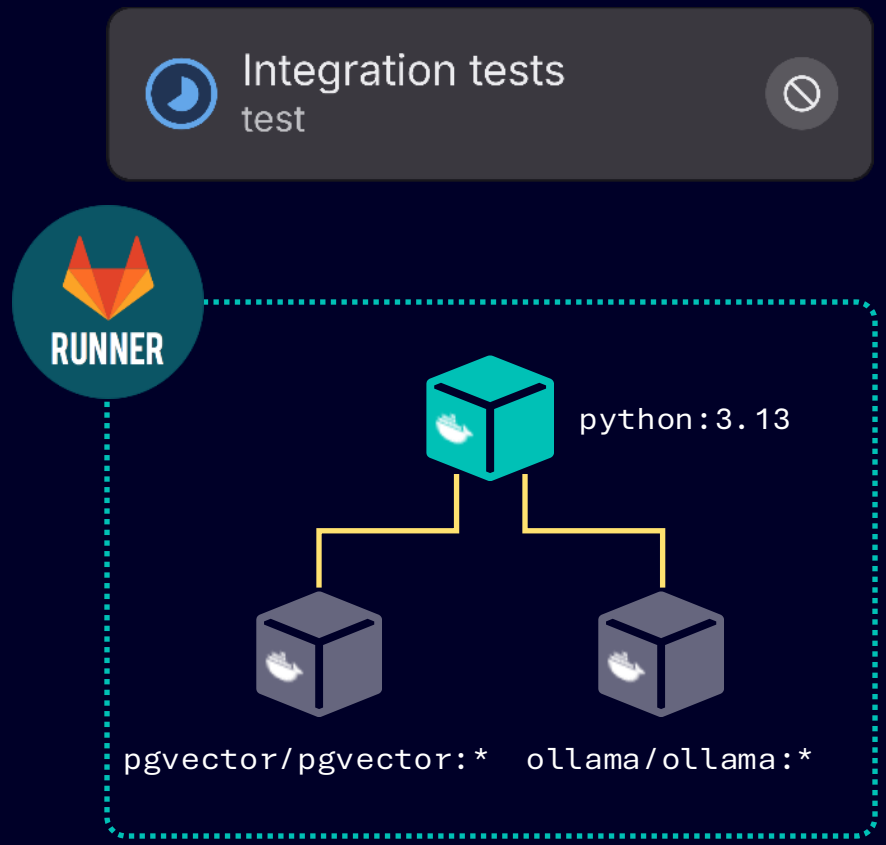
GitLab CI services for seamless integration testing with containerized services




```yaml
Integration tests:
  stage: test

  # Service containers
  services:
    - name: postgres:17
      alias: postgres
      variables:
        POSTGRES_DB: test
        POSTGRES_USER: test
        POSTGRES_PASSWORD: test
    - redis:8

  # Main job
  image: python:3.13
  before_script:
    - pip install -r requirements.txt
  script:
    - pytest --pgvector-host pgvector --ollama-url http://ollama:11434
```


SIEMENS

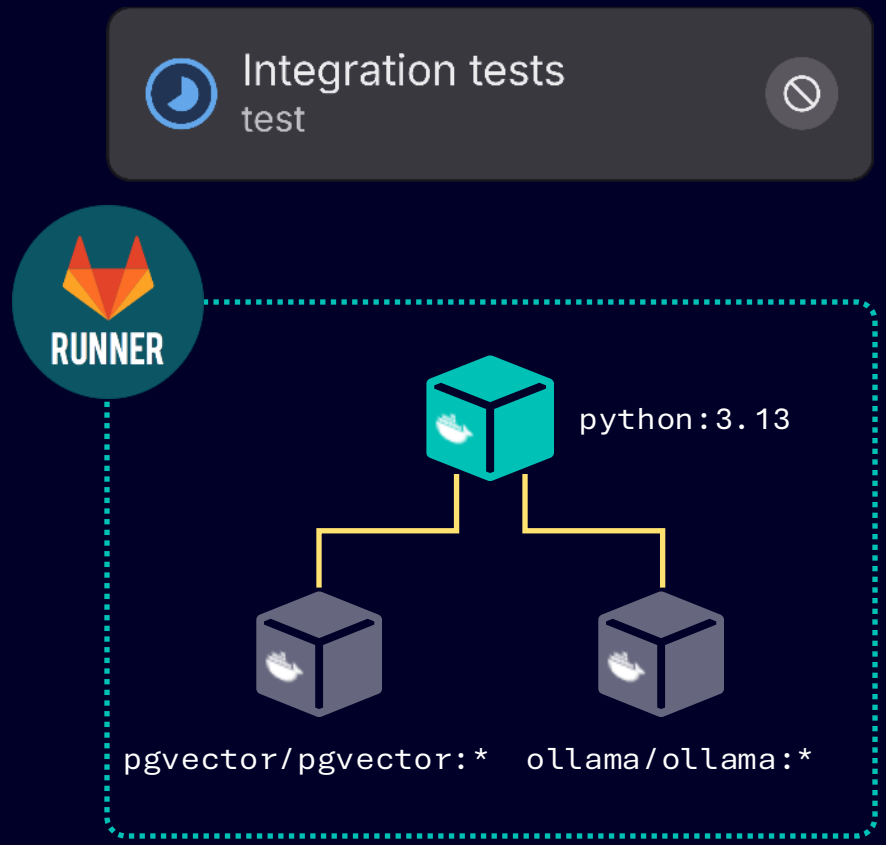
GitLab CI services for seamless integration testing of LLM applications




```
Integration tests:
  stage: test

  # Service containers
  services:
-   - name: postgres:17
-     alias: postgres
+   - name: pgvector/pgvector:0.8.0-pg17
+     alias: pgvector
      variables:
        POSTGRES_DB: test
        POSTGRES_USER: test
        POSTGRES_PASSWORD: test
-   - redis:8
+   - name: ollama/ollama:0.7.1
+     alias: ollama

  # Main job
  image: python:3.13
  before_script:
    - pip install -r requirements.txt
  script:
-   - pytest --pgvector-host pgvector --ollama-url http://ollama:11434
+   - pytest --postgres-host postgres --redis-host redis
```


SIEMENS

GitLab CI services for seamless integration testing of LLM applications




```yaml
Integration tests:
  stage: test

  # Service containers
  services:
    - name: pgvector/pgvector:0.8.0-pg17
      alias: pgvector
      variables:
        POSTGRES_DB: test
        POSTGRES_USER: test
        POSTGRES_PASSWORD: test
    - name: ollama/ollama:0.7.1
      alias: ollama

  # Main job
  image: python:3.13
  before_script:
    - pip install -r requirements.txt
  script:
    - pytest --pgvector-host pgvector --ollama-url http://ollama:11434
```


SIEMENS

33 41

Search or go to...

Project

gitlab-runner

Pinned

Issues 3k

Merge requests 79

Manage

Plan

Code

Merge requests 79

Repository

Branches

Commits

Tags

Repository graph

Compare revisions

Locked files

Build

Deploy

Operate

Monitor

Analyze

Help

Add GPUs support for services

Edit Code

Merged **Sigurd Spieckermann** requested to merge `sisp/gitlab-runner:feat/…` into `main` 2 months ago

Overview 29 Commits 3 Pipelines 9 Changes 5

1 unresolved thread Mark as done

- [ ] Please check this box if this contribution uses AI-generated content (including content generated by GitLab Duo features) as outlined in the GitLab DCO & CLA. As a benefit of being a GitLab Community Contributor, you receive complimentary access to GitLab Duo.

What does this MR do?

This MR adds support for the Docker executor to expose GPUs to services. A new `service_gpus` string in the runner config allows configuring GPU exposure to services using the same syntax as the `gpus` string.

Based on our feature design discussion, I've decided to go ahead and propose an implementation according to variant 2.1.

I've tested this modification on a real test project and confirm that it's working.

Why was this MR needed?

Services that require GPUs have no access to these resources at the moment. See #38513 (closed) for more details.

What's the best way to test this MR?

1. Compile the `gitlab-runner` executable from this MR.
2. Register the runner using the Docker executor in a test project.
3. Add the `service_gpus` setting in the `config.toml` file:

```
[[runners]]
  (...)

  [runners.docker]
    (...)
    service_gpus = "all"
```

4. Add a test CI config in `.gitlab-ci.yml`:

```
variables:
  CI_DEBUG_SERVICES: "true"

test:
  services:
    - name: debian:latest
```

Assignee
Sigurd Spieckermann

2 Reviewers
All required approvals given
Isaac Durham
Axel von Bertoldi

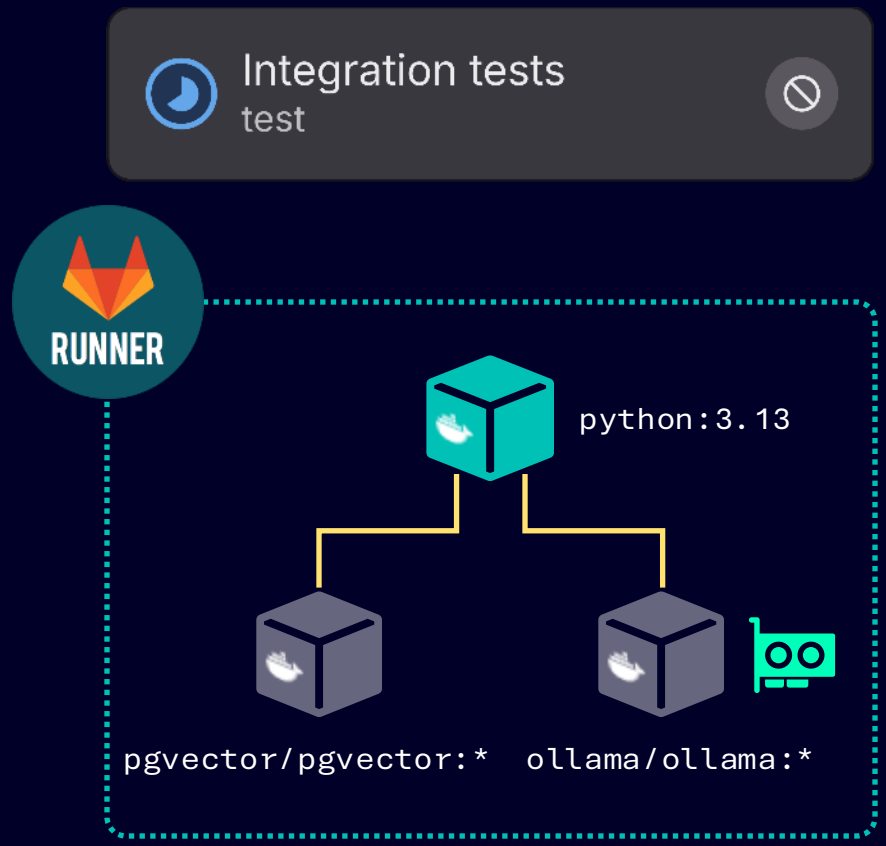
Labels
devops verify, group runner, section ci, tw triaged, type feature, workflow ready for review, Category:Runner Core, Community contribution, Technical Writing, documentation, linked-issue

Milestone
17.10 (expired)

Time tracking
No estimate or time spent

10 Participants
+ 2 more

GitLab CI services for seamless integration testing of LLM applications




```yaml
Integration tests:
  stage: test

  # Service containers
  services:
    - name: pgvector/pgvector:0.8.0-pg17
      alias: pgvector
      variables:
        POSTGRES_DB: test
        POSTGRES_USER: test
        POSTGRES_PASSWORD: test
    - name: ollama/ollama:0.7.1
      alias: ollama

  # Main job
  image: python:3.13
  before_script:
    - pip install -r requirements.txt
  script:
    - pytest --pgvector-host pgvector --ollama-url http://ollama:11434
```


SIEMENS

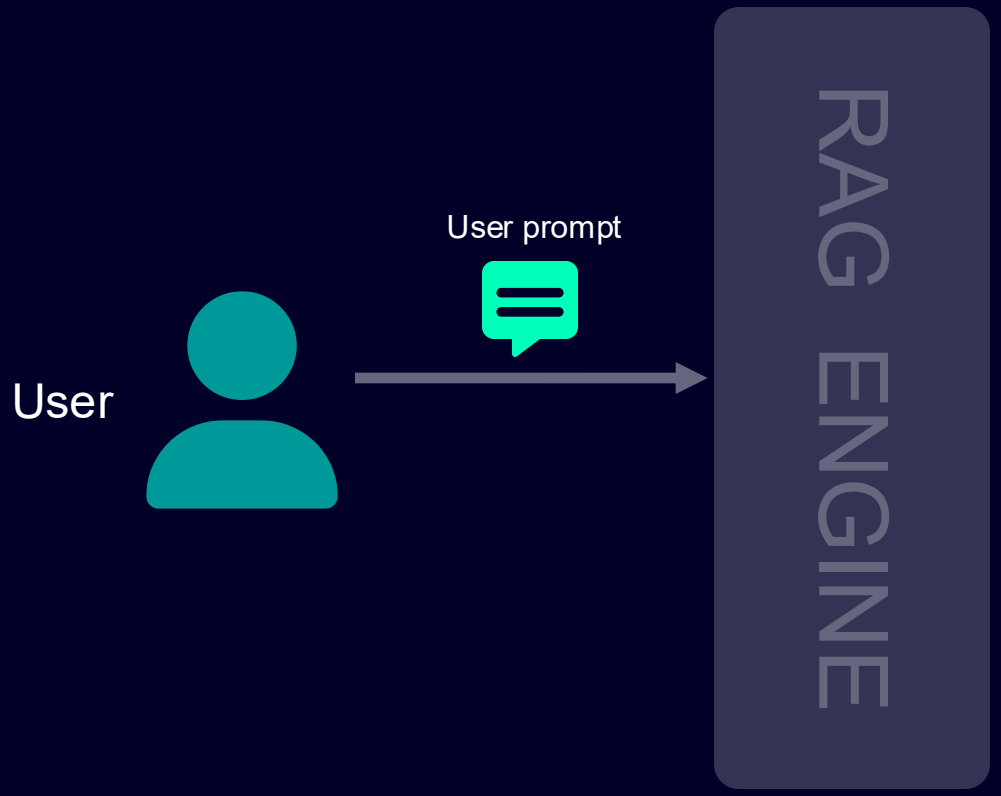
Example:
Retrieval-Augmented Generation

… with Python, LangChain, and DeepEval on GitLab

SIEMENS

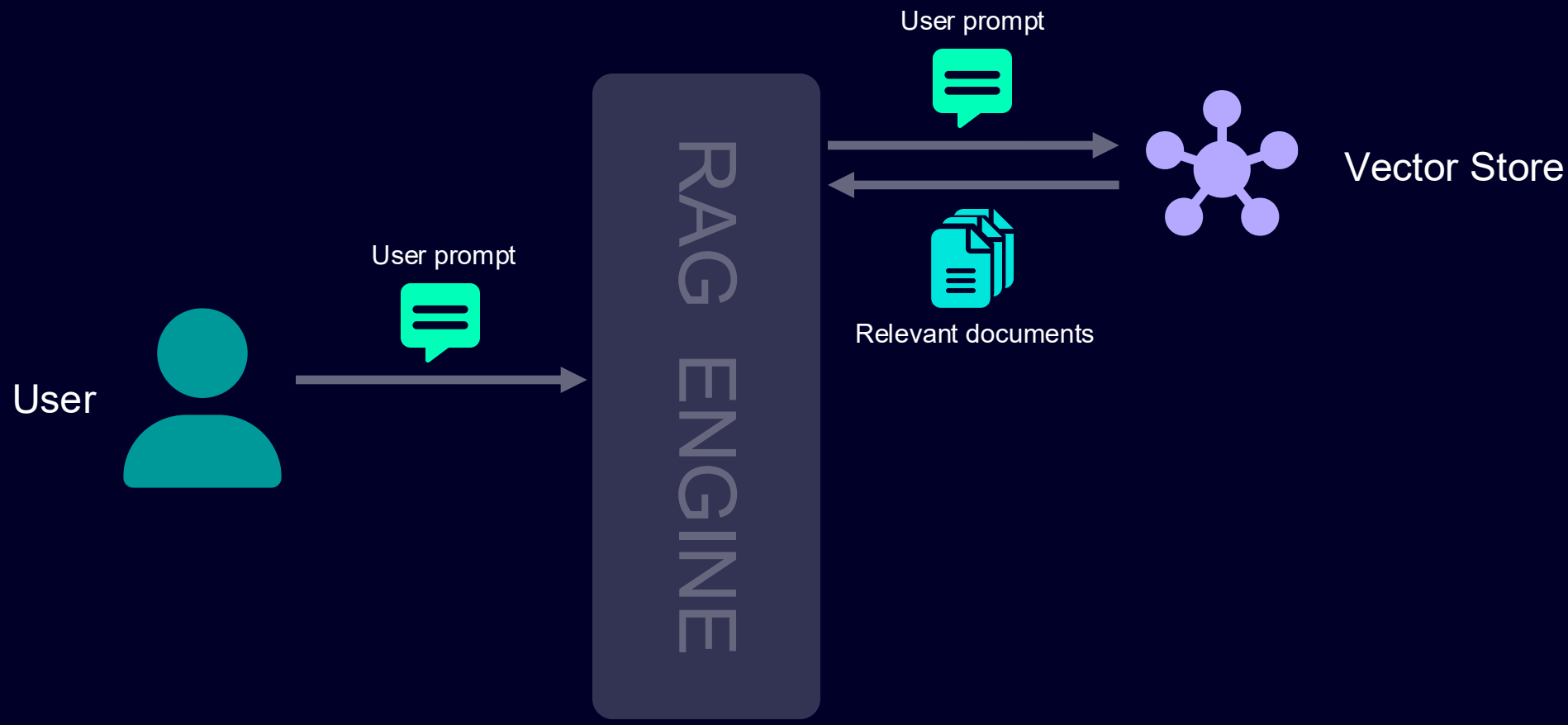
Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG)



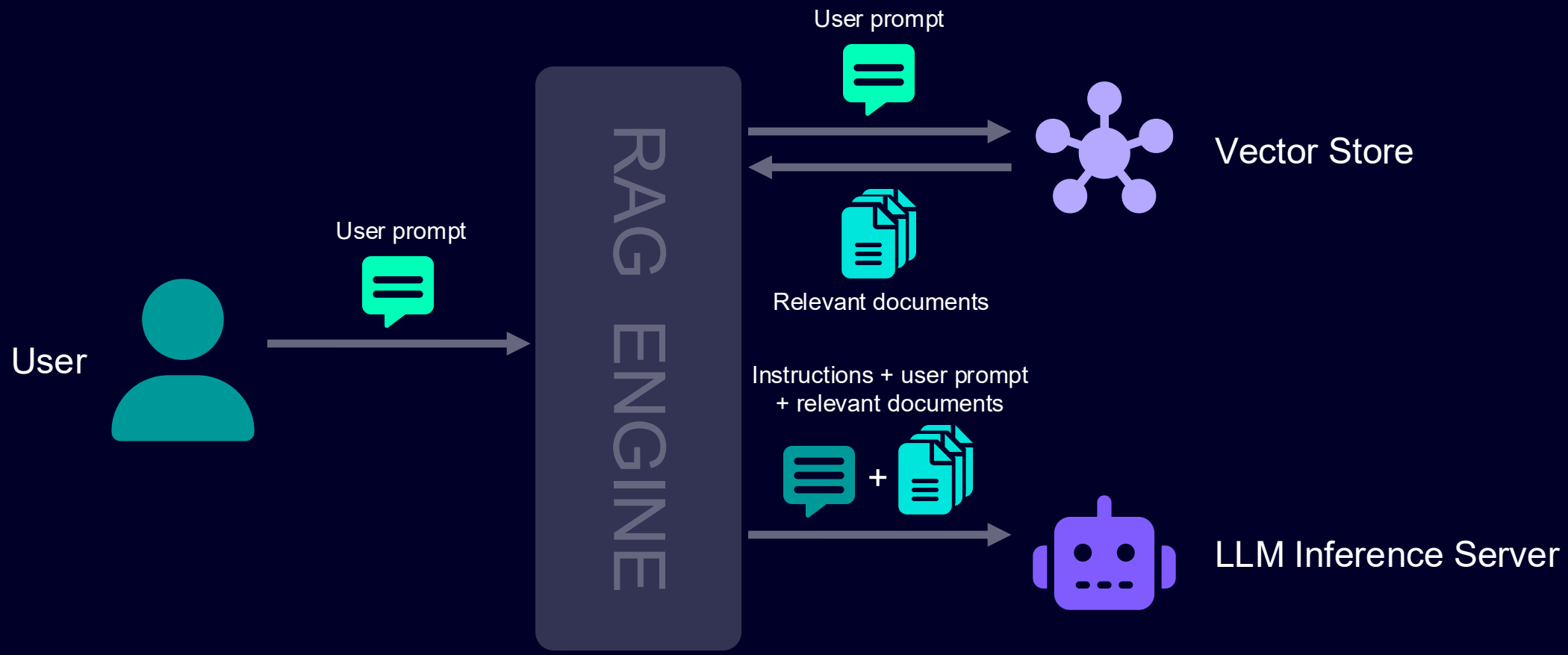
SIEMENS

Retrieval-Augmented Generation (RAG)



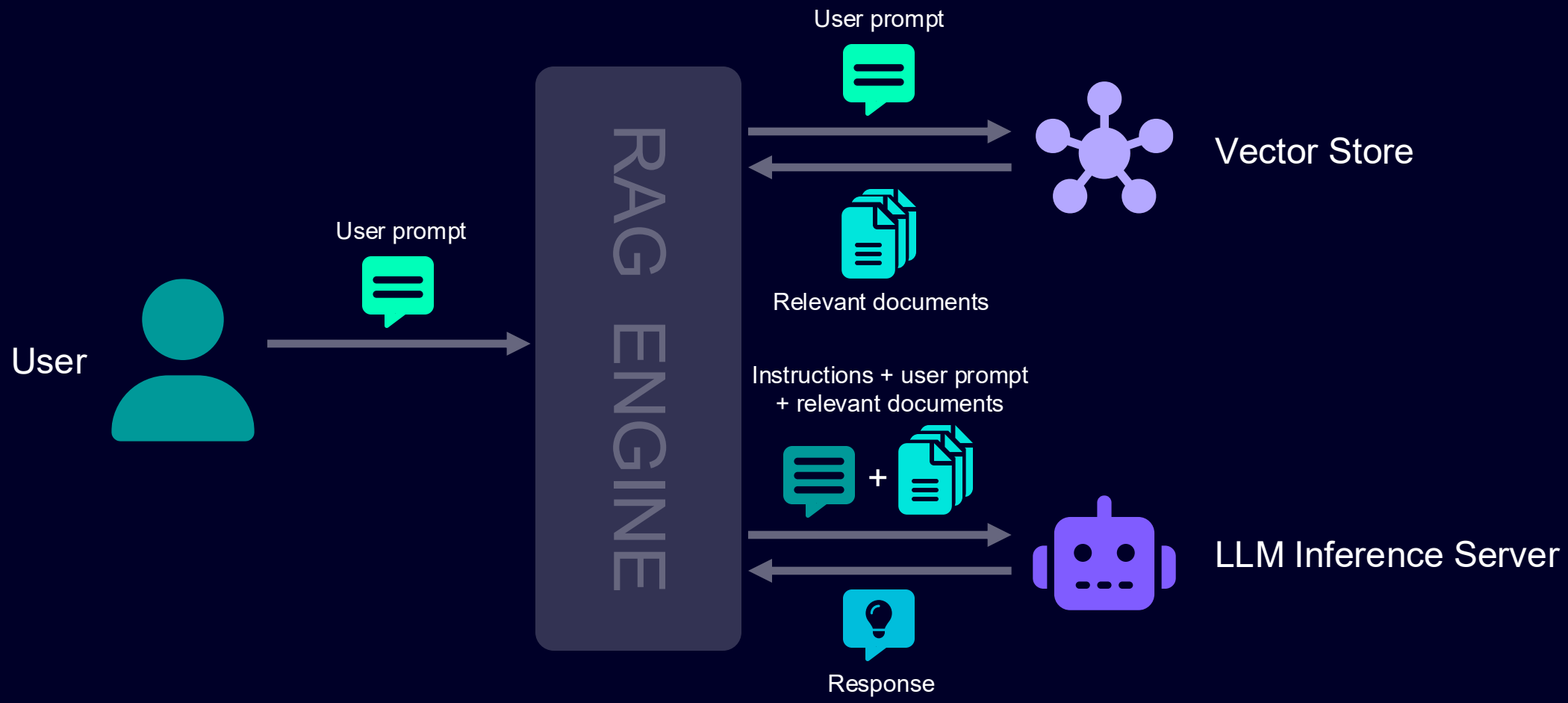
SIEMENS

Retrieval-Augmented Generation (RAG)



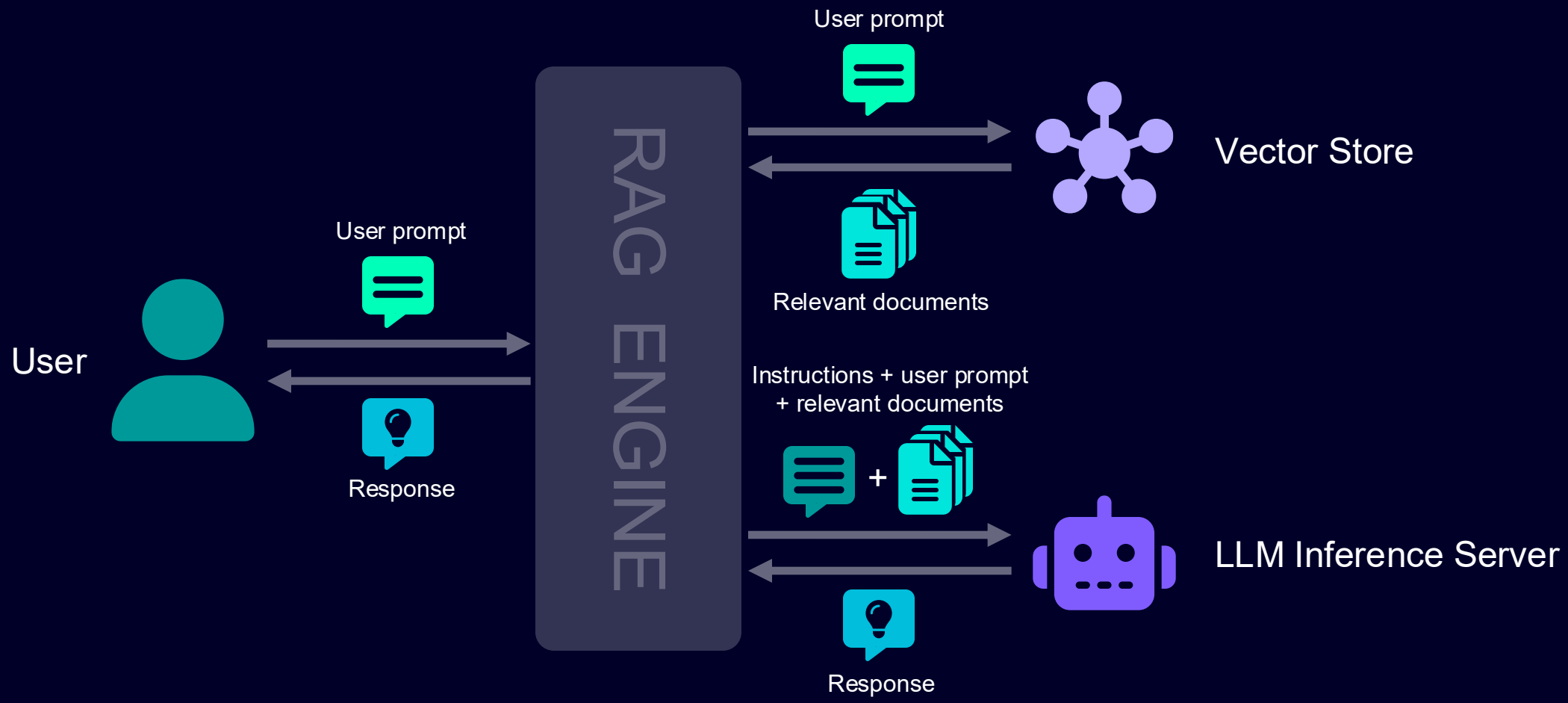
SIEMENS

Retrieval-Augmented Generation (RAG)



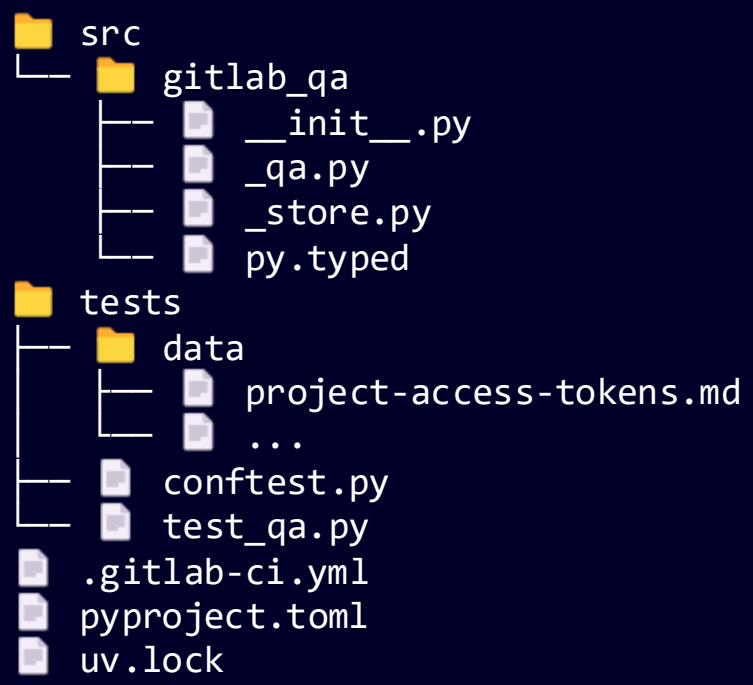
SIEMENS

Retrieval-Augmented Generation (RAG)



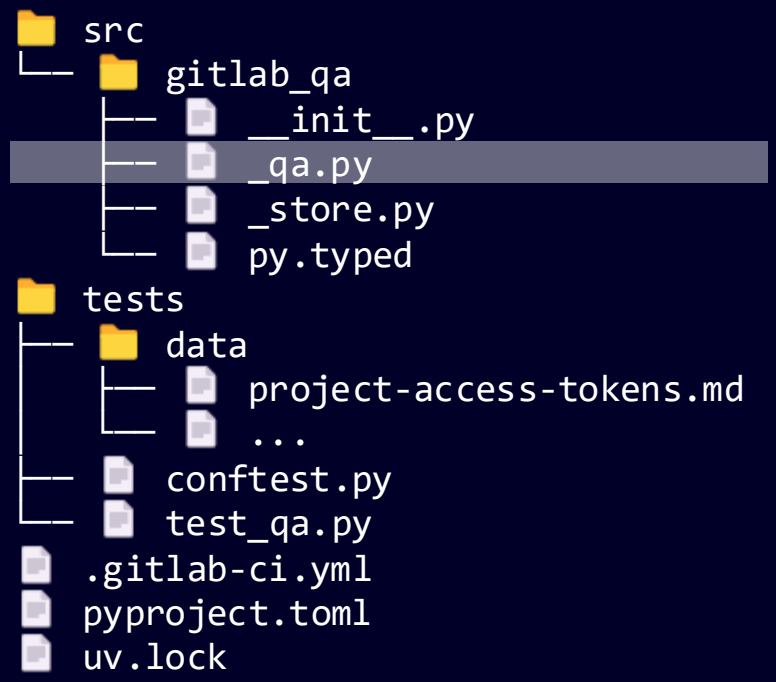
SIEMENS

GitLab Q&A demo project



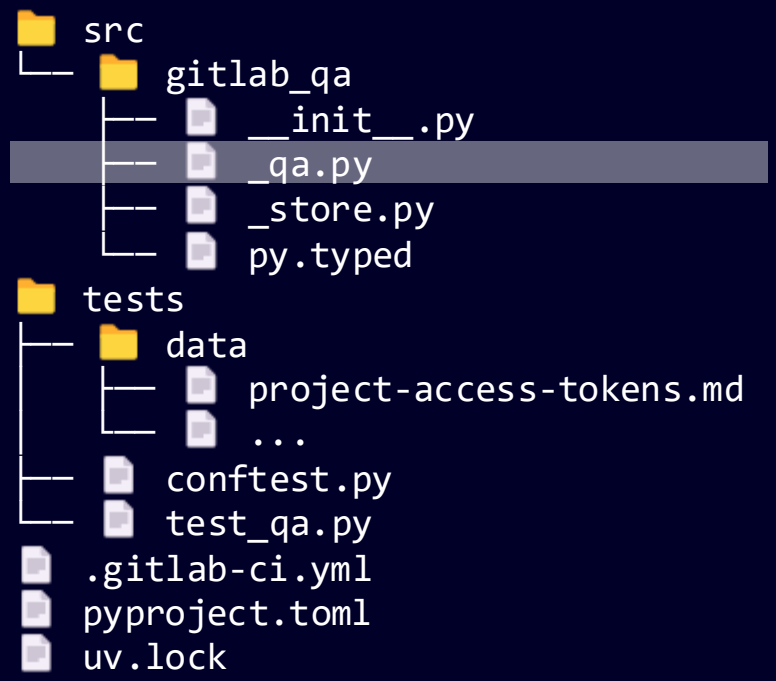
SIEMENS

GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = ...

    PROMPT: Final = ...

    ...
```

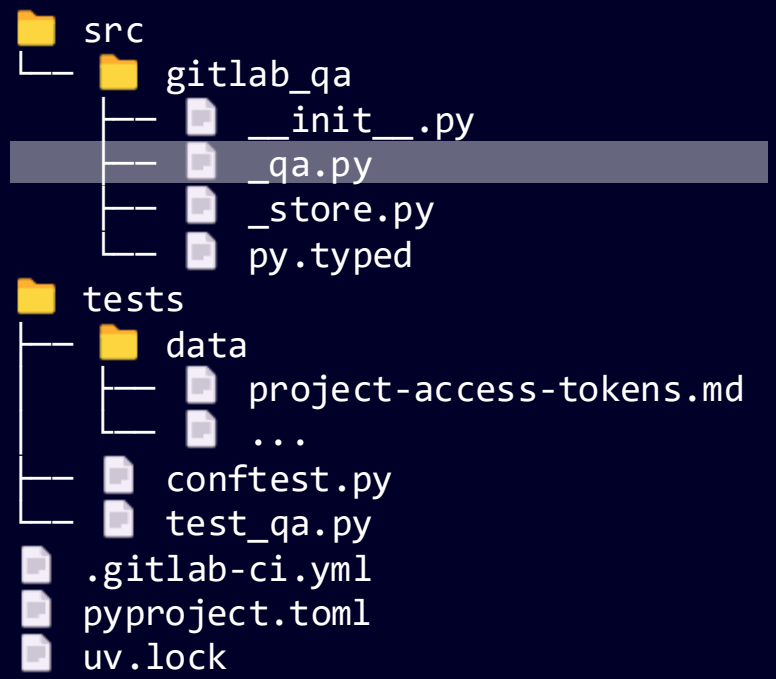
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = ...

    PROMPT: Final = ...

    ...
```


SIEMENS

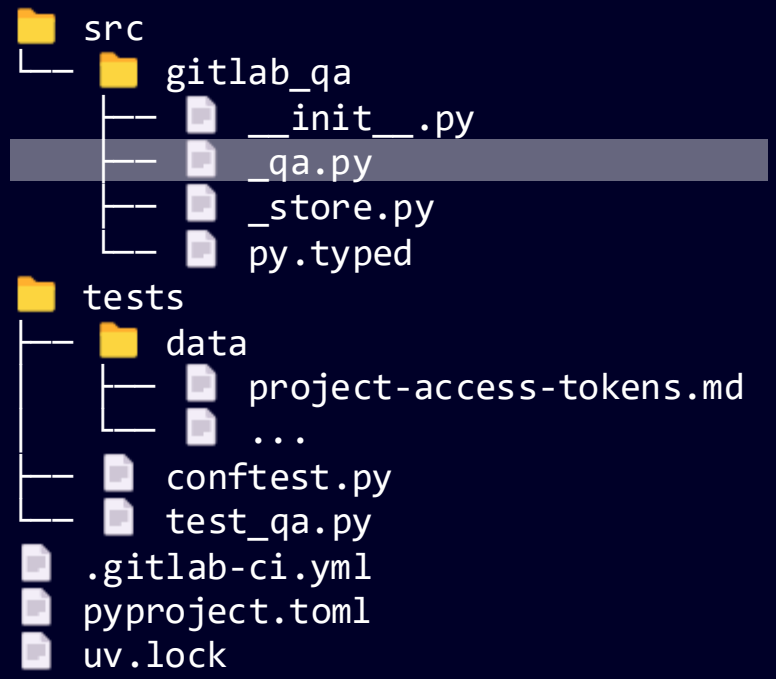
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ...

    ...
```


SIEMENS

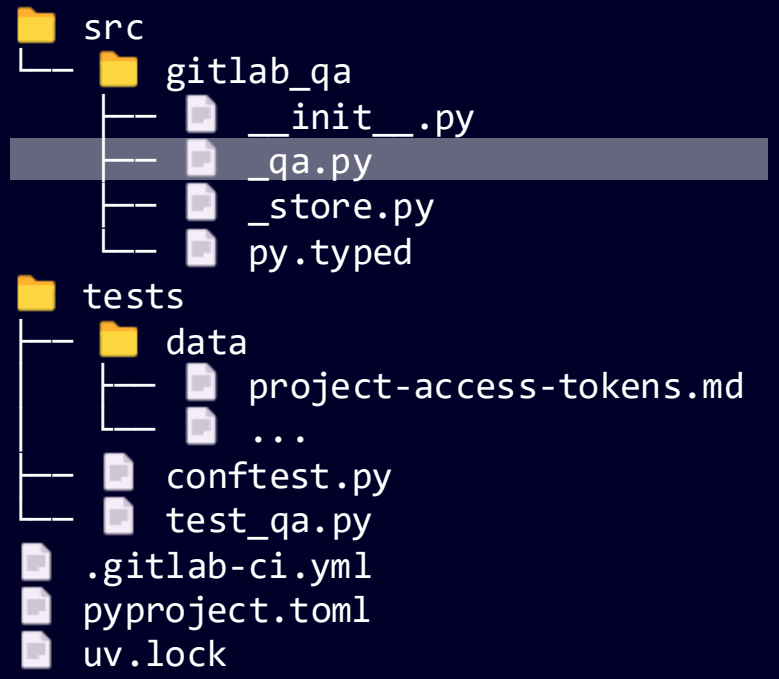
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ...

    ...
```


SIEMENS

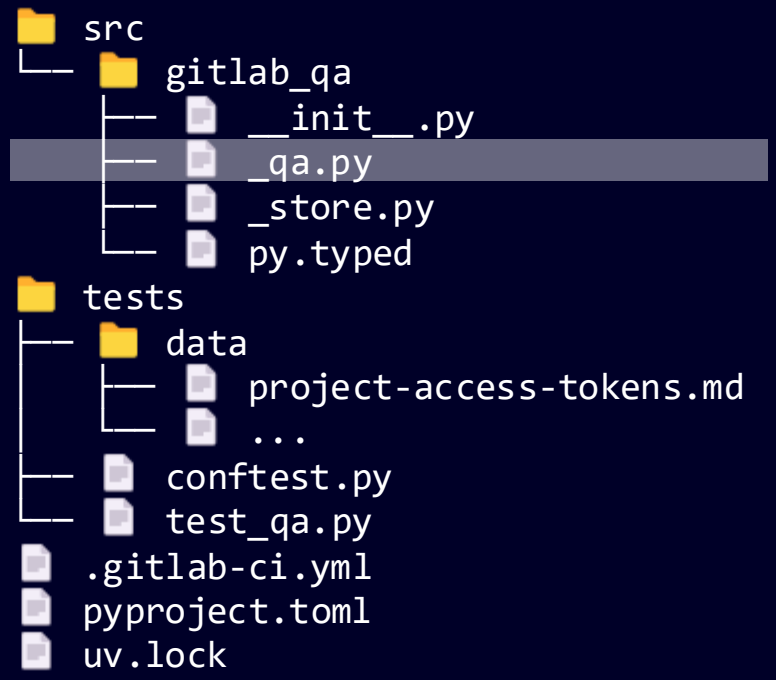
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(
        [
            SystemMessagePromptTemplate.from_template(
                ...
            ),
            HumanMessagePromptTemplate.from_template(
                ...
            ),
        ]
    )

    ...
```


SIEMENS

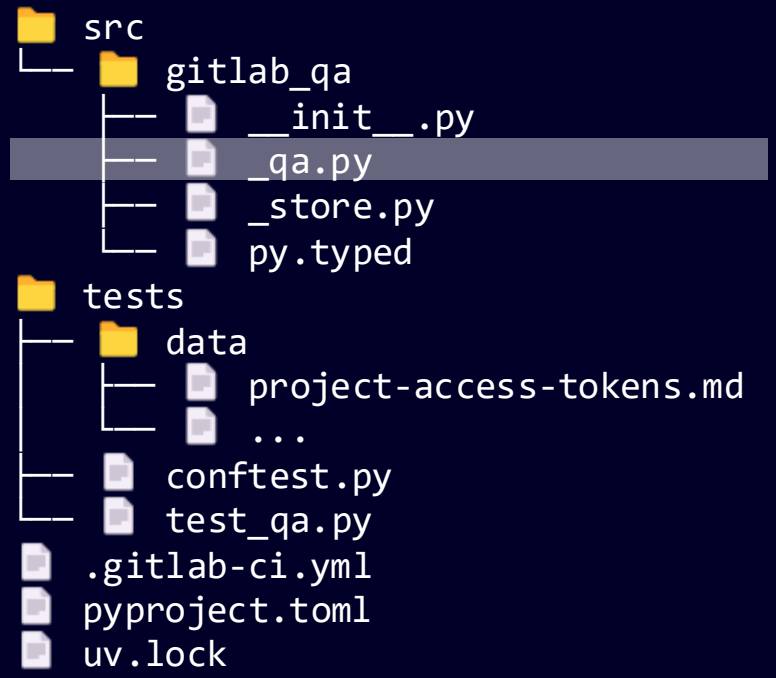
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(
        [
            SystemMessagePromptTemplate.from_template(
                ...
            ),
            HumanMessagePromptTemplate.from_template(
                ...
            ),
        ]
    )

    ...
```


SIEMENS

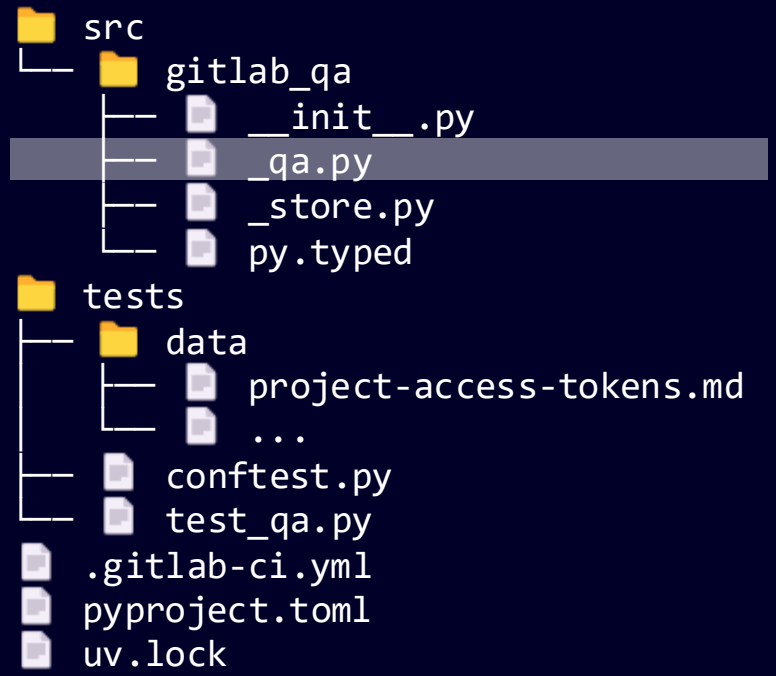
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(
        [
            SystemMessagePromptTemplate.from_template(
                "You are a technical assistant specialized in GitLab. "
                "You answer questions strictly using the provided GitLab "
                "documentation. If the answer is not directly supported by the "
                'provided context, reply EXACTLY with "I don\'t know.", nothing else.'
            ),
            HumanMessagePromptTemplate.from_template(
                ...
            ),
        ]
    )

    ...
```


SIEMENS

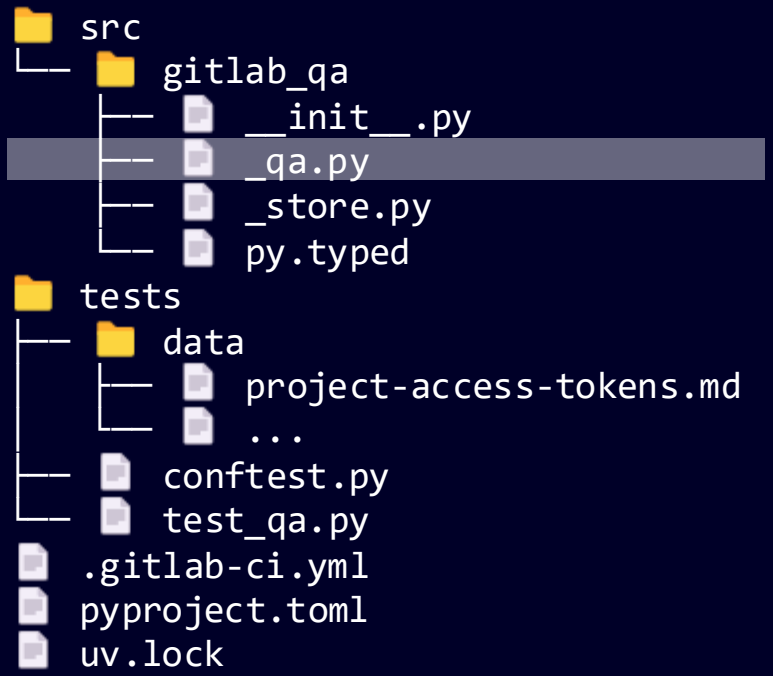
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(
        [
            SystemMessagePromptTemplate.from_template(
                "You are a technical assistant specialized in GitLab. "
                "You answer questions strictly using the provided GitLab "
                "documentation. If the answer is not directly supported by the "
                'provided context, reply EXACTLY with "I don\'t know.", nothing else.'
            ),
            HumanMessagePromptTemplate.from_template(
                ...
            ),
        ]
    )

    ...
```


SIEMENS

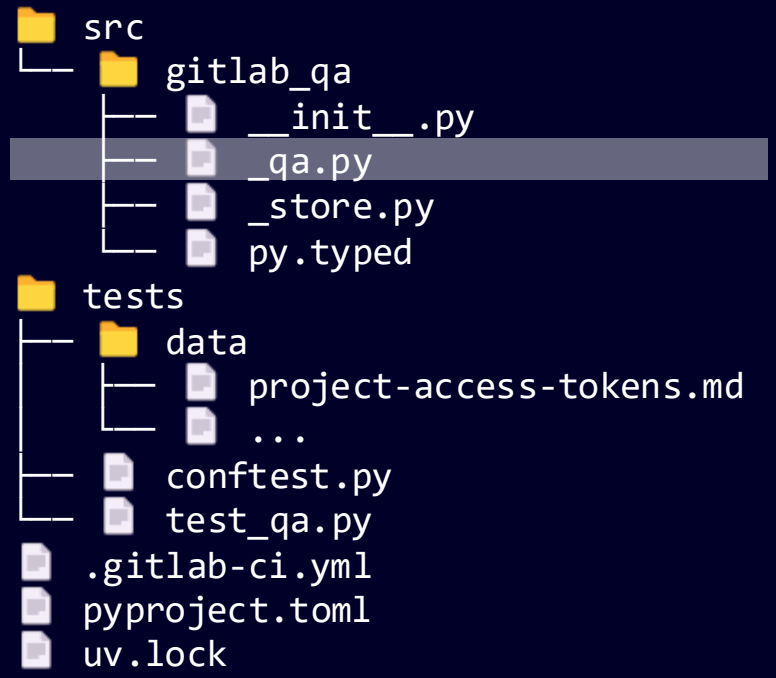
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(
        [
            SystemMessagePromptTemplate.from_template(
                "You are a technical assistant specialized in GitLab. "
                "You answer questions strictly using the provided GitLab "
                "documentation. If the answer is not directly supported by the "
                'provided context, reply EXACTLY with "I don\'t know.", nothing else.'
            ),
            HumanMessagePromptTemplate.from_template(
                "Context from GitLab documentation:\n\n{context}\n\n"
                "Question: {question}\n\n"
                "Provide a technical, accurate, and concise answer based only on the "
                "above context."
            ),
        ]
    )

    ...
```


SIEMENS

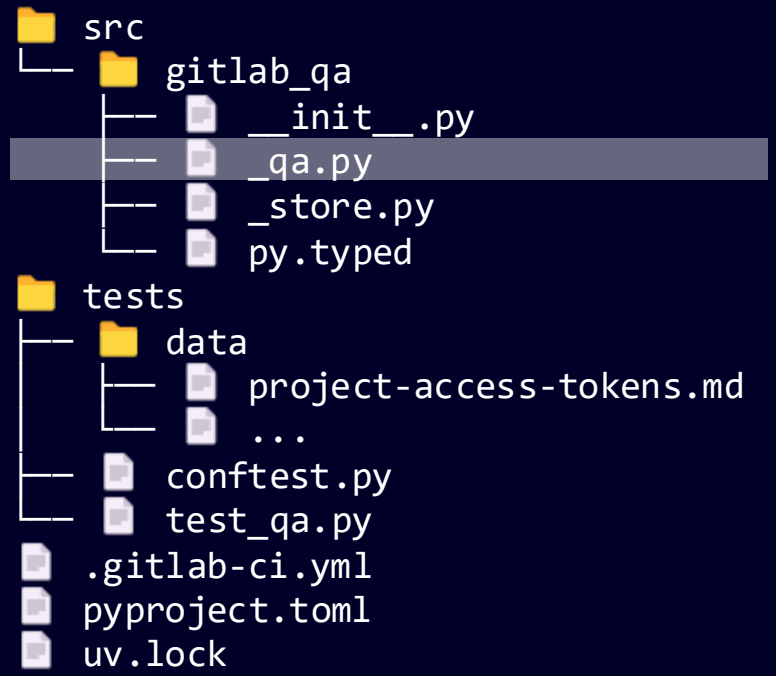
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    ...
```


SIEMENS

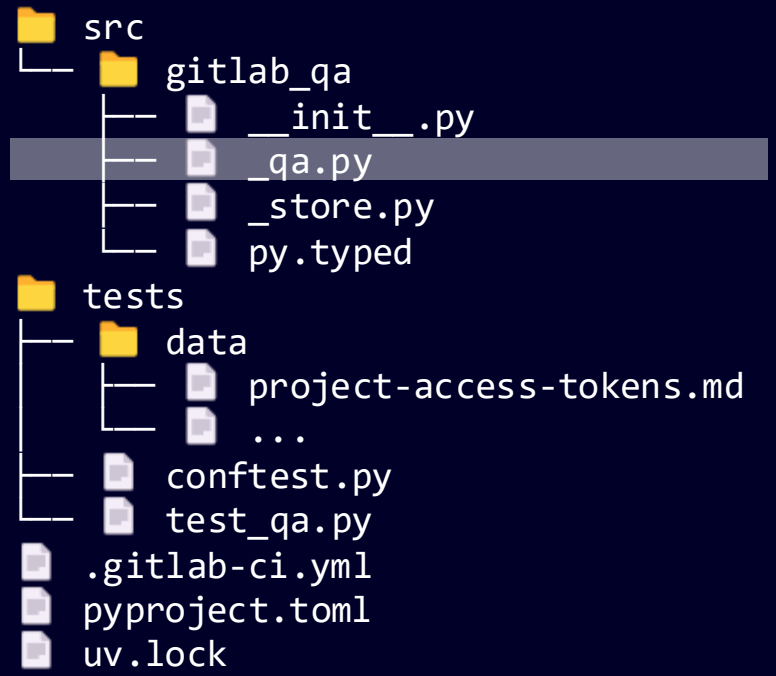
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        ...

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

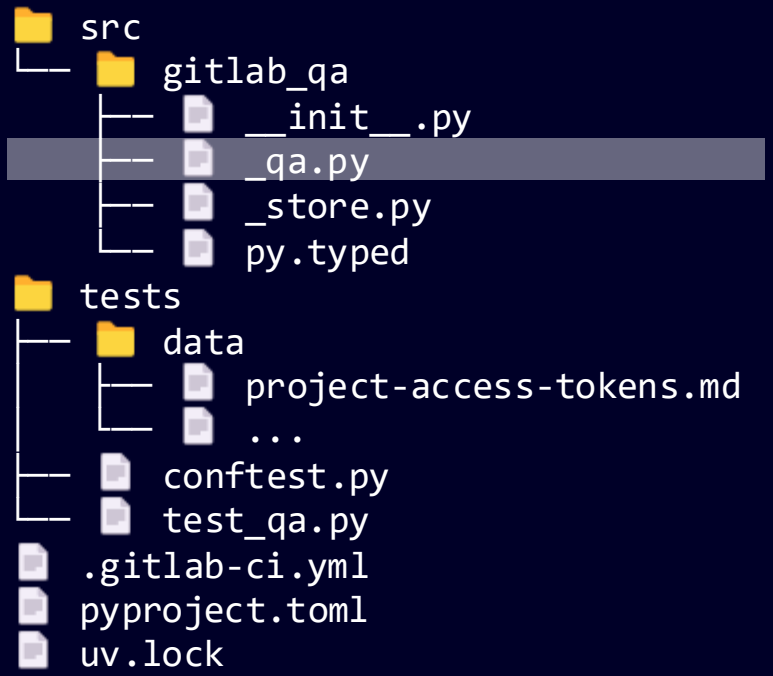
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        ...

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

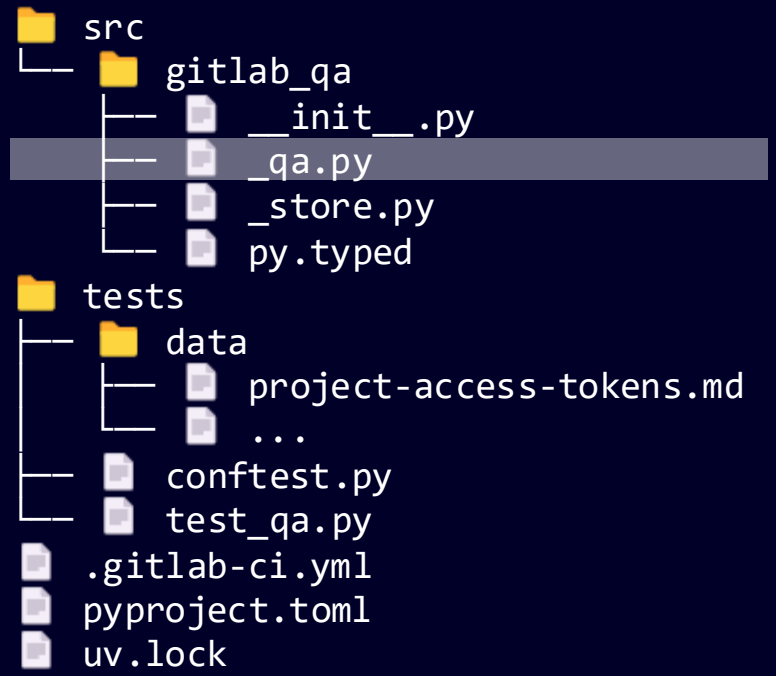
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=...,
                context=...,
            )
        ) | ...

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

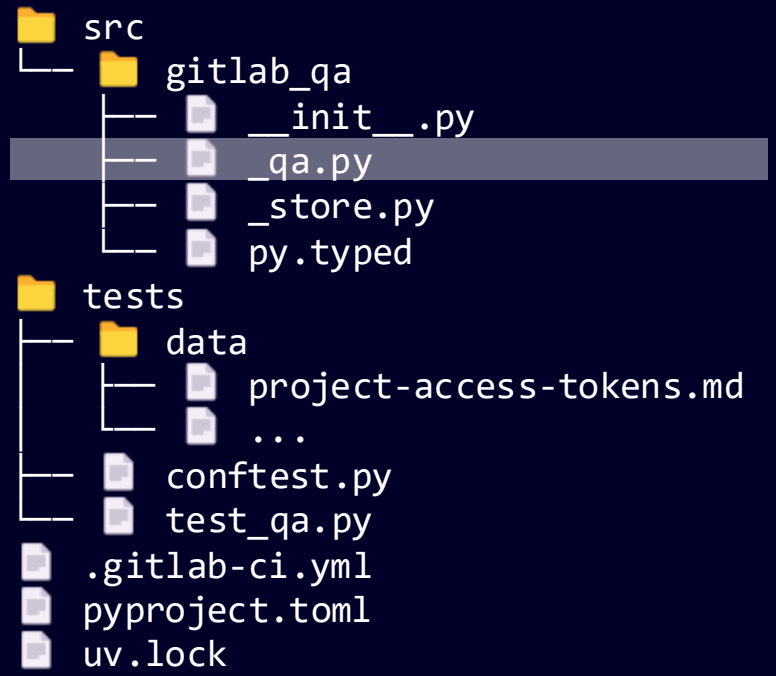
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=...,
                context=...,
            )
        ) | ...

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

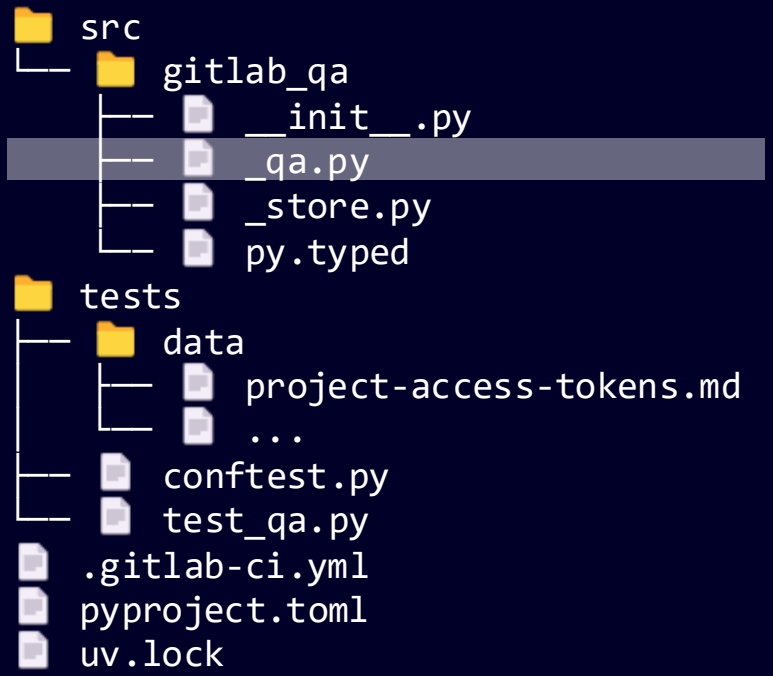
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=...,
            )
        ) | ...

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

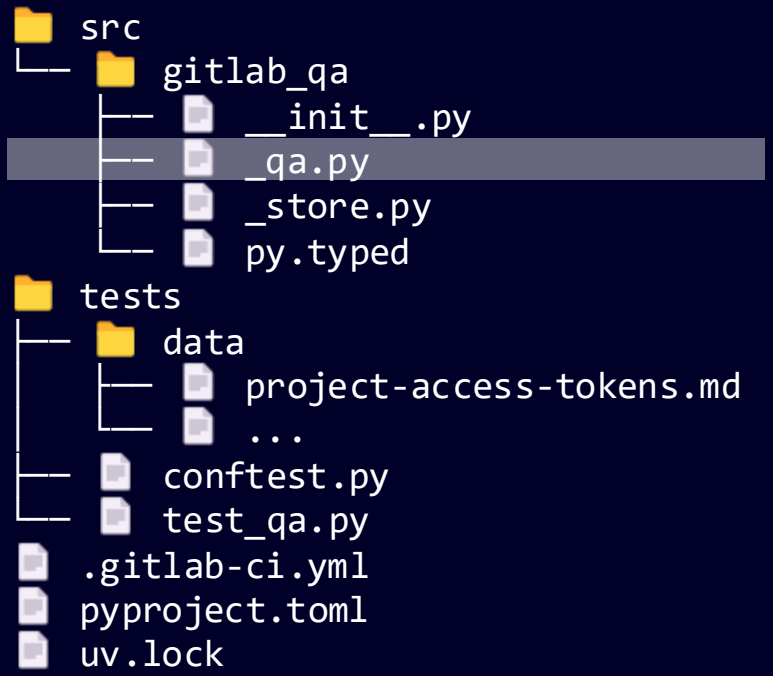
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=...,
            )
        ) | ...

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

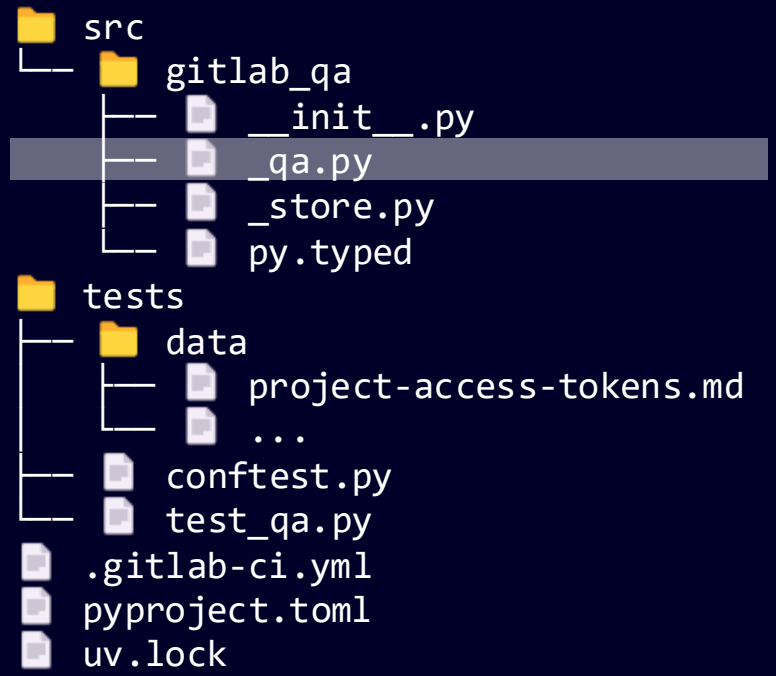
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | ...,
            )
        ) | ...

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

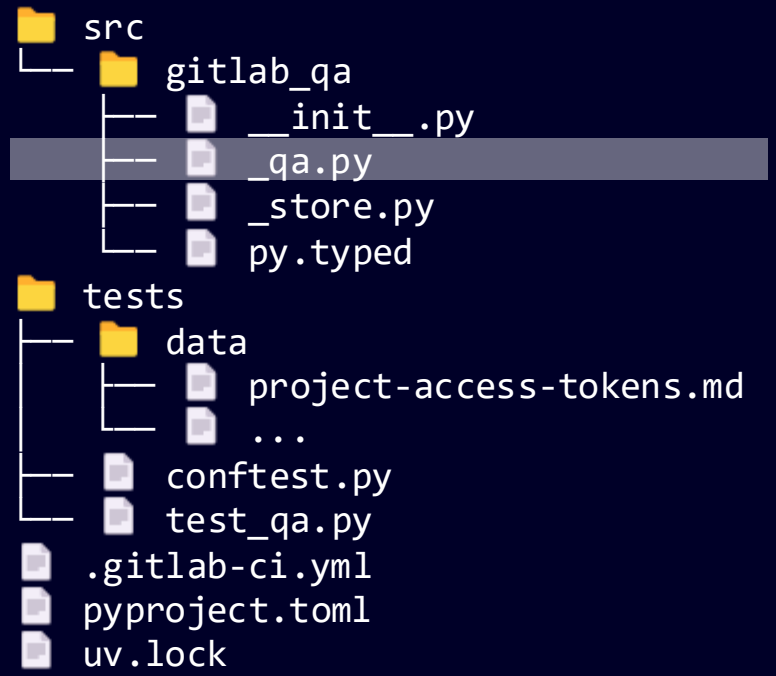
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | ...

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

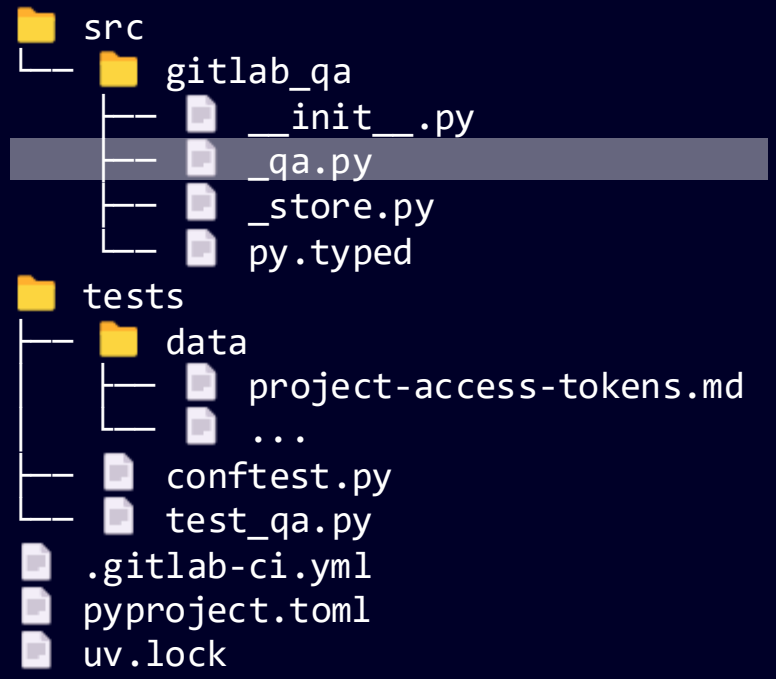
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | ...

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

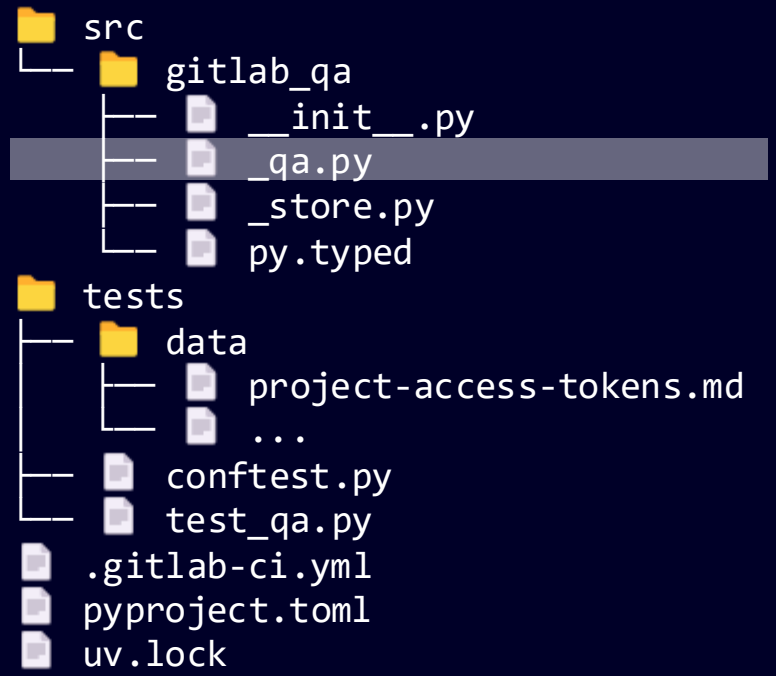
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=...,
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

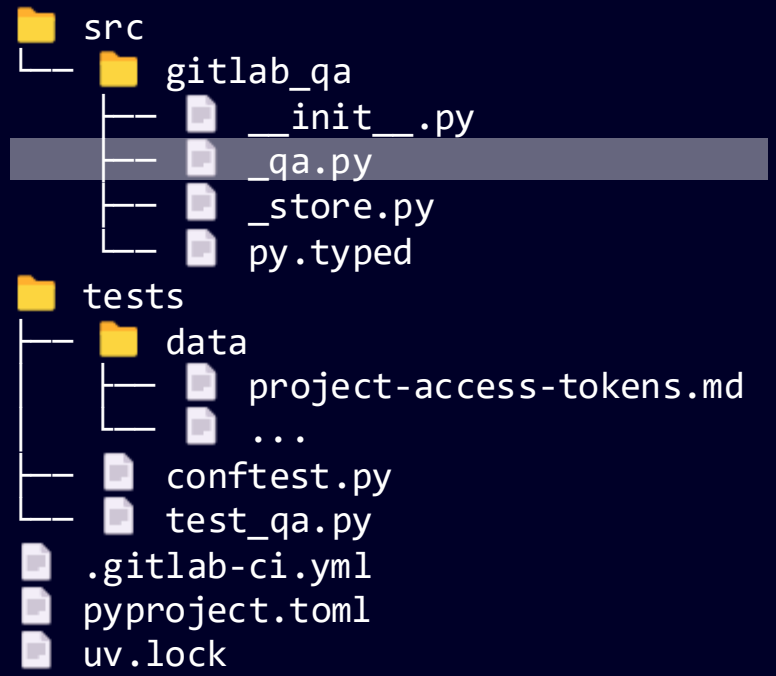
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=...,
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

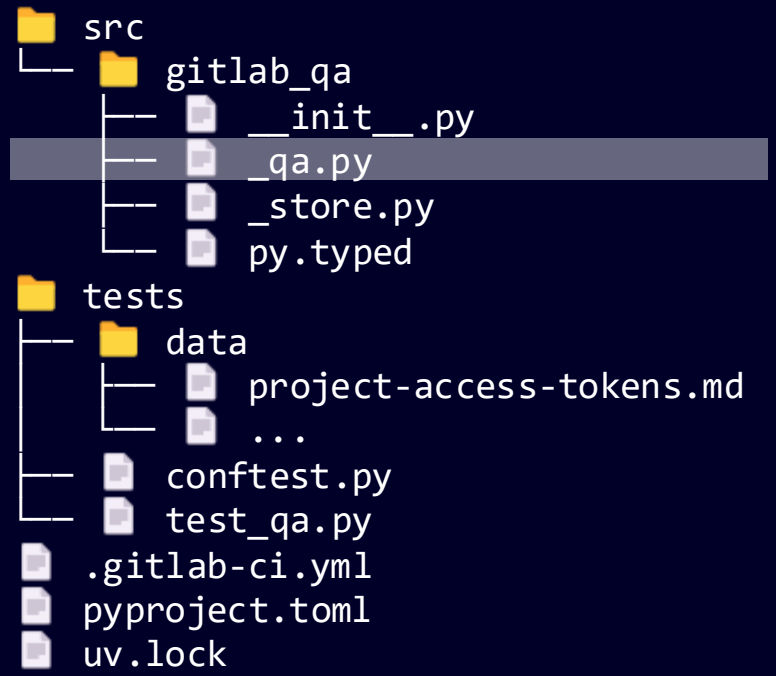
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough() | ...
            ),
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

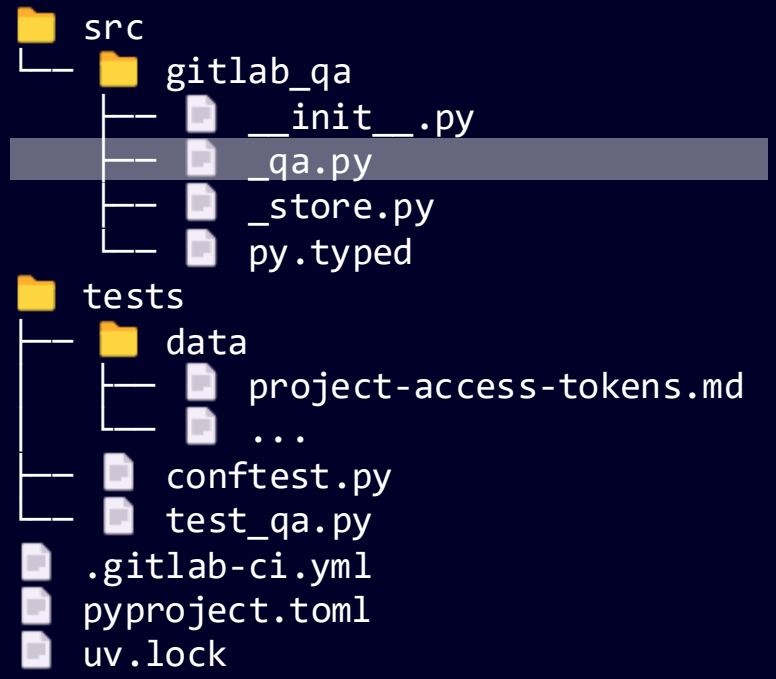
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | ...
            ),
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

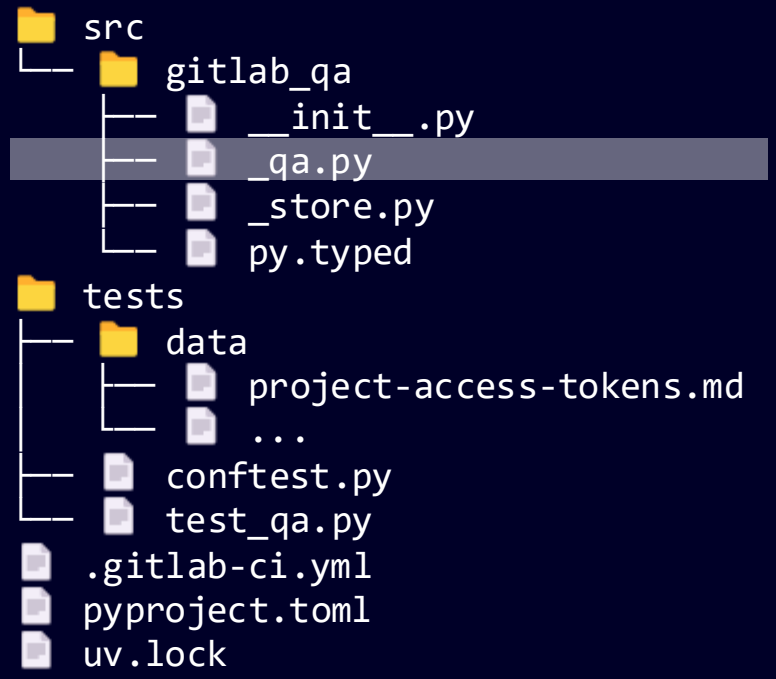
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | ...
            ),
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

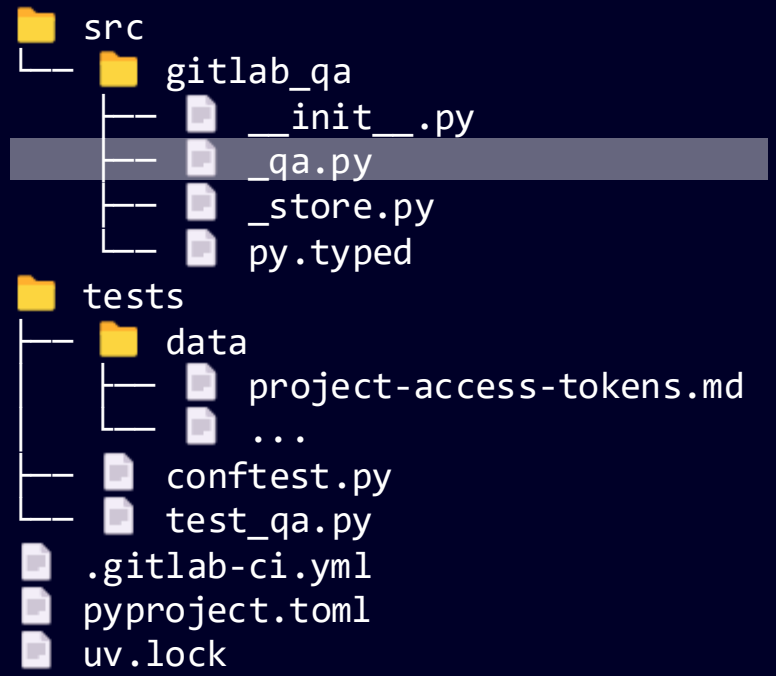
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ...
            ),
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

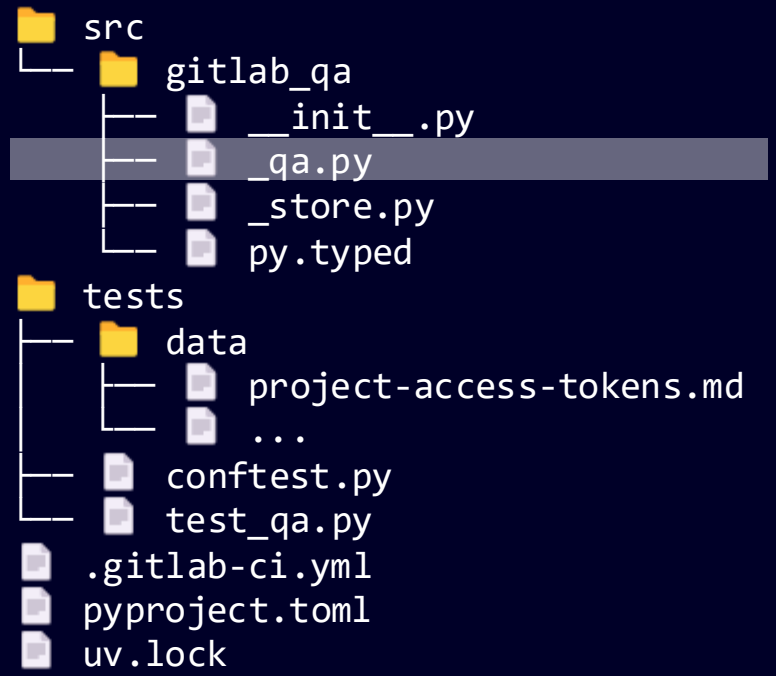
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ...
            ),
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

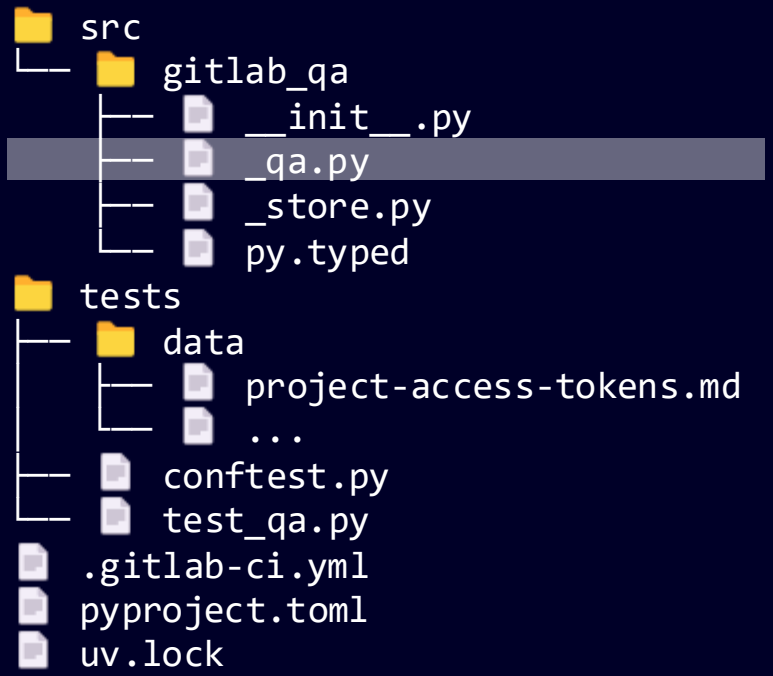
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ChatOllama(model=self.MODEL, base_url=ollama_url, seed=0)
                | ...
            ),
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

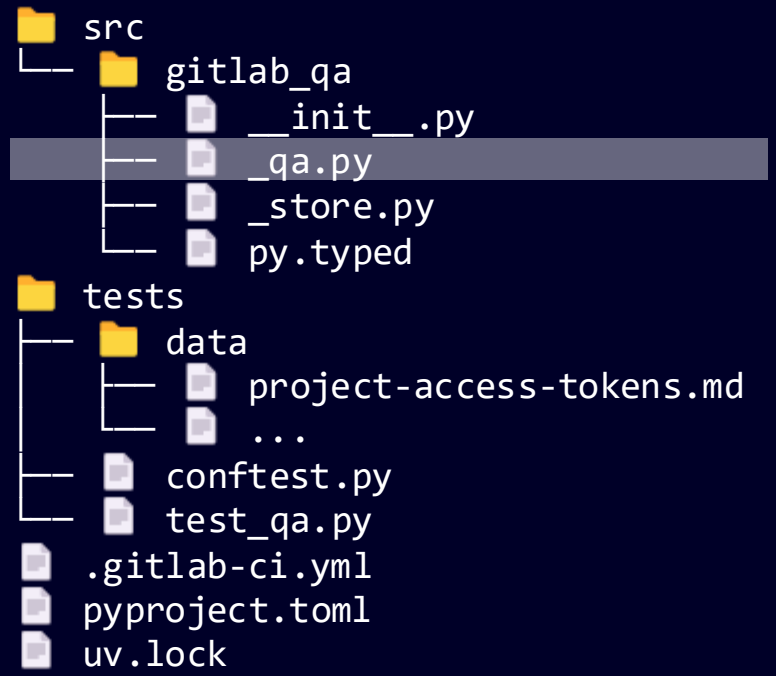
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ChatOllama(model=self.MODEL, base_url=ollama_url, seed=0)
                | ...
            ),
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

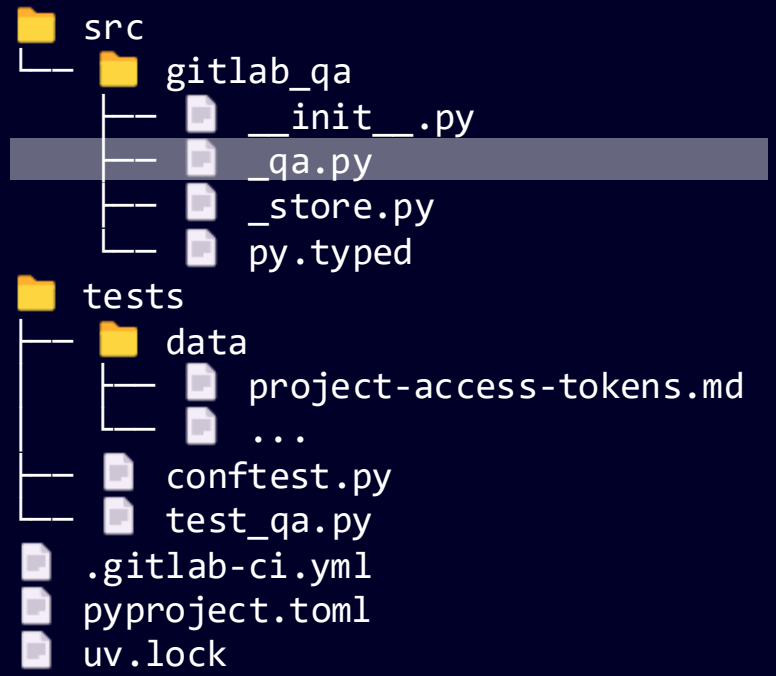
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ChatOllama(model=self.MODEL, base_url=ollama_url, seed=0)
                | StrOutputParser()
            ),
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

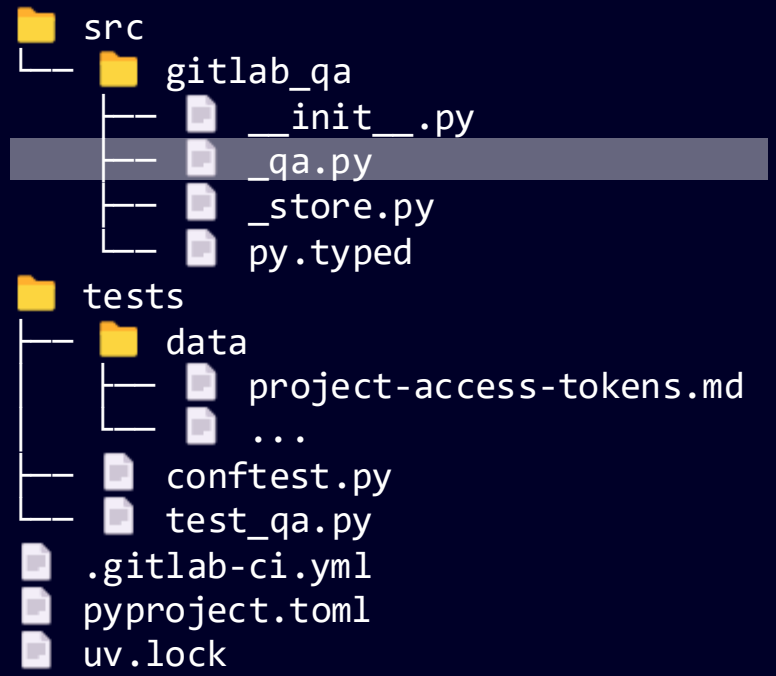
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ChatOllama(model=self.MODEL, base_url=ollama_url, seed=0)
                | StrOutputParser()
            ),
            context=...,
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

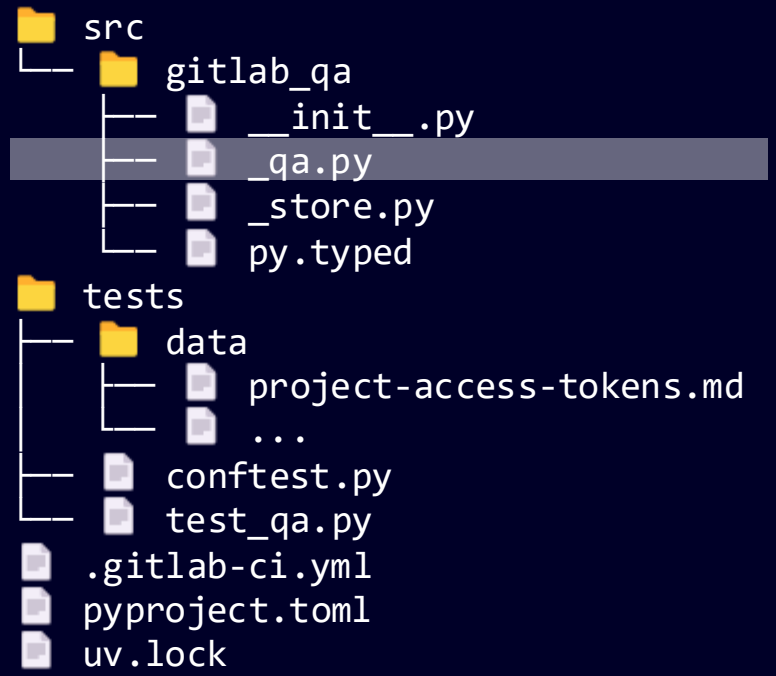
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ChatOllama(model=self.MODEL, base_url=ollama_url, seed=0)
                | StrOutputParser()
            ),
            context=RunnableLambda(itemgetter("context")),
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

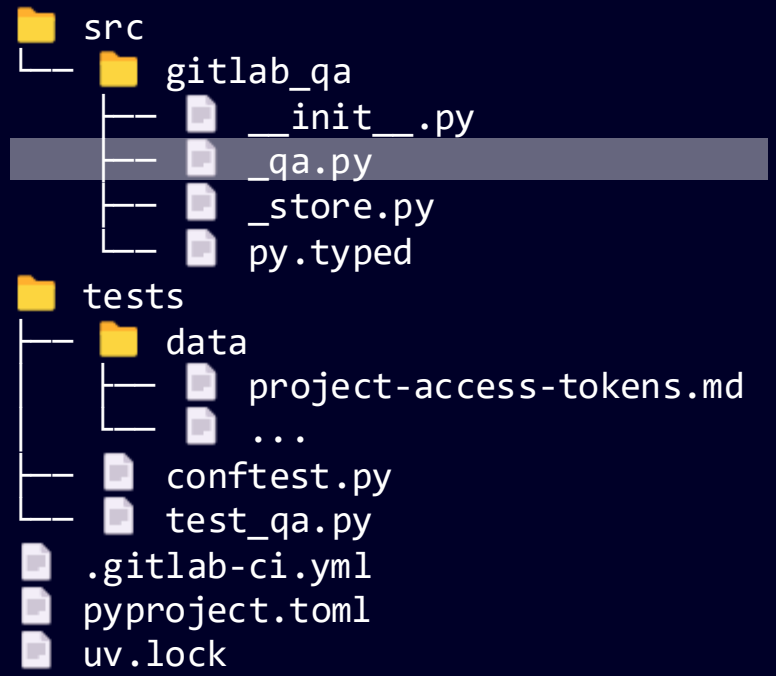
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ChatOllama(model=self.MODEL, base_url=ollama_url, seed=0)
                | StrOutputParser()
            ),
            context=RunnableLambda(itemgetter("context")),
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        ...
```


SIEMENS

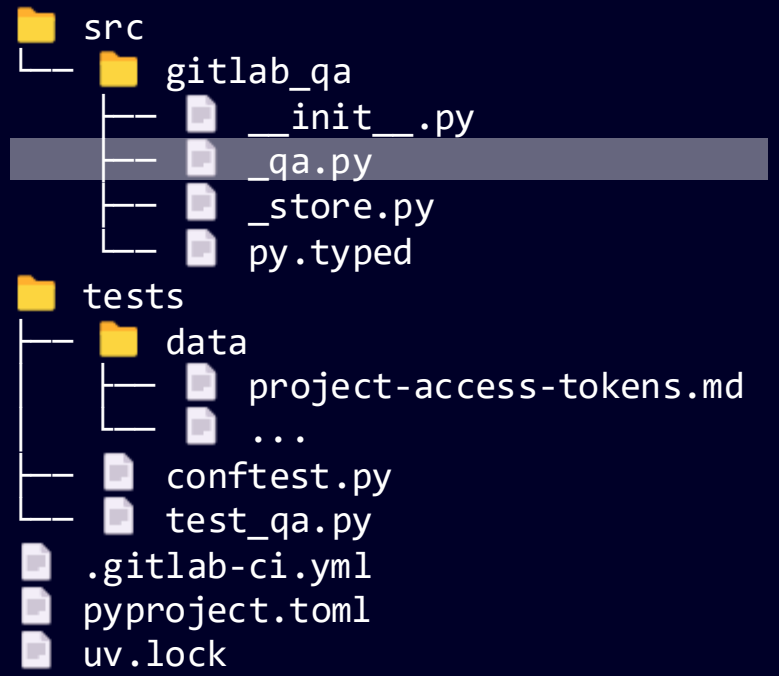
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ChatOllama(model=self.MODEL, base_url=ollama_url, seed=0)
                | StrOutputParser()
            ),
            context=RunnableLambda(itemgetter("context")),
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        response = self._qa_chain.invoke(question)
        ...
```


SIEMENS

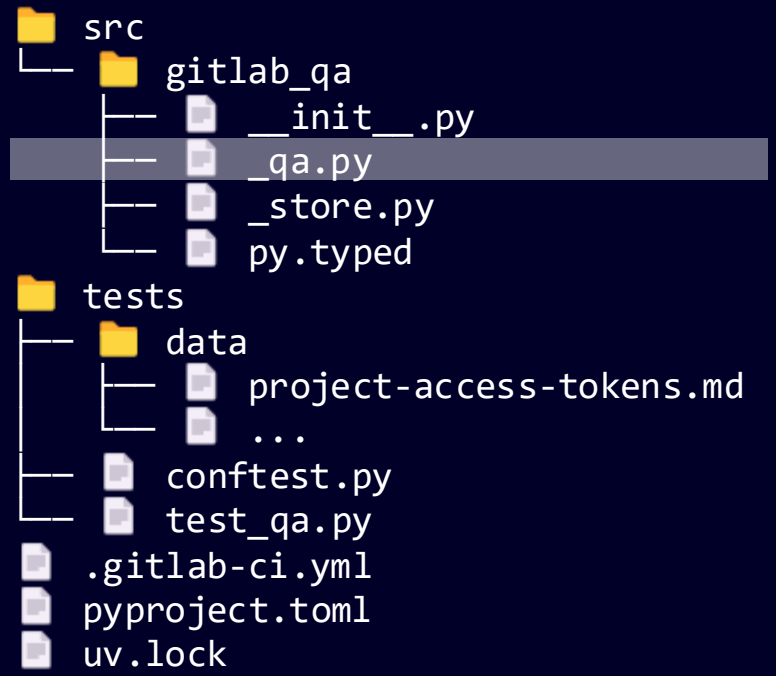
GitLab Q&A demo project




```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ChatOllama(model=self.MODEL, base_url=ollama_url, seed=0)
                | StrOutputParser()
            ),
            context=RunnableLambda(itemgetter("context")),
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        response = self._qa_chain.invoke(question)
        ...
```


SIEMENS

GitLab Q&A demo project

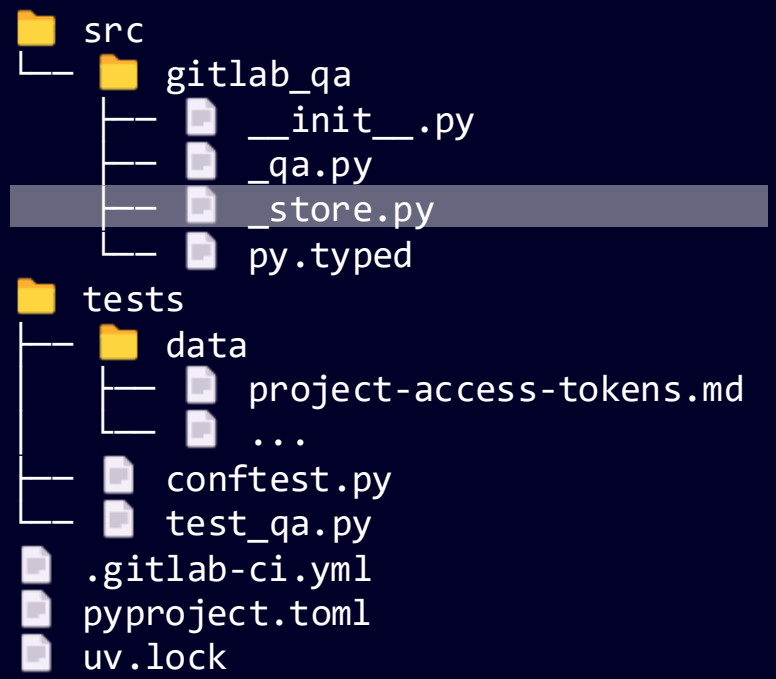



```python
class GitlabQA:
    MODEL: Final[str] = "phi4-mini"

    PROMPT: Final = ChatPromptTemplate.from_messages(...)

    def __init__(
        self,
        retriever: VectorStoreRetriever,
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._qa_chain = (
            RunnableParallel(
                question=RunnablePassthrough(),
                context=retriever | (lambda docs: [d.page_content for d in docs]),
            )
        ) | RunnableParallel(
            answer=(
                RunnablePassthrough().assign(
                    context=lambda x: "\n\n".join(x["context"])
                )
                | self.PROMPT
                | ChatOllama(model=self.MODEL, base_url=ollama_url, seed=0)
                | StrOutputParser()
            ),
            context=RunnableLambda(itemgetter("context")),
        )

    def ask(self, question: str) -> tuple[str, Sequence[str]]:
        response = self._qa_chain.invoke(question)
        return response["answer"], response["context"]
```


SIEMENS

GitLab Q&A demo project

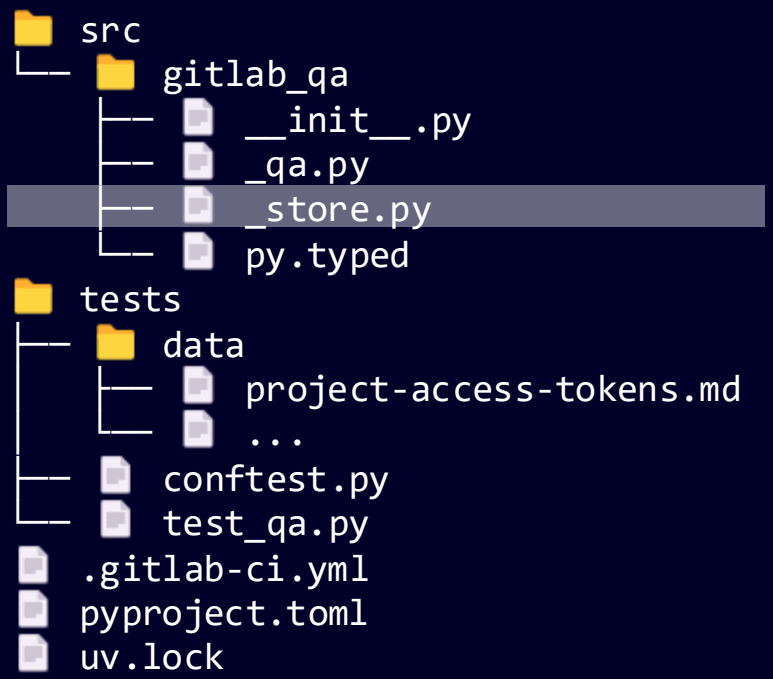
```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = ...

    ...
```



SIEMENS

GitLab Q&A demo project

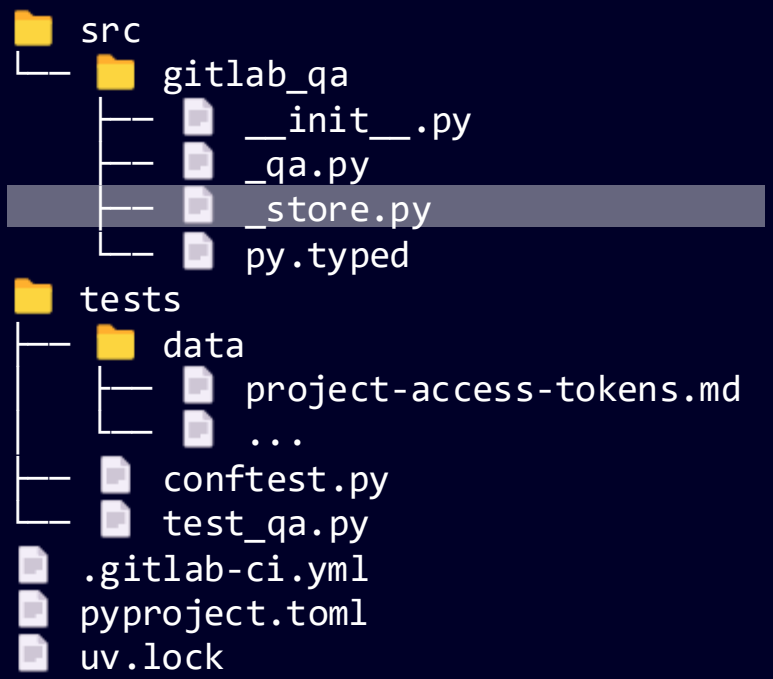
```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = ...

    ...
```



SIEMENS

GitLab Q&A demo project

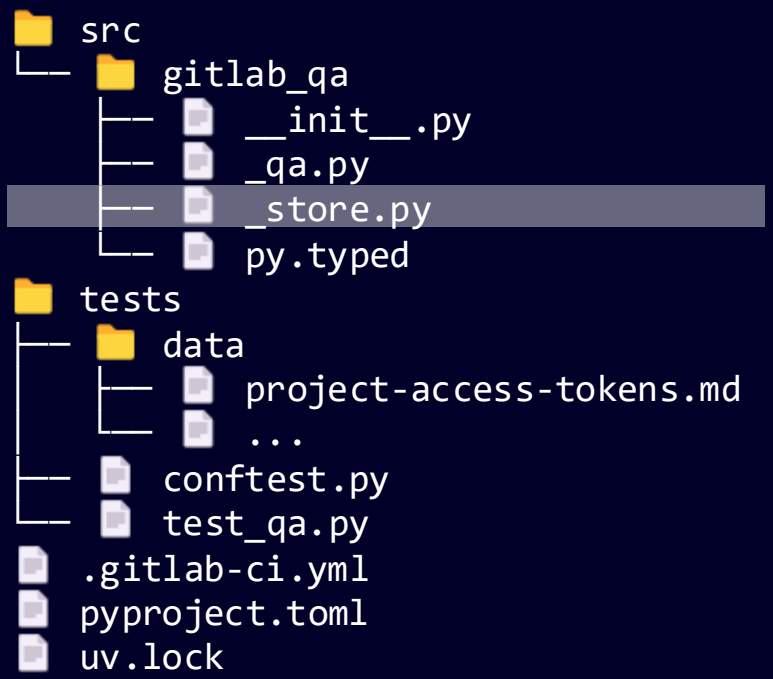
```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    ...
```



SIEMENS

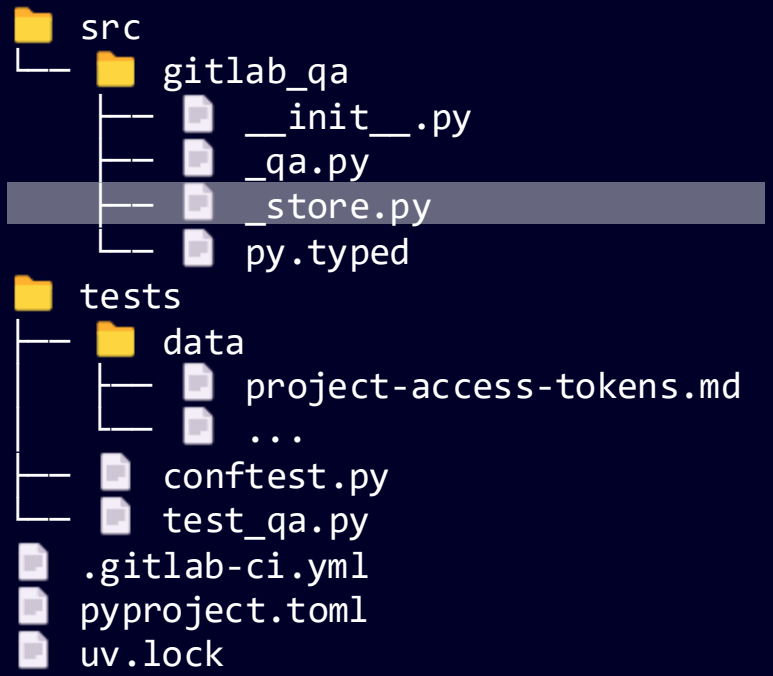
GitLab Q&A demo project

```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    ...
```



SIEMENS

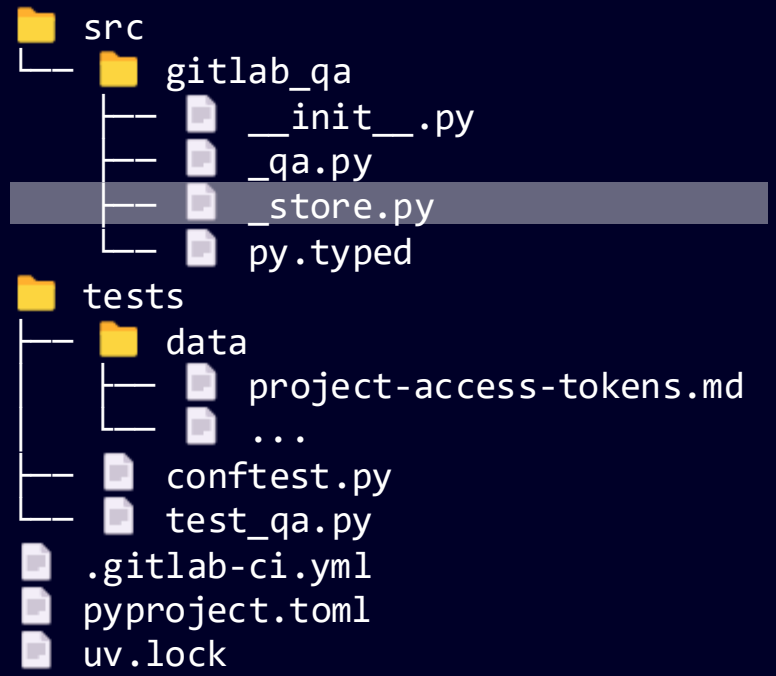
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

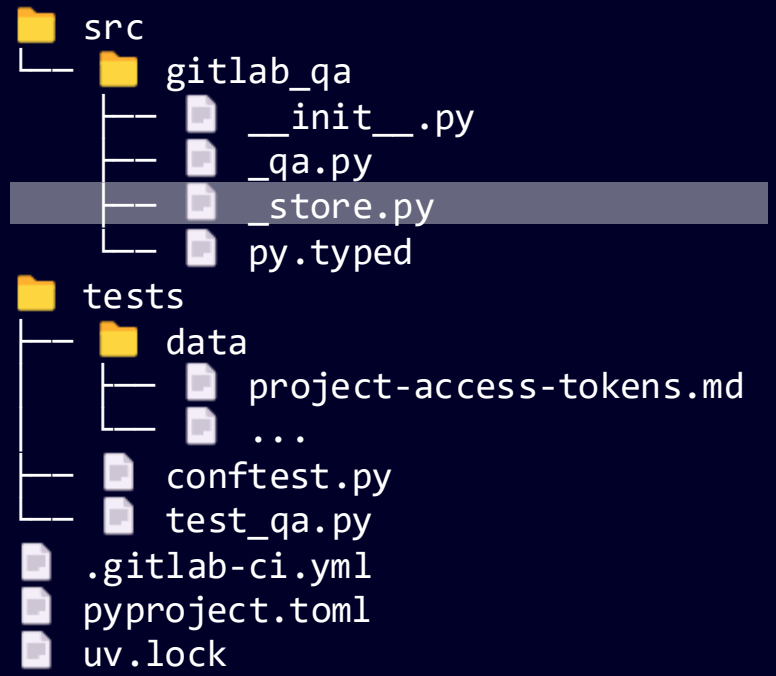
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = ...
        self._record_manager = ...
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

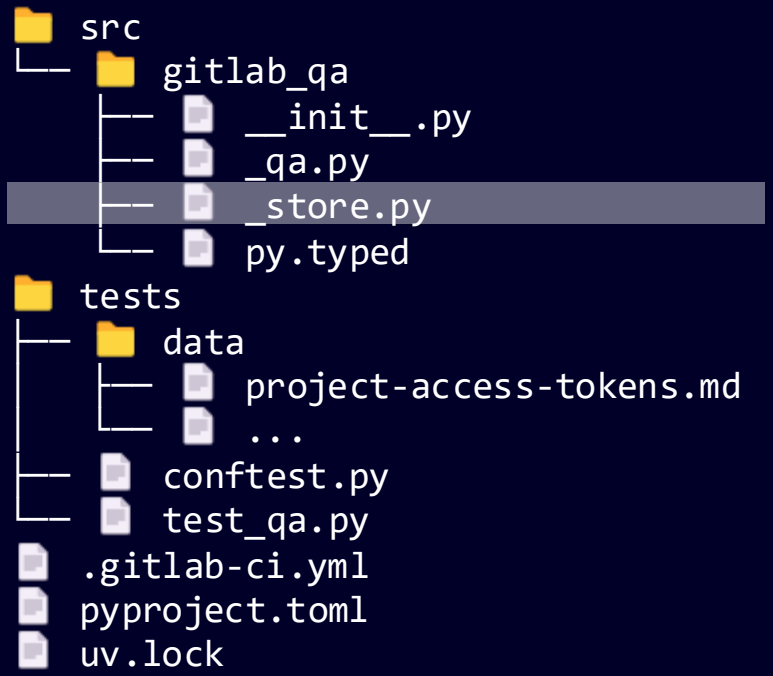
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = ...
        self._record_manager = ...
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

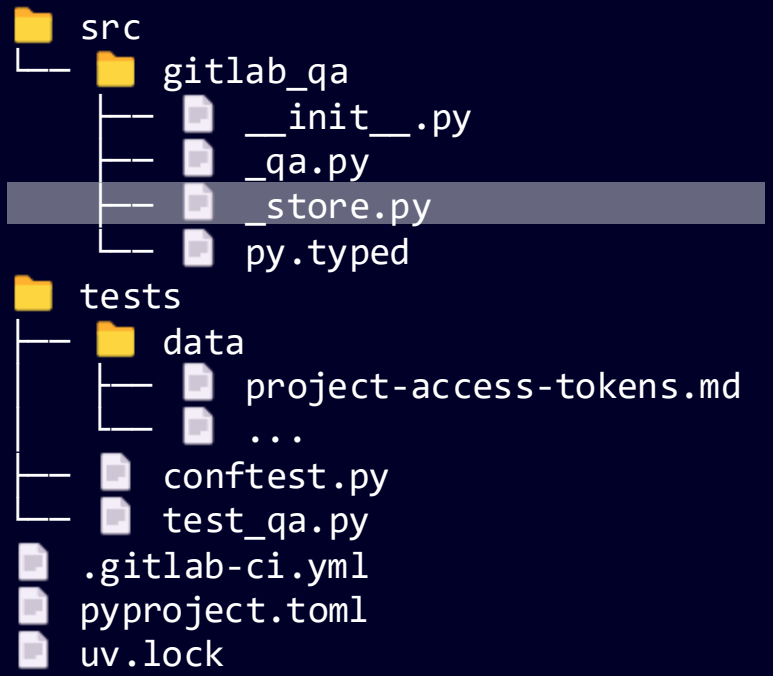
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            ...,
            connection=...,
            collection_name=...,
        )
        self._record_manager = ...
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

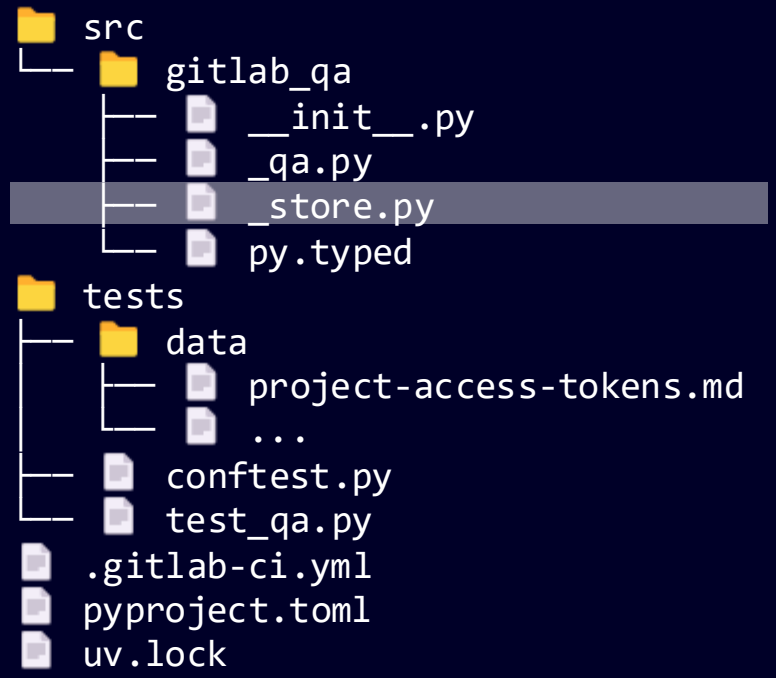
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            ...,
            connection=...,
            collection_name=...,
        )
        self._record_manager = ...
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

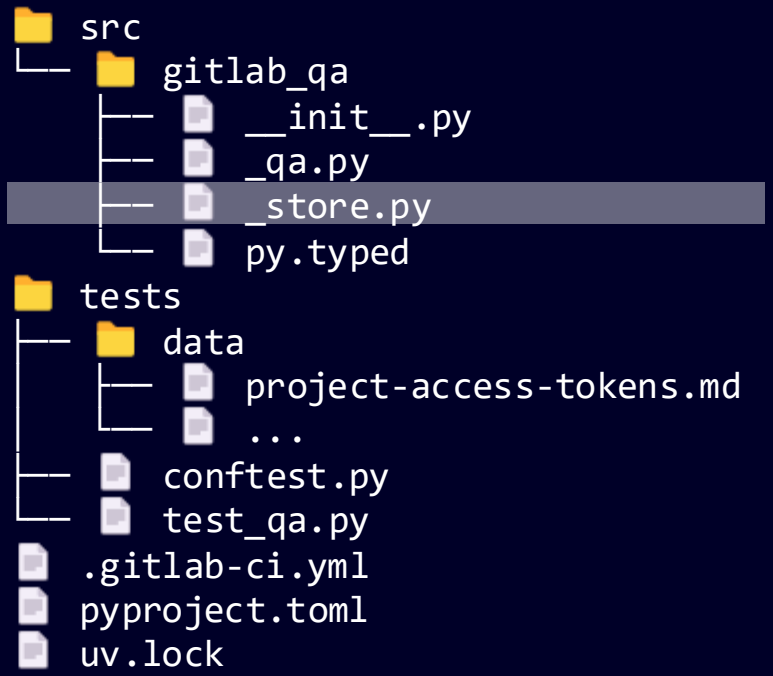
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=...,
            collection_name=...,
        )
        self._record_manager = ...
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

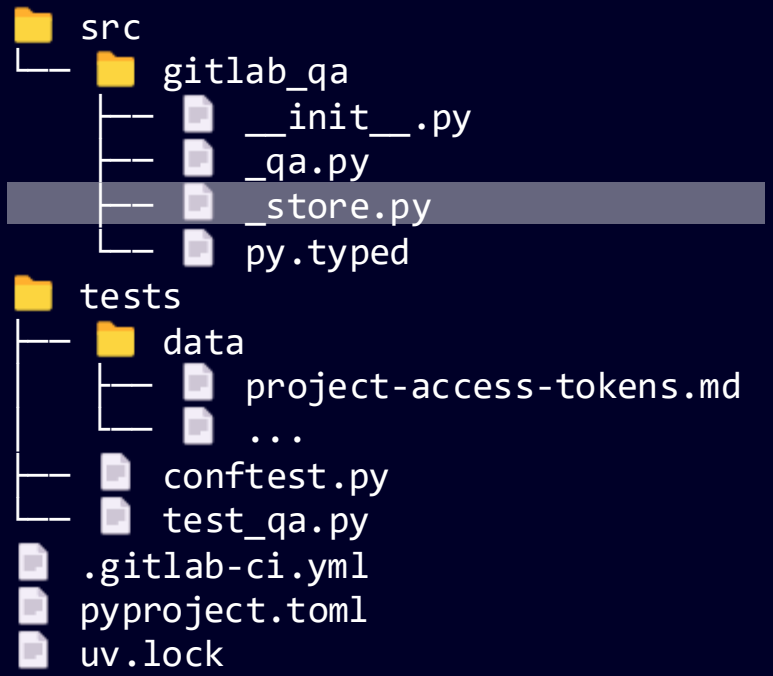
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=...,
            collection_name=...,
        )
        self._record_manager = ...
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

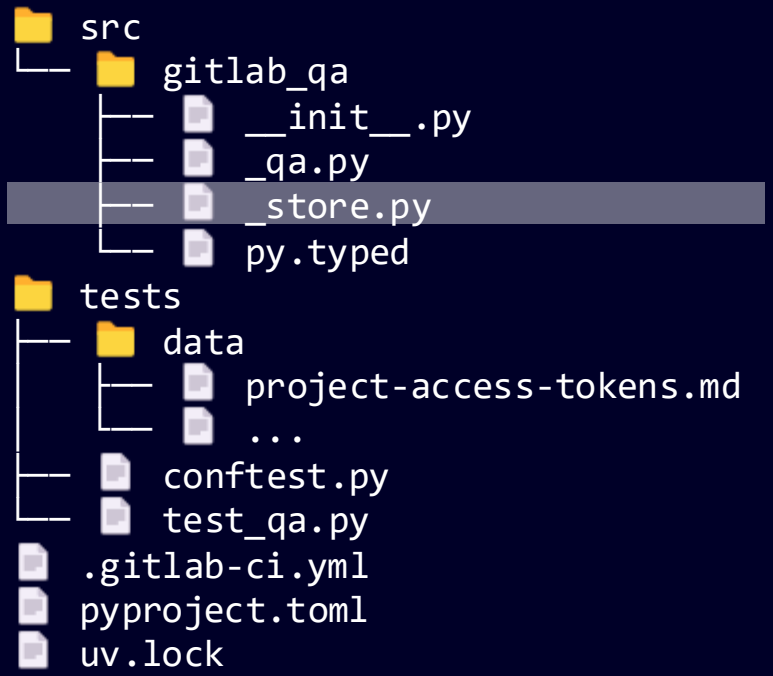
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=...,
        )
        self._record_manager = ...
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

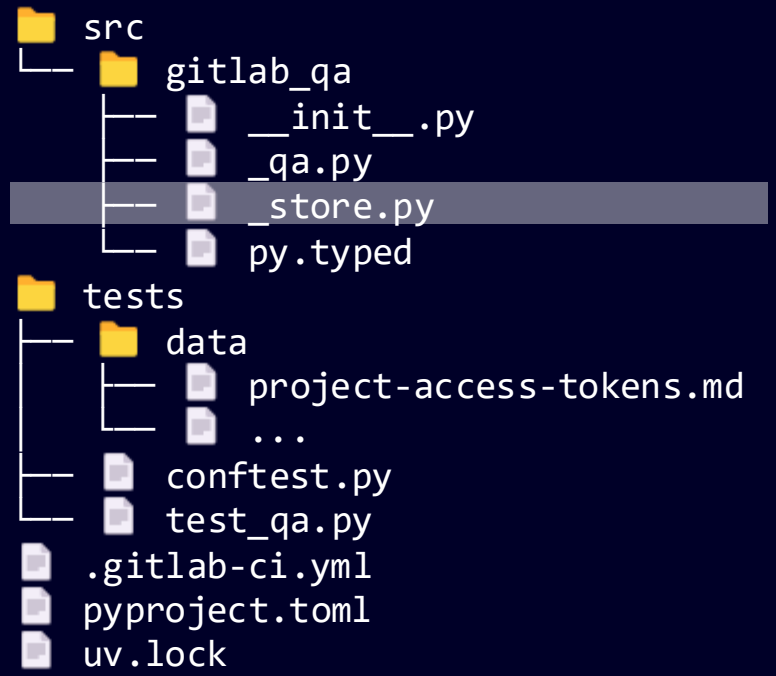
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=...,
        )
        self._record_manager = ...
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

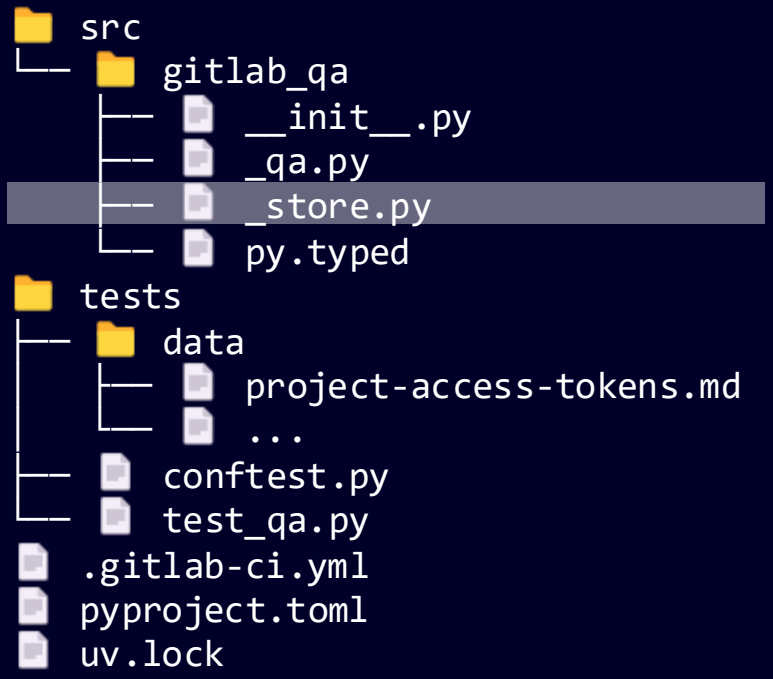
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = ...
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

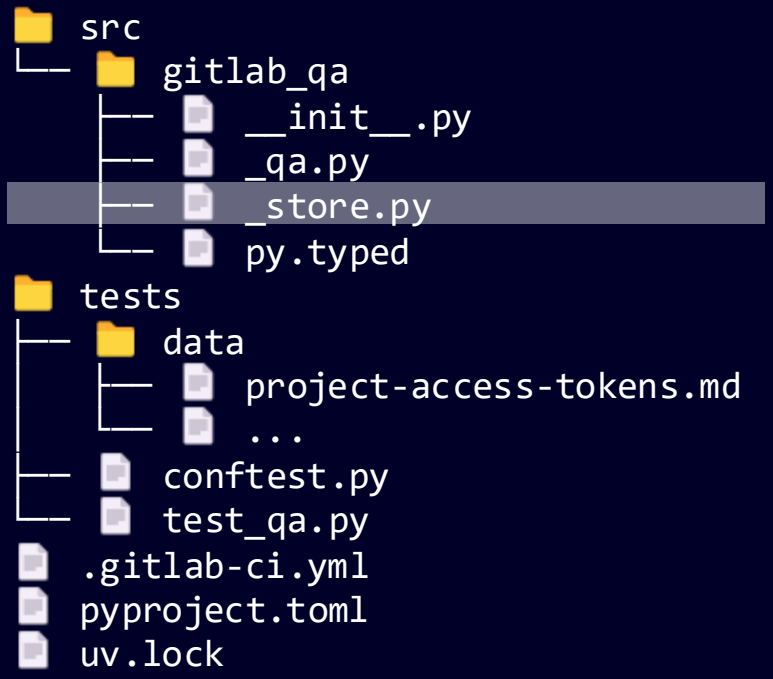
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

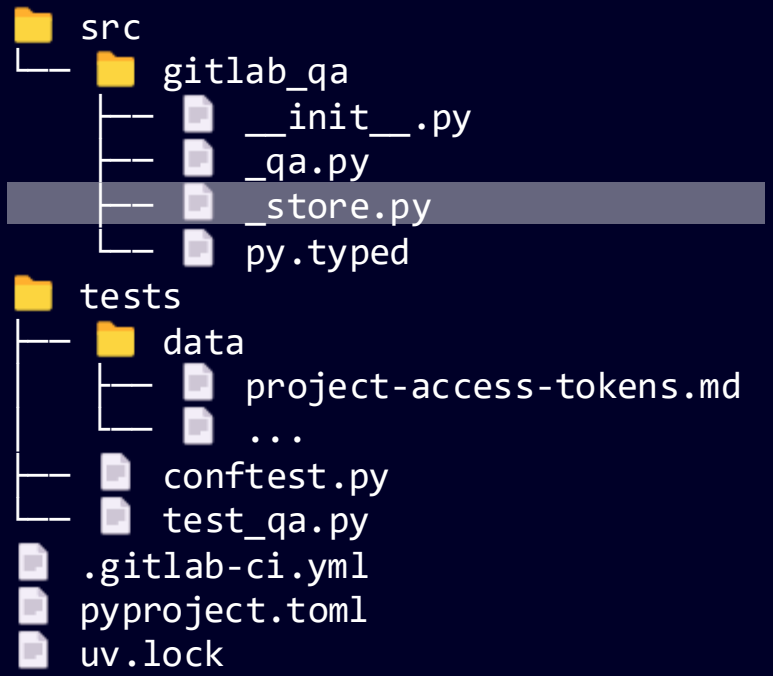
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = ...

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

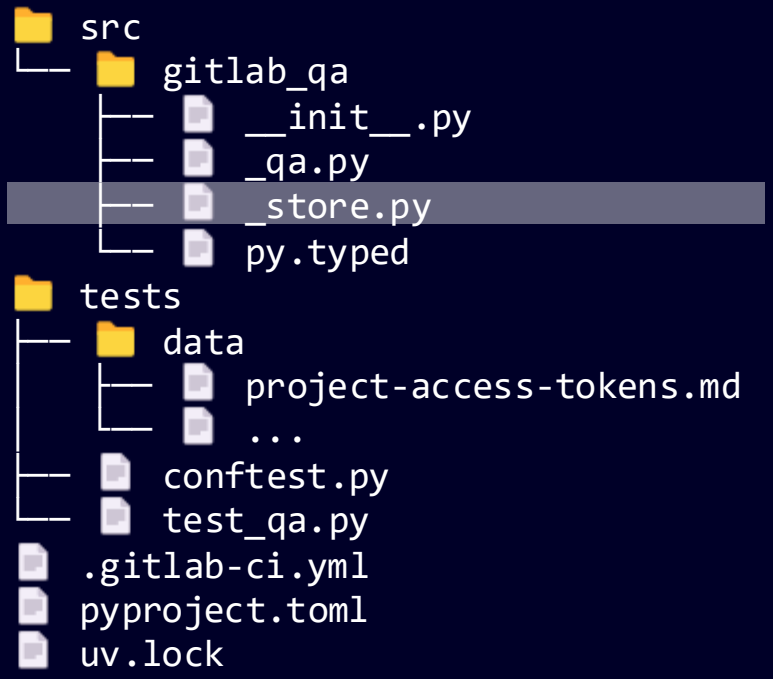
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

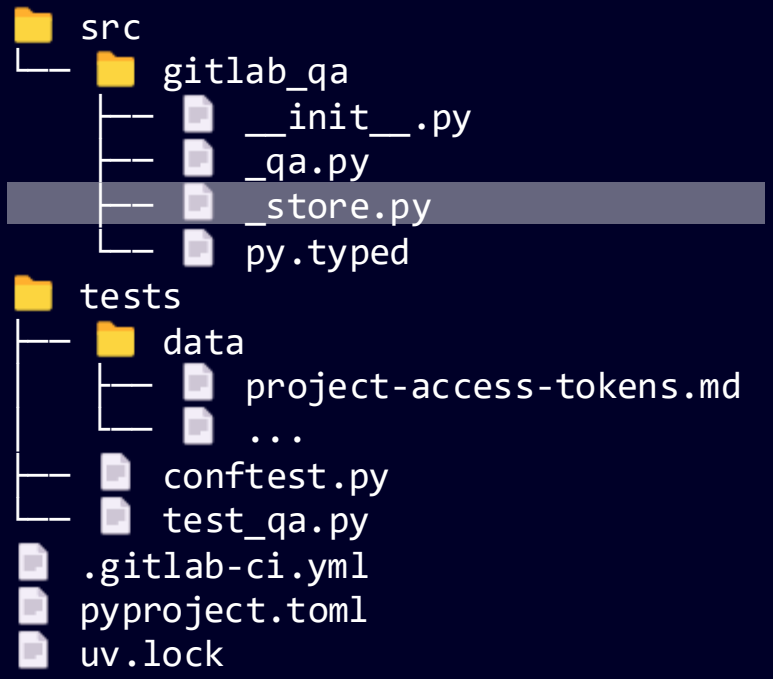
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        ...

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

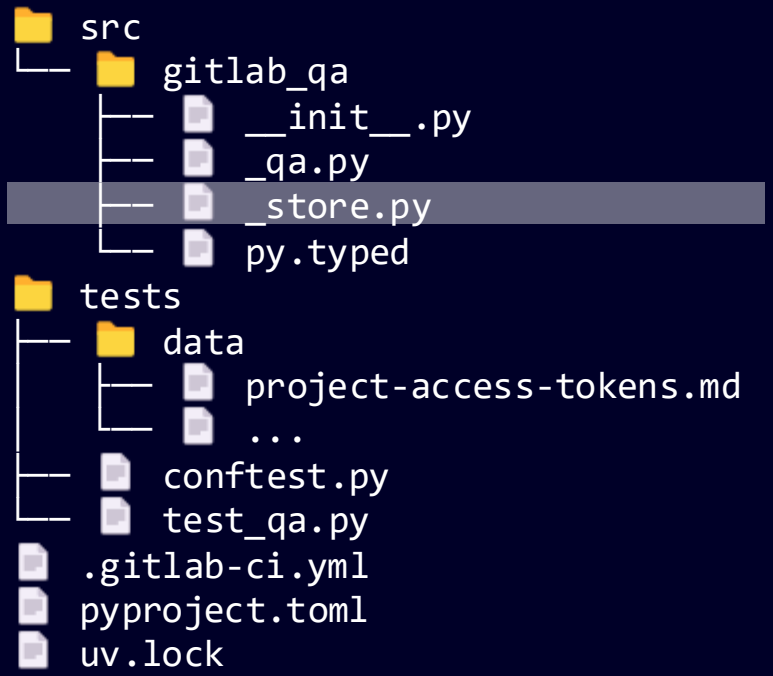
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

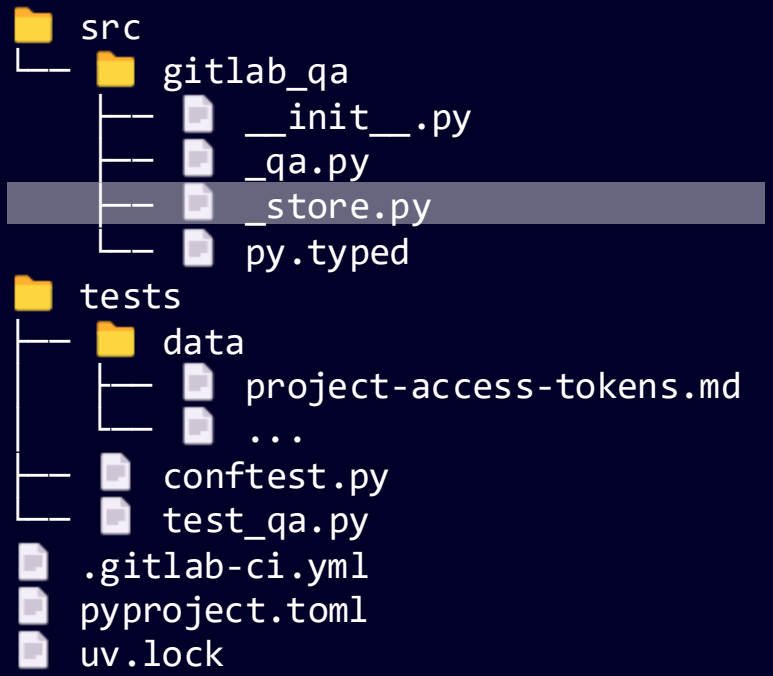
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

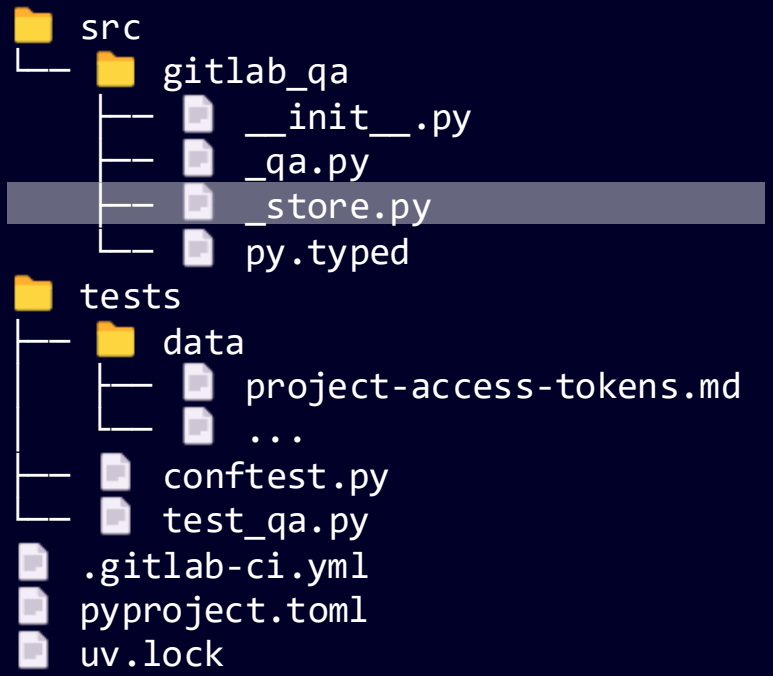
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

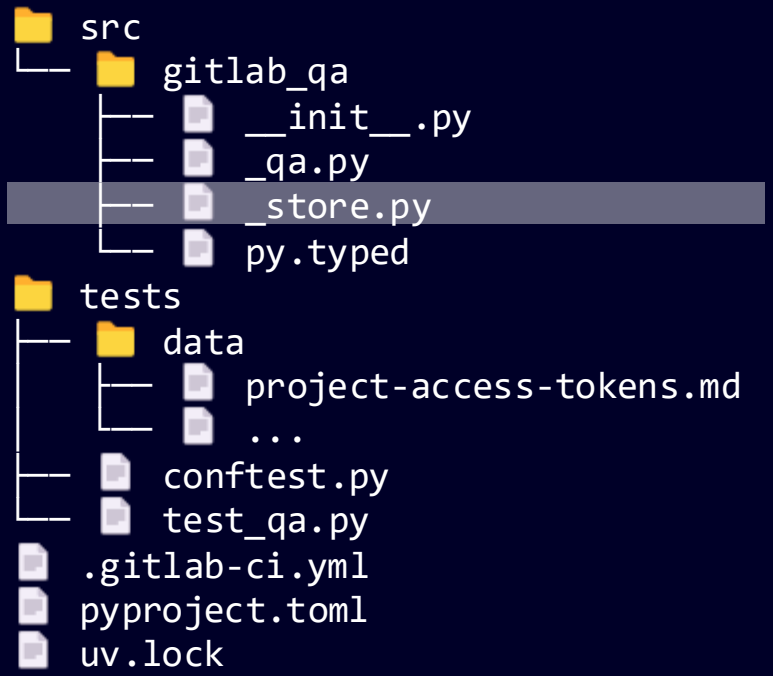
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        ...

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

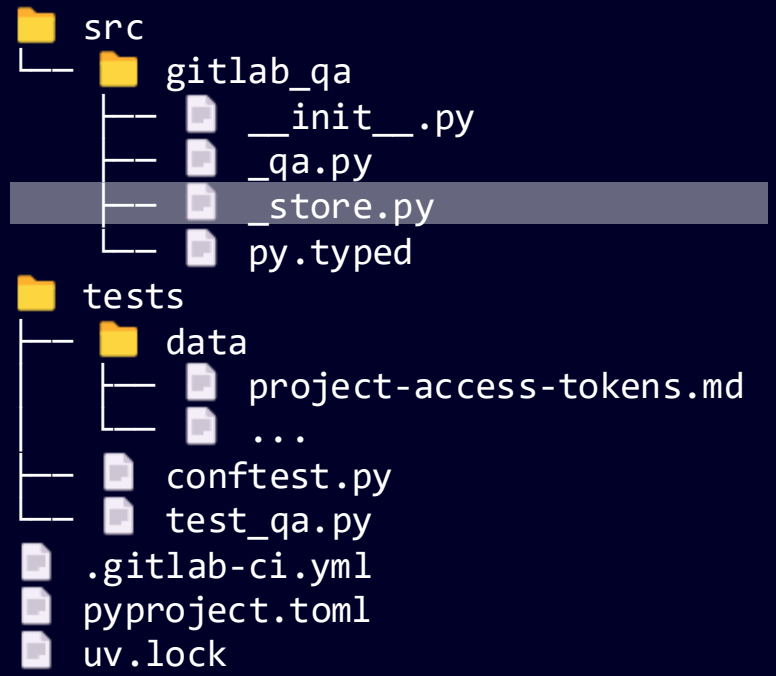
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            ...,
            ...,
            ...,
            cleanup=...,
            source_id_key=...,
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

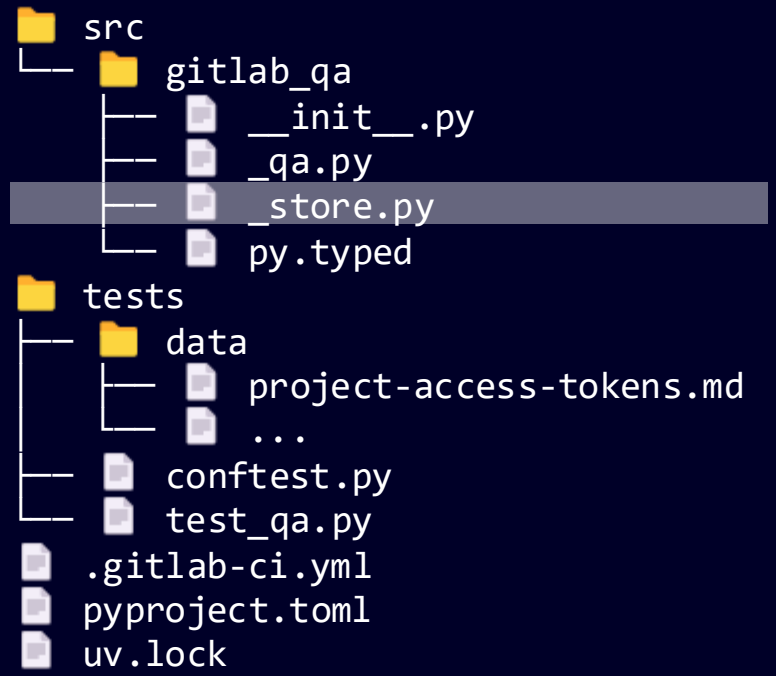
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            ...,
            ...,
            ...,
            cleanup=...,
            source_id_key=...,
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

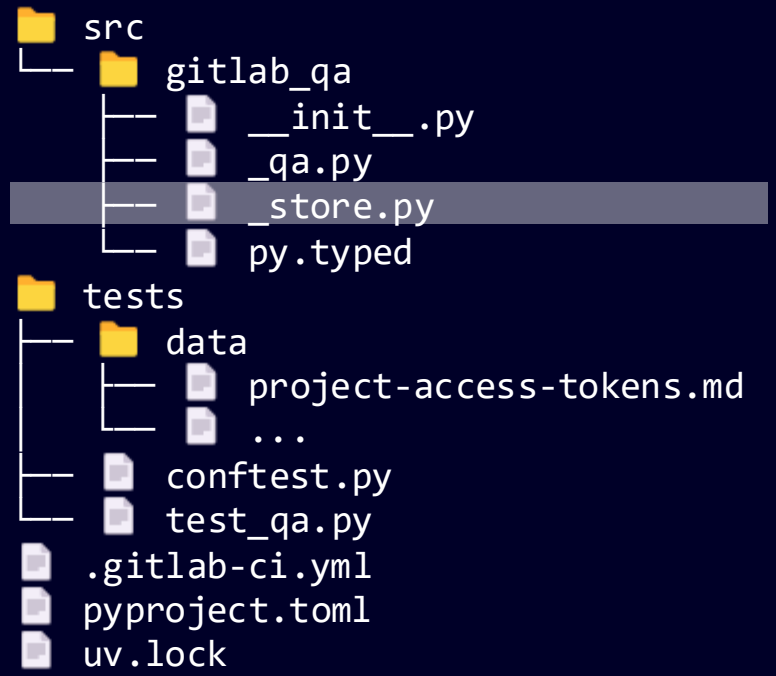
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            ...,
            ...,
            cleanup=...,
            source_id_key=...,
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

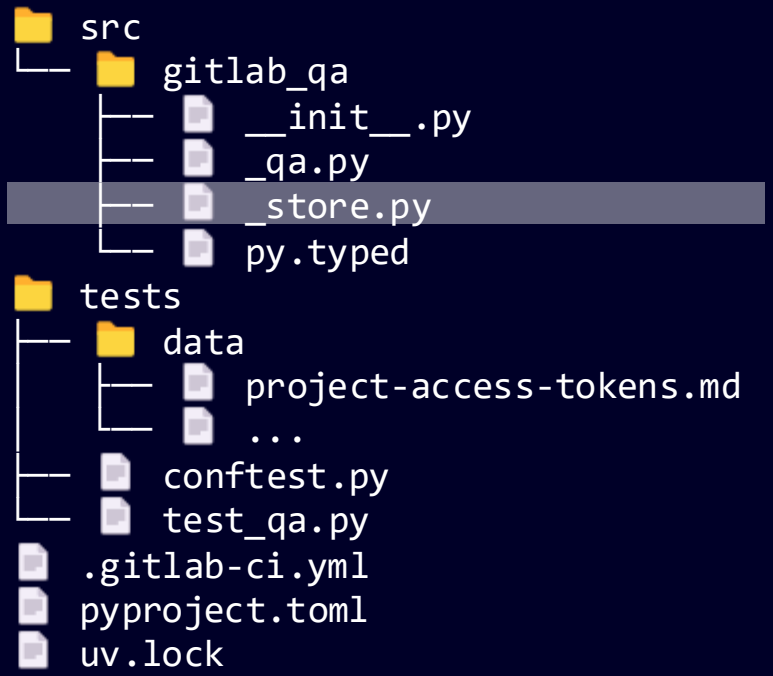
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            ...,
            ...,
            cleanup=...,
            source_id_key=...,
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

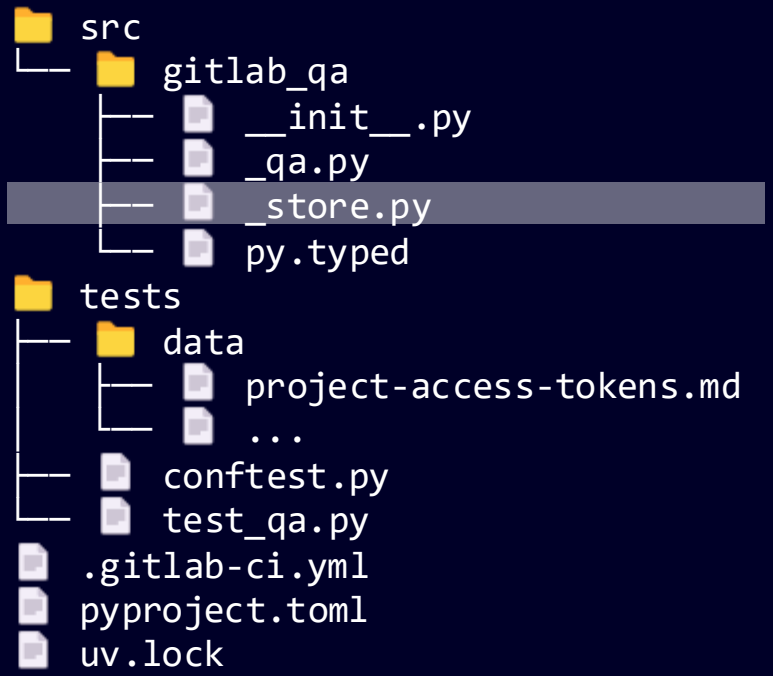
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            self._record_manager,
            ...,
            cleanup=...,
            source_id_key=...,
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

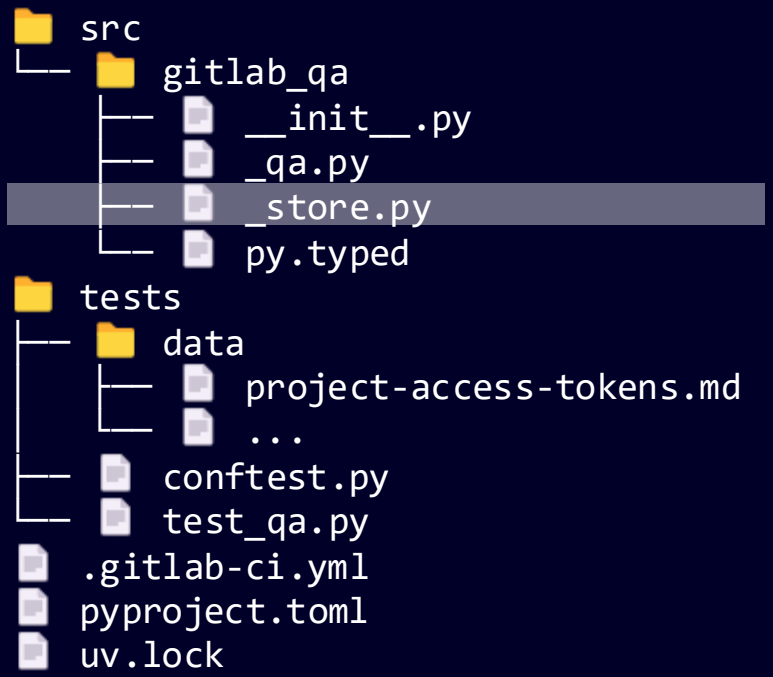
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            self._record_manager,
            ...,
            cleanup=...,
            source_id_key=...,
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

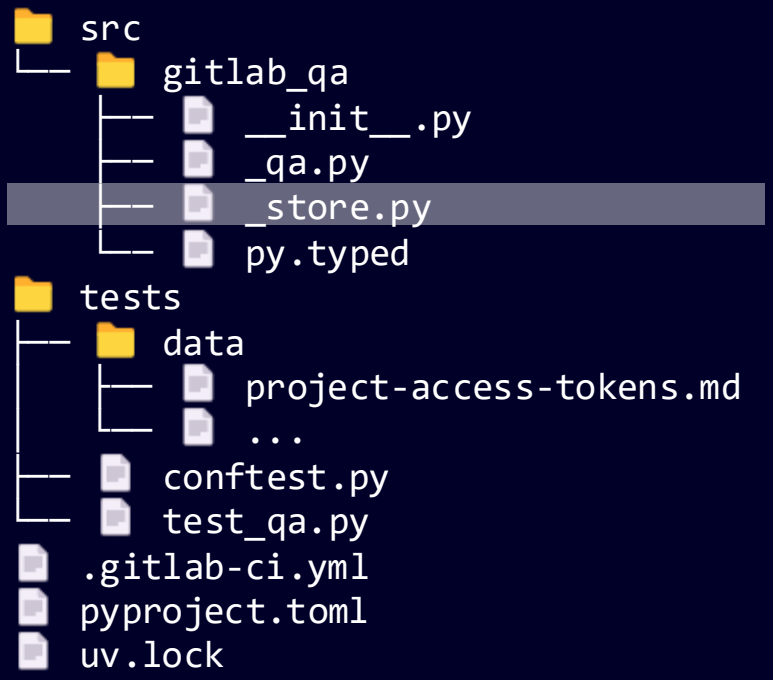
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            self._record_manager,
            self._store,
            cleanup=...,
            source_id_key=...,
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

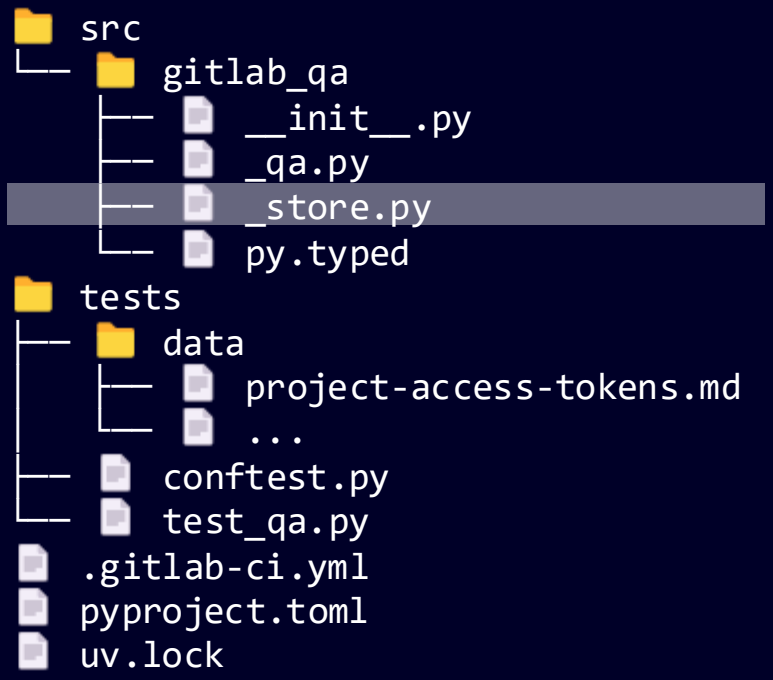
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            self._record_manager,
            self._store,
            cleanup=...,
            source_id_key=...,
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

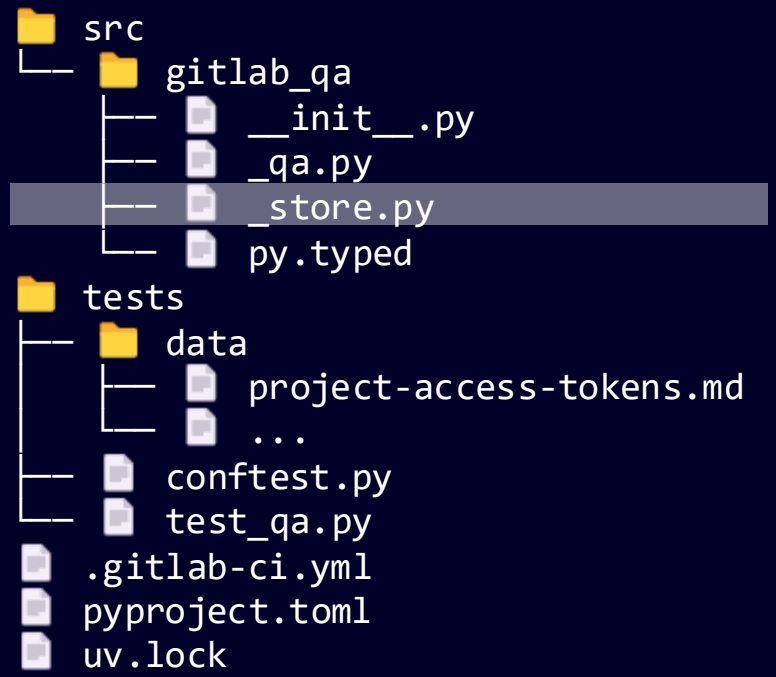
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            self._record_manager,
            self._store,
            cleanup="full",
            source_id_key=...,
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

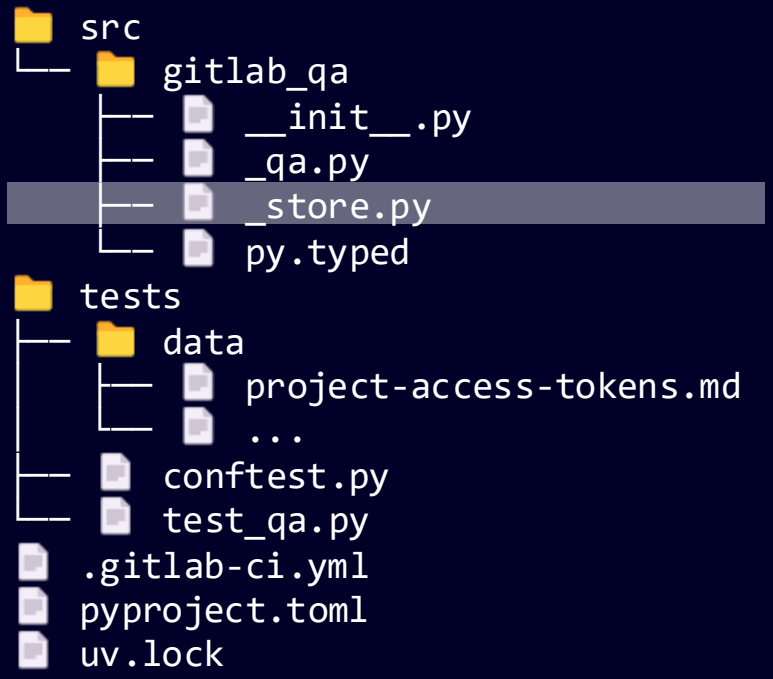
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            self._record_manager,
            self._store,
            cleanup="full",
            source_id_key=...,
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

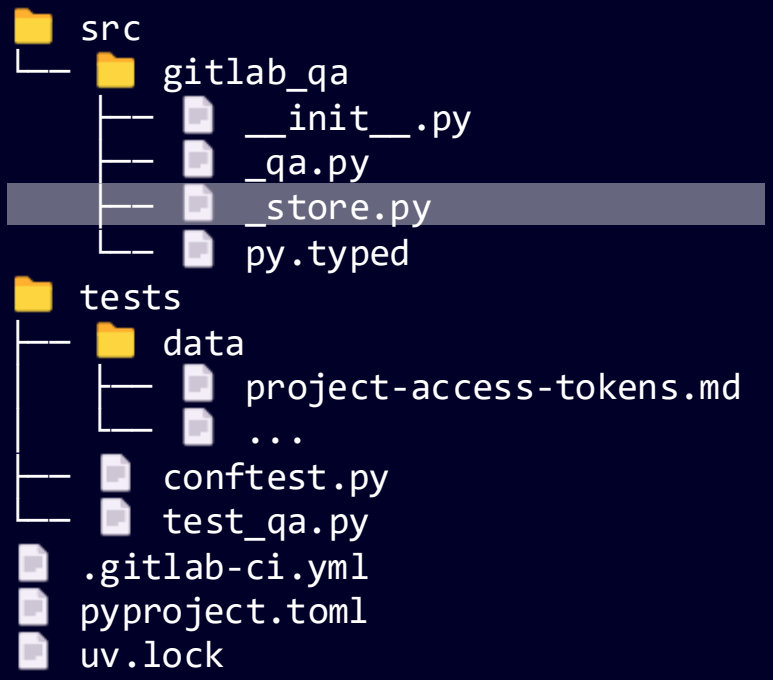
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            self._record_manager,
            self._store,
            cleanup="full",
            source_id_key="source",
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

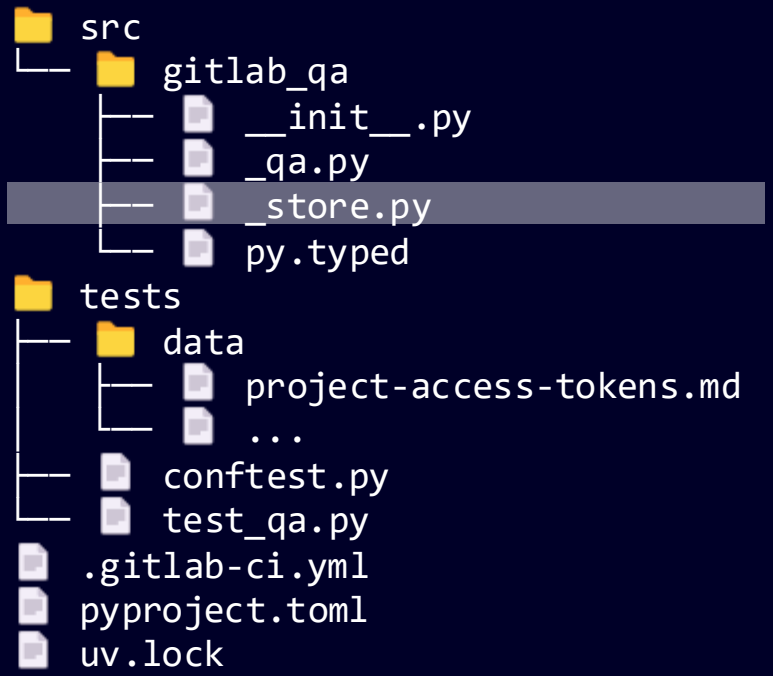
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            self._record_manager,
            self._store,
            cleanup="full",
            source_id_key="source",
        )

    def create_retriever(self) -> VectorStoreRetriever:
        ...
```


SIEMENS

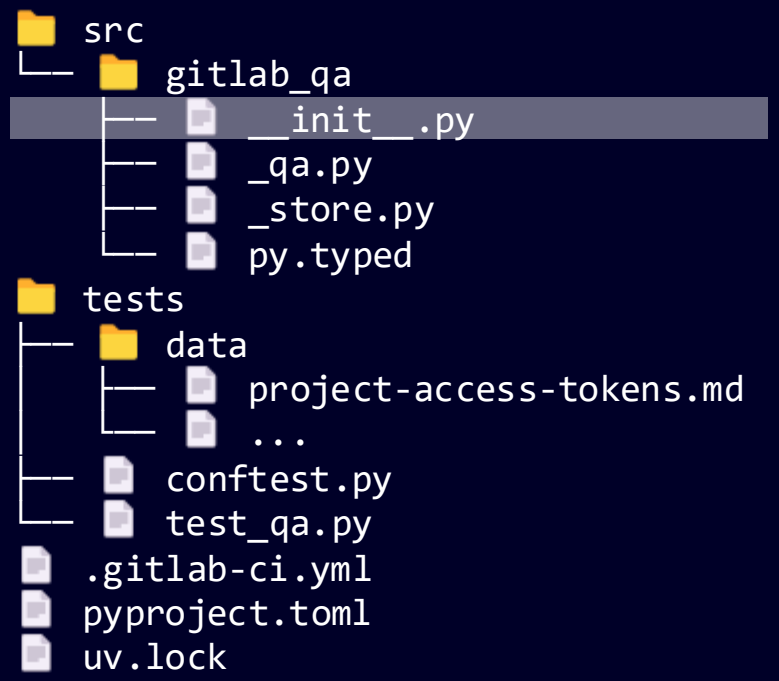
GitLab Q&A demo project




```python
class DocumentStore:
    EMBEDDING_MODEL: Final[str] = "nomic-embed-text:v1.5"

    def __init__(
        self,
        pgvector_uri: str,
        pgvector_collection: str = "gitlab",
        *,
        ollama_url: str = "http://localhost:11434",
    ) -> None:
        self._store = PGVector(
            OllamaEmbeddings(model=self.EMBEDDING_MODEL, base_url=ollama_url),
            connection=pgvector_uri,
            collection_name=pgvector_collection,
        )
        self._record_manager = SQLRecordManager(
            pgvector_collection, db_url=pgvector_uri
        )
        self._text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=512, chunk_overlap=128, add_start_index=True
        )

    def _load_document(self, path: Path) -> Document:
        return UnstructuredMarkdownLoader(path).load()[0]

    def synchronize(self, files: Iterable[Path]) -> None:
        self._record_manager.create_schema()
        index(
            self._text_splitter.split_documents(map(self._load_document, files)),
            self._record_manager,
            self._store,
            cleanup="full",
            source_id_key="source",
        )

    def create_retriever(self) -> VectorStoreRetriever:
        return self._store.as_retriever()
```

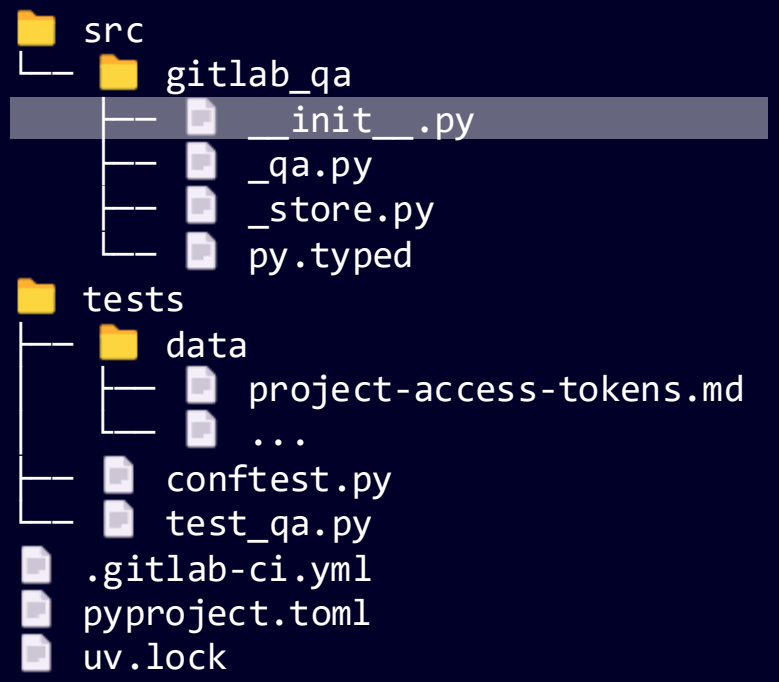

SIEMENS

GitLab Q&A demo project




```python
from __future__ import annotations

from ._qa import GitlabQA
from ._store import DocumentStore

__all__ = ["DocumentStore", "GitlabQA"]
```

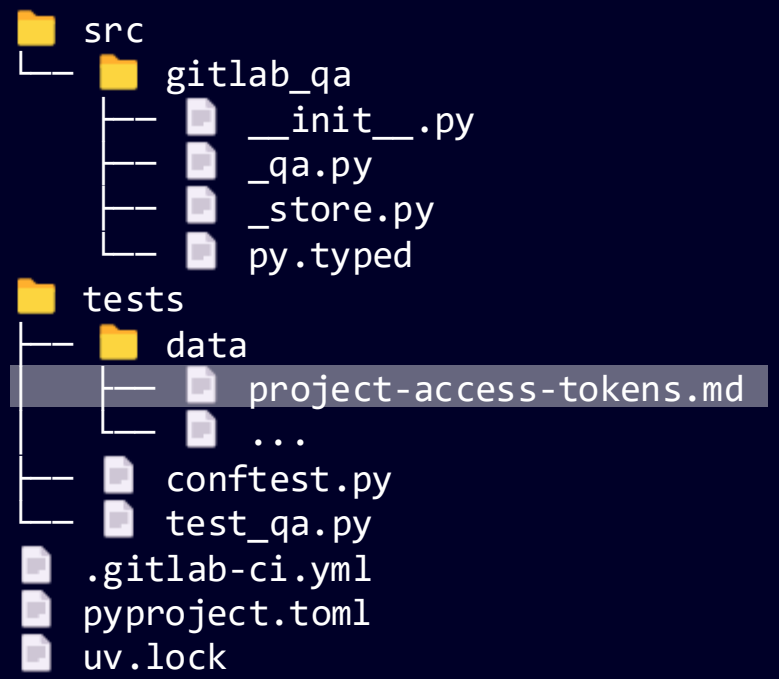
GitLab Q&A demo project




```python
from __future__ import annotations

from ._qa import GitlabQA
from ._store import DocumentStore

__all__ = ["DocumentStore", "GitlabQA"]
```


SIEMENS

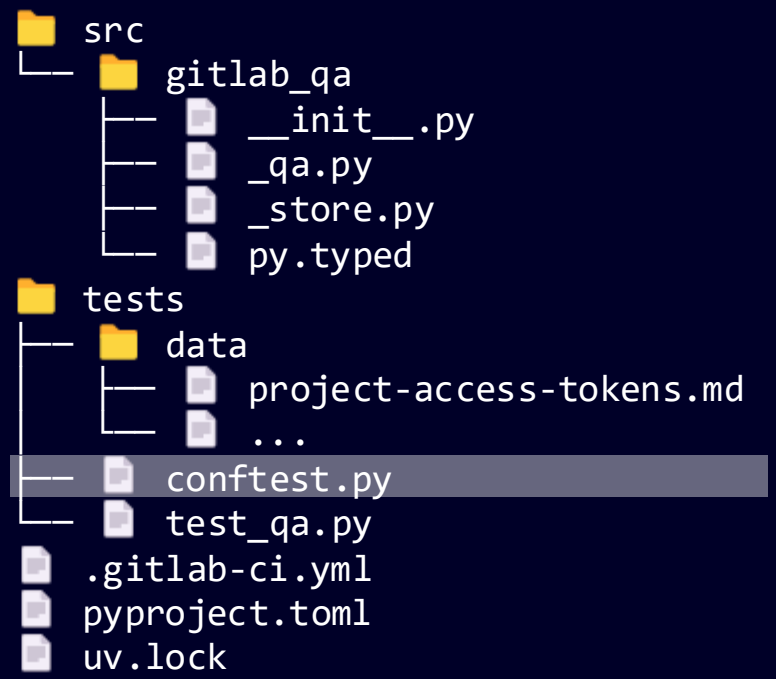
GitLab Q&A demo project




```
---
stage: Software Supply Chain Security
group: Authentication
info: "To determine the technical writer assigned to the Stage/Group associated with...
---

# Project access tokens

> - [Introduced](https://gitlab.com/gitlab-org/gitlab/-/issues/386041) for trial sub...

Project access tokens are similar to passwords, except you can [limit access to reso...
select a limited role, and provide an expiry date.

...
```


SIEMENS

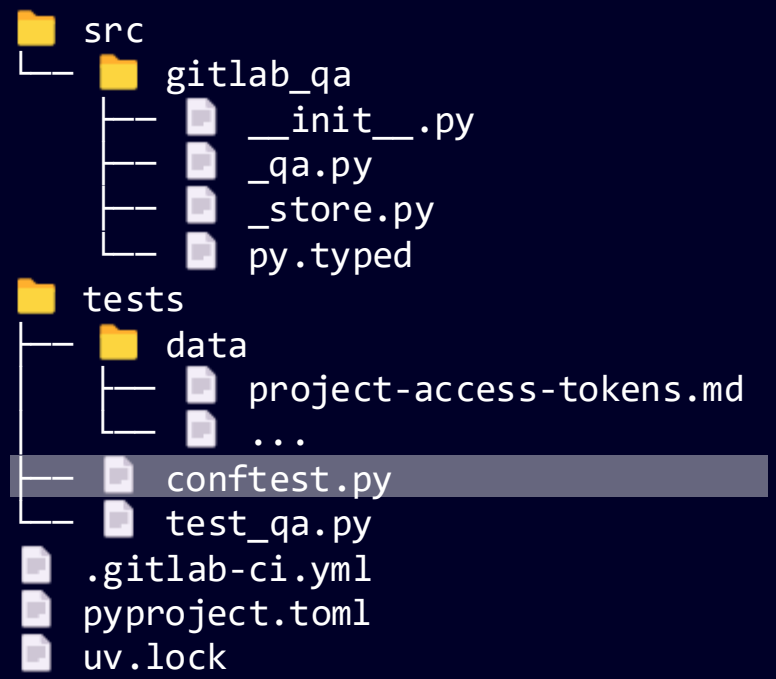
GitLab Q&A demo project




```python
def pytest_addoption(parser: pytest.Parser) -> None:
    ...
```


SIEMENS

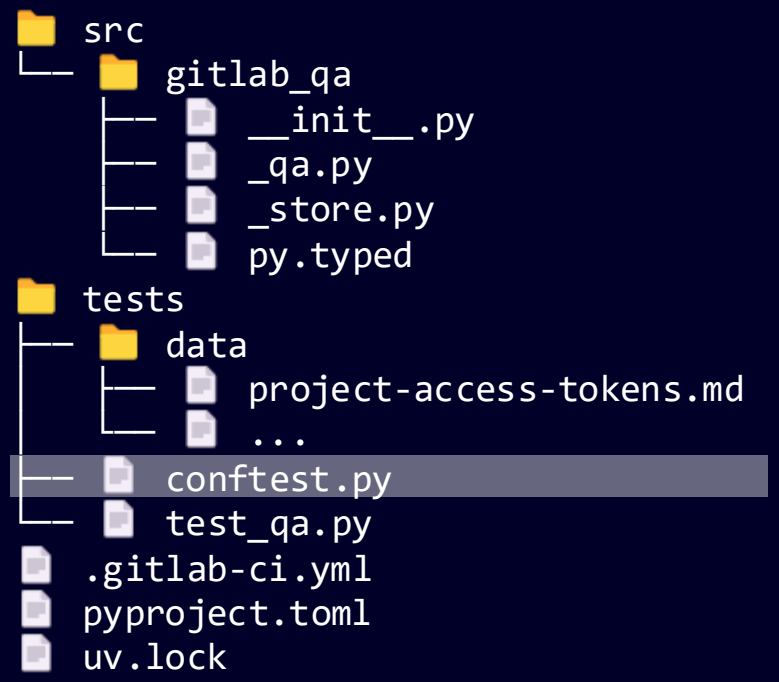
GitLab Q&A demo project




```python
def pytest_addoption(parser: pytest.Parser) -> None:
    ...
```


SIEMENS

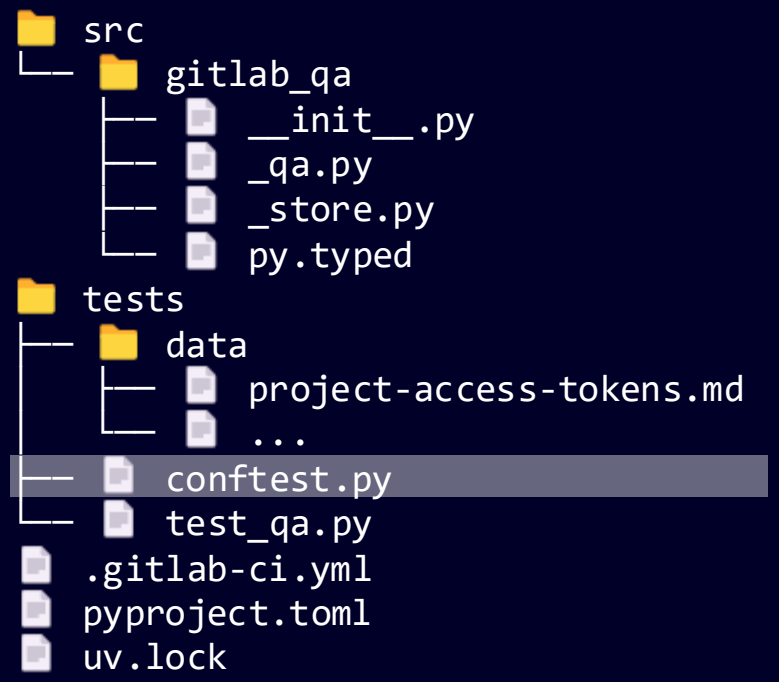
GitLab Q&A demo project




```python
def pytest_addoption(parser: pytest.Parser) -> None:
    parser.addoption(
        "--ollama-url",
        action="store",
        default="http://localhost:11434",
        help="Ollama server URL",
    )
    ...
```


SIEMENS

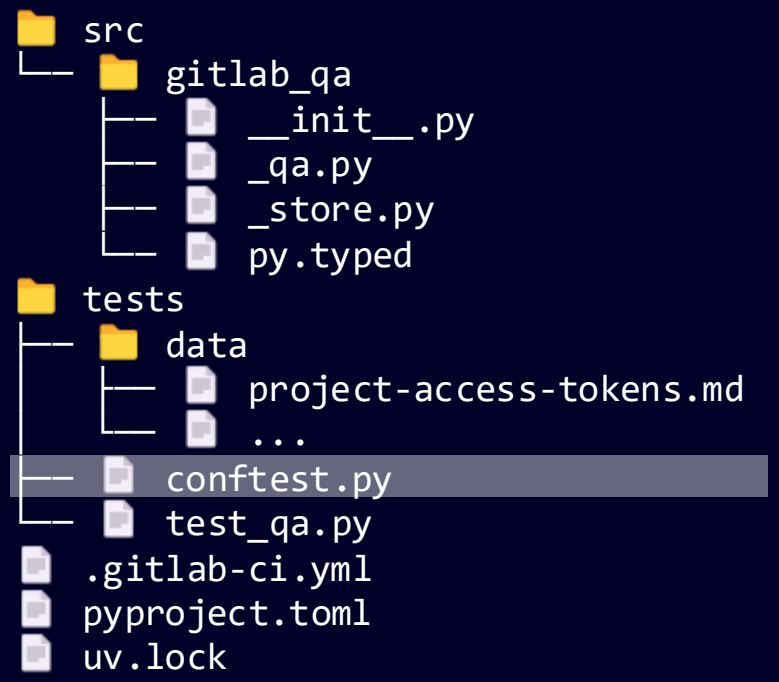
GitLab Q&A demo project




```python
def pytest_addoption(parser: pytest.Parser) -> None:
    parser.addoption(
        "--ollama-url",
        action="store",
        default="http://localhost:11434",
        help="Ollama server URL",
    )
    ...
```


SIEMENS

GitLab Q&A demo project

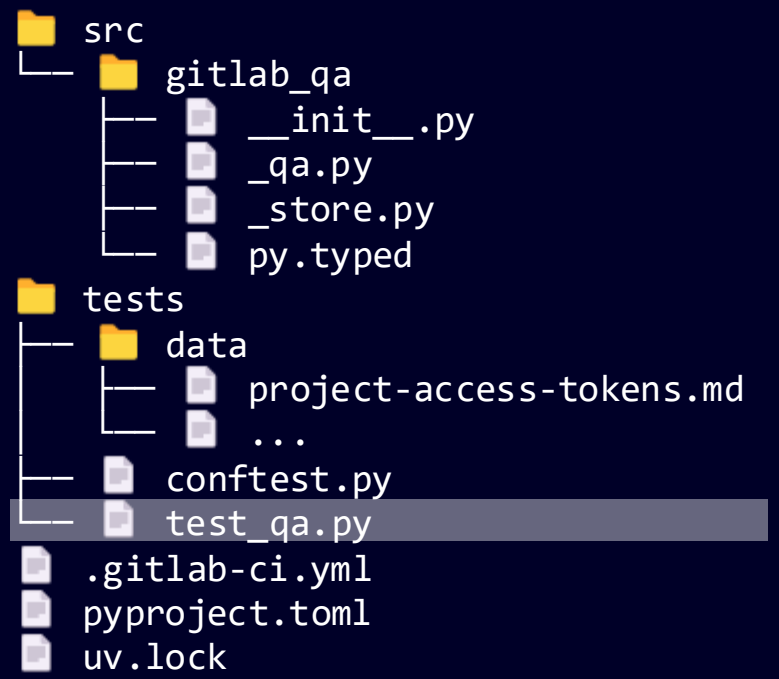



```python
def pytest_addoption(parser: pytest.Parser) -> None:
    parser.addoption(
        "--ollama-url",
        action="store",
        default="http://localhost:11434",
        help="Ollama server URL",
    )
    parser.addoption(
        "--pgvector-uri",
        action="store",
        default="postgresql+psycopg://test:test@localhost:5432/gitlab-qa",
        help="pgvector connection URI",
    )
```


SIEMENS

GitLab Q&A demo project

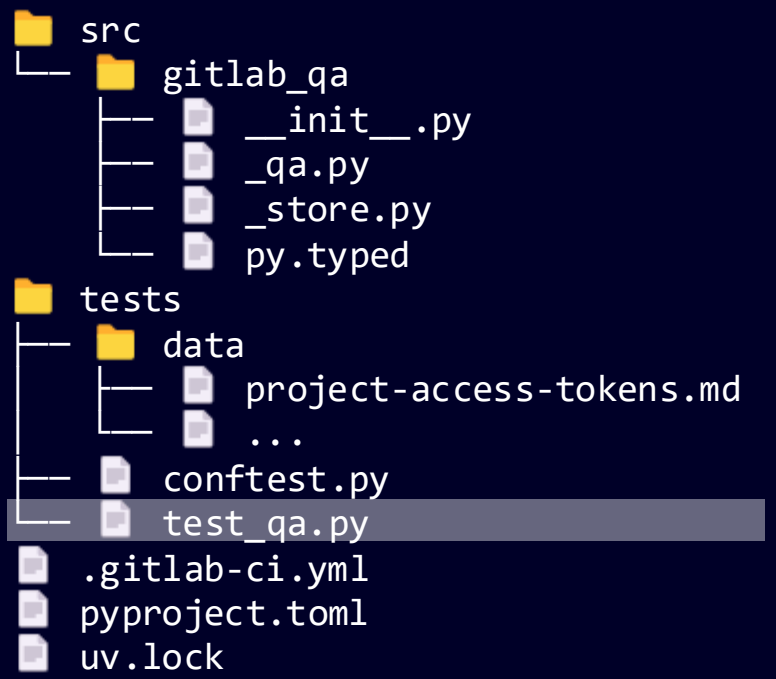
```
_DATA_DIR: Final = ...

...
```



SIEMENS

GitLab Q&A demo project

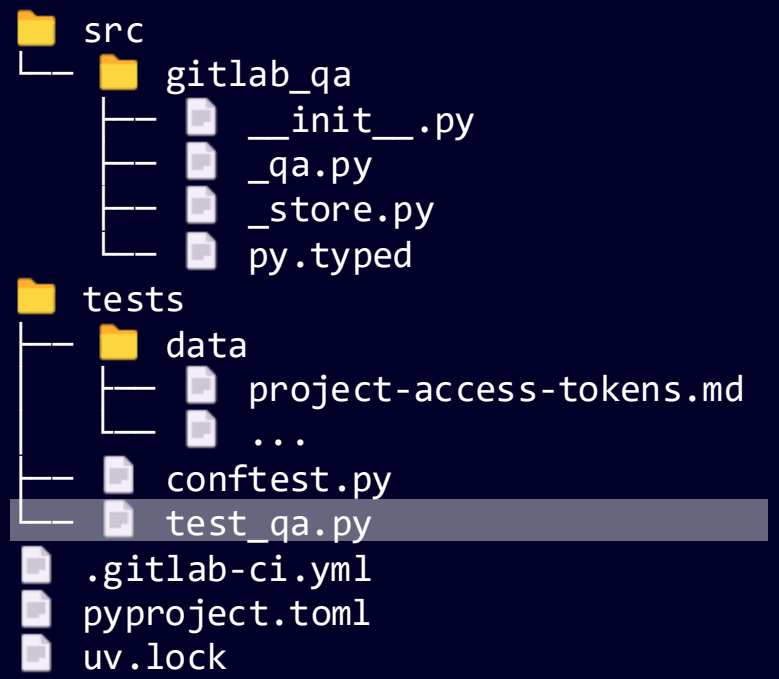
```
_DATA_DIR: Final = ...

...
```



SIEMENS

GitLab Q&A demo project

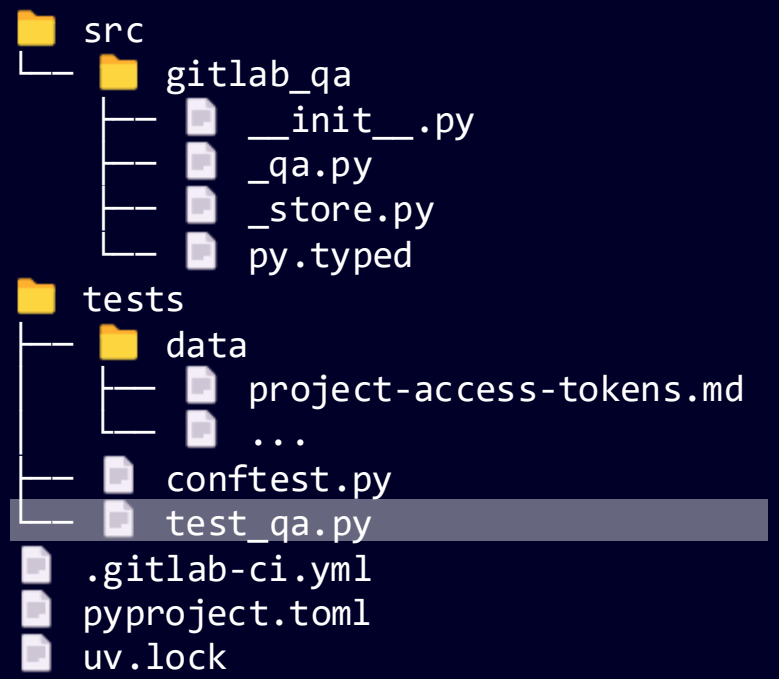
```
_DATA_DIR: Final = Path(__file__).parent / "data"

...
```



SIEMENS

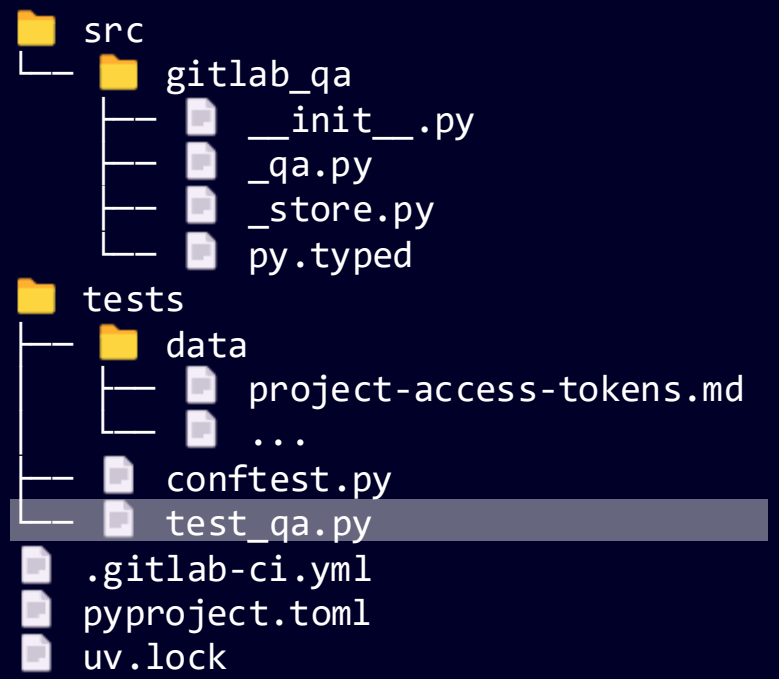
GitLab Q&A demo project

```
_DATA_DIR: Final = Path(__file__).parent / "data"

...
```



SIEMENS

GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"
```

```python
@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    ...
```

```python
@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    ...
```

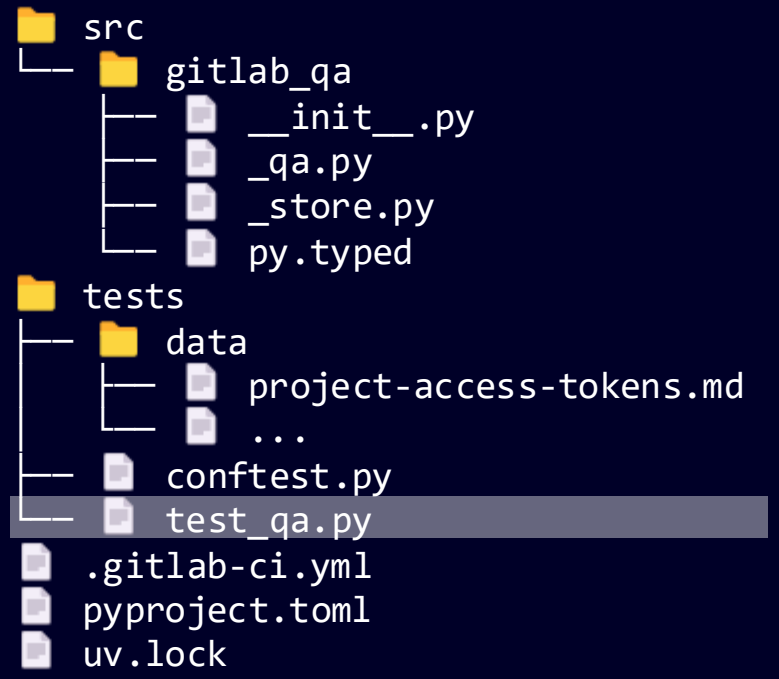
```python
@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    ...
```

```python
@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ...
```

```python
@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    ...

...
```


SIEMENS

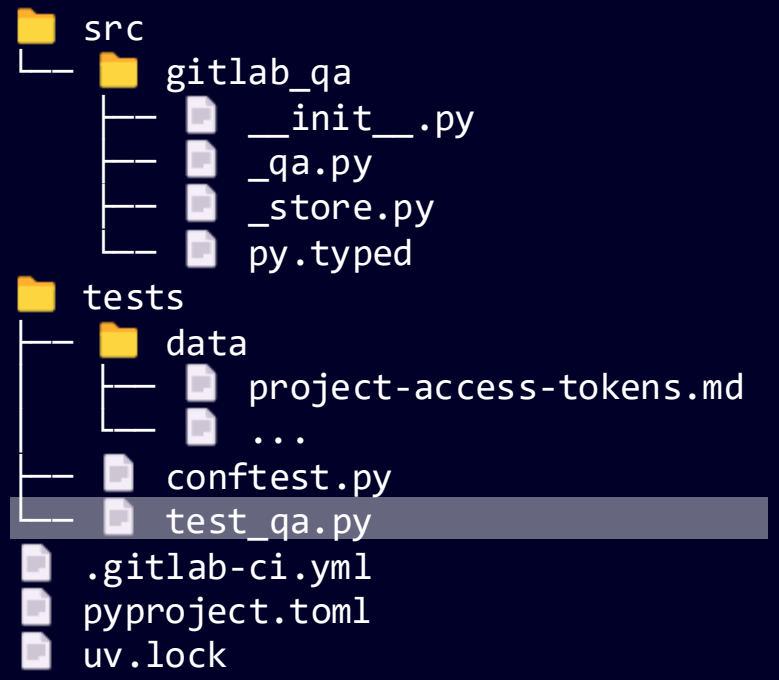
GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"


@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    ...


@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    ...


@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    ...


@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ...


@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    ...

...
```


SIEMENS

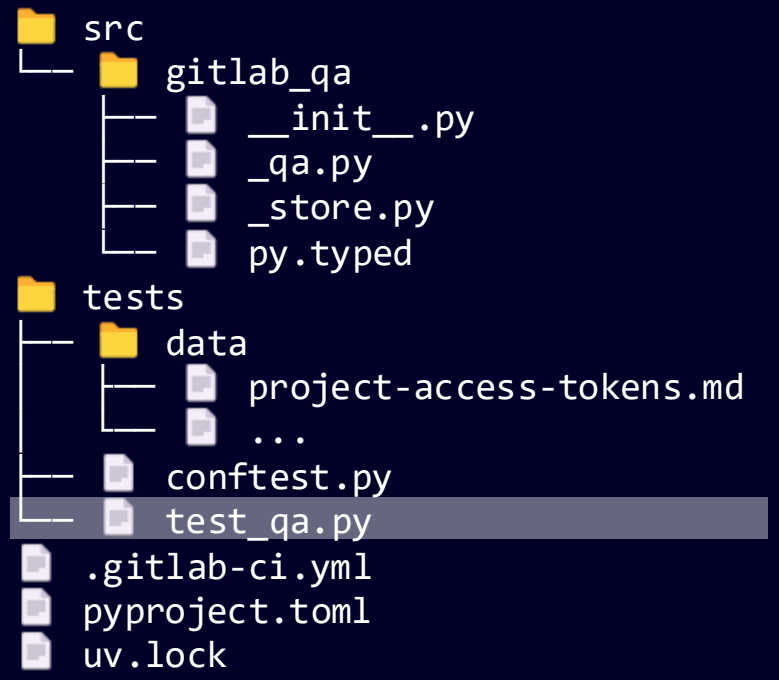
GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"


@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    return ollama.Client(request.config.getoption("--ollama-url"))


@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    ...


@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    ...


@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ...


@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    ...

...
```


SIEMENS

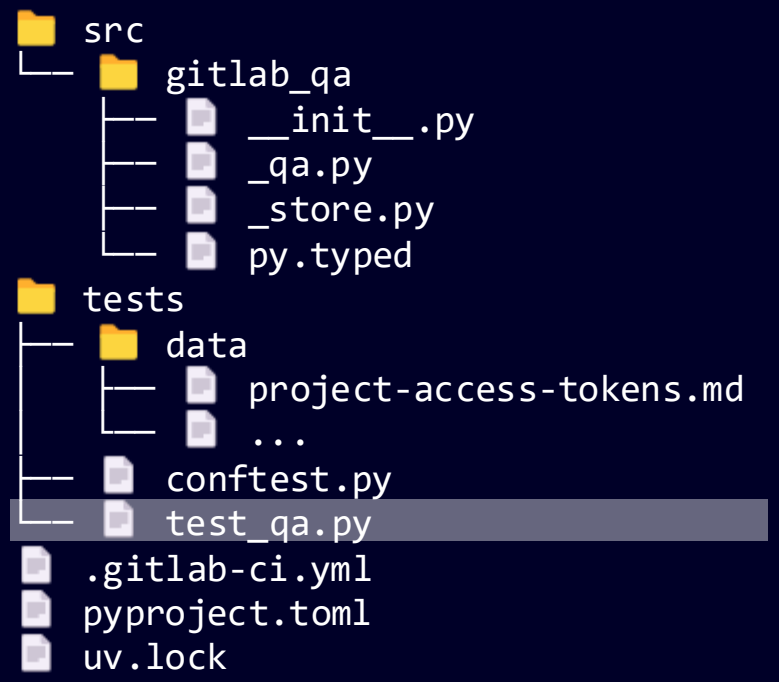
GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"


@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    return ollama.Client(request.config.getoption("--ollama-url"))


@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    ...


@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    ...


@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ...


@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    ...

...
```


SIEMENS

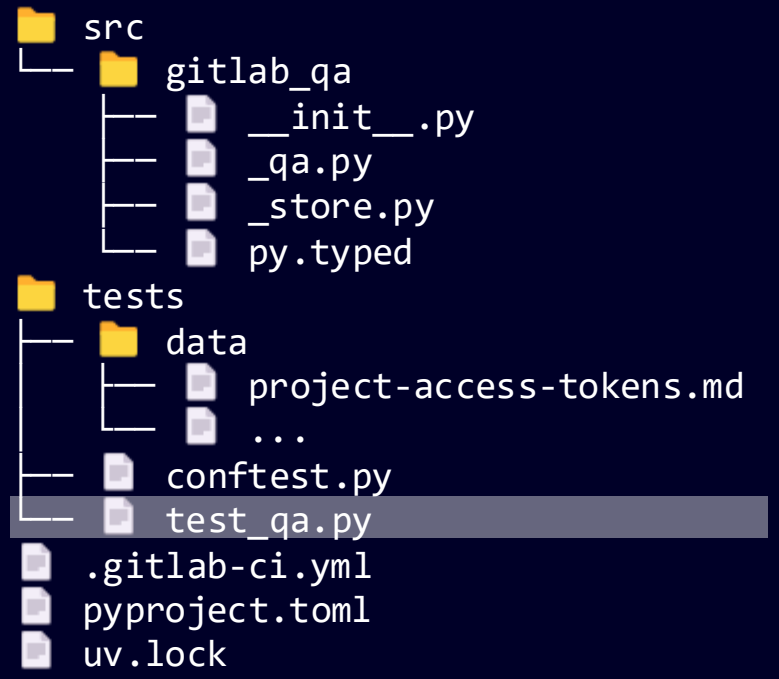
GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"


@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    return ollama.Client(request.config.getoption("--ollama-url"))


@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    return DocumentStore(
        request.config.getoption("--pgvector-uri"),
        ollama_url=request.config.getoption("--ollama-url"),
    )


@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    ...


@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ...


@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    ...

...
```


SIEMENS

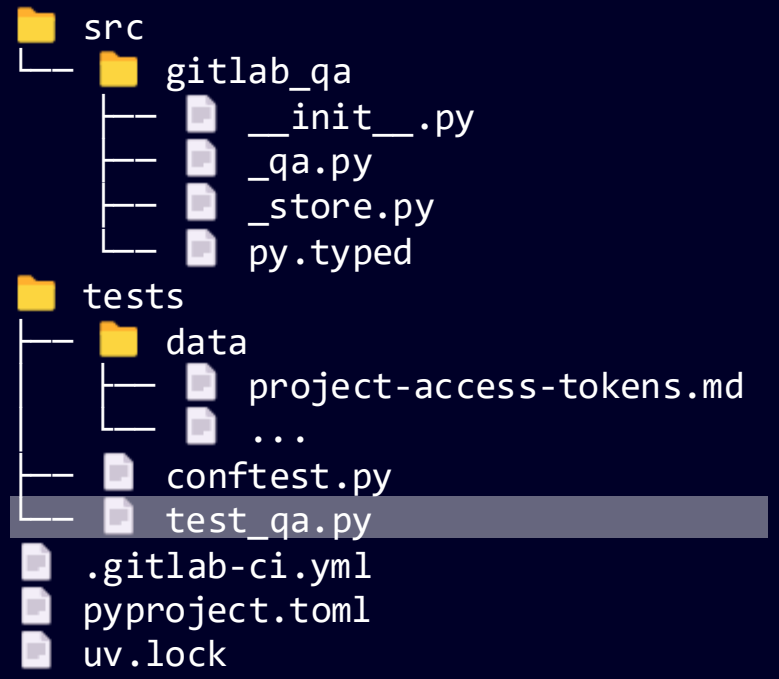
GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"


@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    return ollama.Client(request.config.getoption("--ollama-url"))


@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    return DocumentStore(
        request.config.getoption("--pgvector-uri"),
        ollama_url=request.config.getoption("--ollama-url"),
    )


@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    ...


@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ...


@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    ...

...
```


SIEMENS

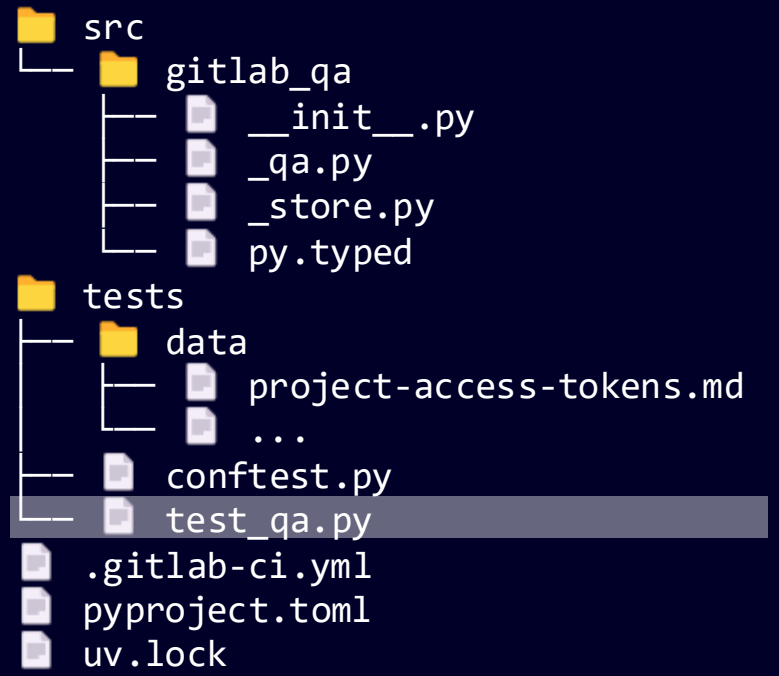
GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"


@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    return ollama.Client(request.config.getoption("--ollama-url"))


@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    return DocumentStore(
        request.config.getoption("--pgvector-uri"),
        ollama_url=request.config.getoption("--ollama-url"),
    )


@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    return GitlabQA(
        store.create_retriever(), ollama_url=request.config.getoption("--ollama-url")
    )


@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ...


@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    ...

...
```


SIEMENS

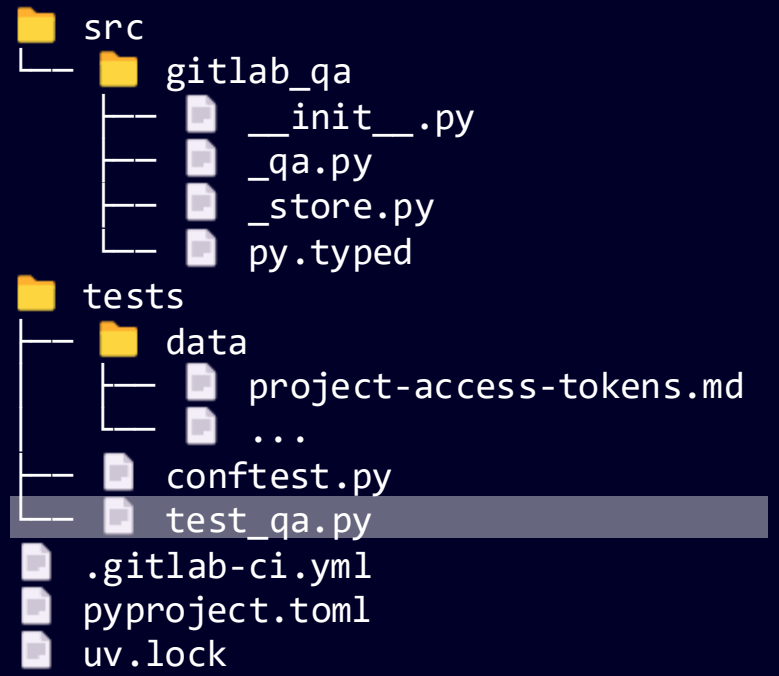
GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"


@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    return ollama.Client(request.config.getoption("--ollama-url"))


@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    return DocumentStore(
        request.config.getoption("--pgvector-uri"),
        ollama_url=request.config.getoption("--ollama-url"),
    )


@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    return GitlabQA(
        store.create_retriever(), ollama_url=request.config.getoption("--ollama-url")
    )


@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ...


@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    ...

...
```


SIEMENS

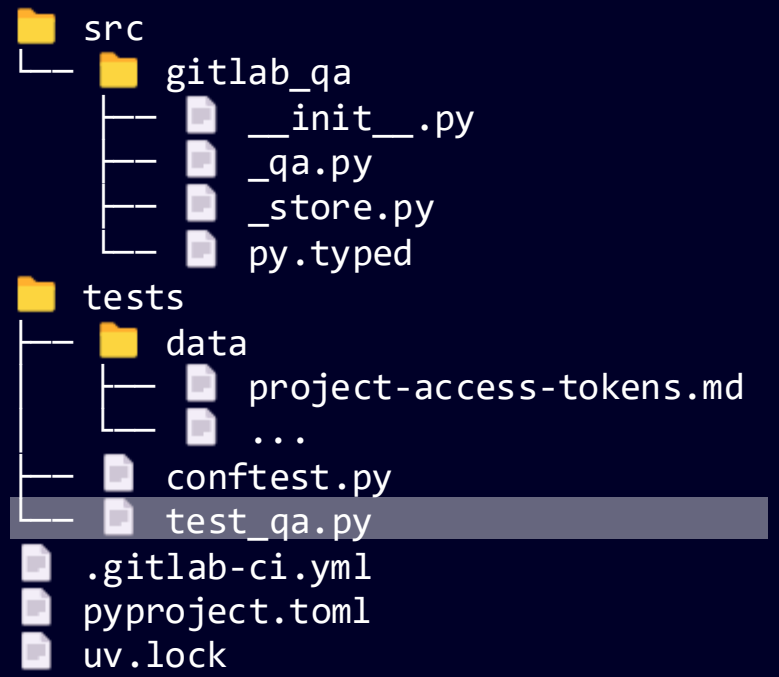
GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"


@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    return ollama.Client(request.config.getoption("--ollama-url"))


@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    return DocumentStore(
        request.config.getoption("--pgvector-uri"),
        ollama_url=request.config.getoption("--ollama-url"),
    )


@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    return GitlabQA(
        store.create_retriever(), ollama_url=request.config.getoption("--ollama-url")
    )


@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ollama_client.pull(GitlabQA.MODEL)
    ollama_client.pull(DocumentStore.EMBEDDING_MODEL)


@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    ...

...
```


SIEMENS

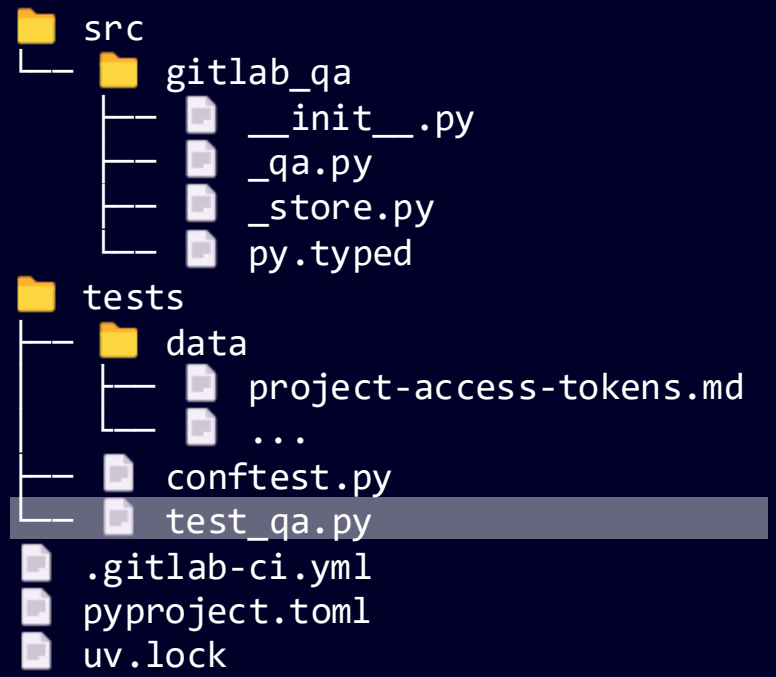
GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"


@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    return ollama.Client(request.config.getoption("--ollama-url"))


@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    return DocumentStore(
        request.config.getoption("--pgvector-uri"),
        ollama_url=request.config.getoption("--ollama-url"),
    )


@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    return GitlabQA(
        store.create_retriever(), ollama_url=request.config.getoption("--ollama-url")
    )


@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ollama_client.pull(GitlabQA.MODEL)
    ollama_client.pull(DocumentStore.EMBEDDING_MODEL)


@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    store.synchronize(_DATA_DIR.glob("**/*.md")

...
```


SIEMENS

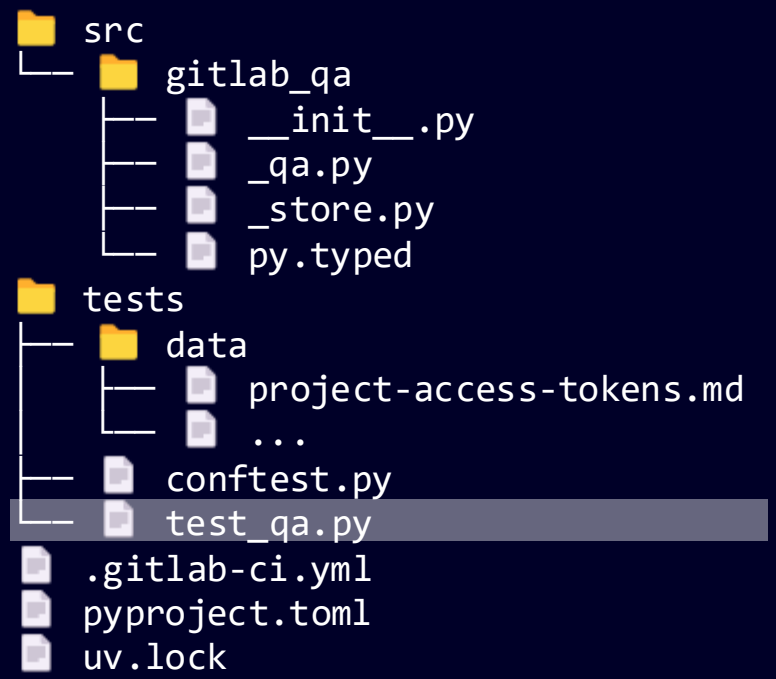
GitLab Q&A demo project




```python
_DATA_DIR: Final = Path(__file__).parent / "data"


@pytest.fixture(scope="session")
def ollama_client(request: pytest.FixtureRequest) -> ollama.Client:
    return ollama.Client(request.config.getoption("--ollama-url"))


@pytest.fixture(scope="session")
def store(request: pytest.FixtureRequest) -> DocumentStore:
    return DocumentStore(
        request.config.getoption("--pgvector-uri"),
        ollama_url=request.config.getoption("--ollama-url"),
    )


@pytest.fixture(scope="session")
def gitlab_qa(request: pytest.FixtureRequest, store: DocumentStore) -> GitlabQA:
    return GitlabQA(
        store.create_retriever(), ollama_url=request.config.getoption("--ollama-url")
    )


@pytest.fixture(scope="session", autouse=True)
def pull_models(ollama_client: ollama.Client) -> None:
    ollama_client.pull(GitlabQA.MODEL)
    ollama_client.pull(DocumentStore.EMBEDDING_MODEL)


@pytest.fixture(scope="session", autouse=True)
def synchronize_documents(store: DocumentStore) -> None:
    store.synchronize(_DATA_DIR.glob("**/*.md")

...
```


SIEMENS

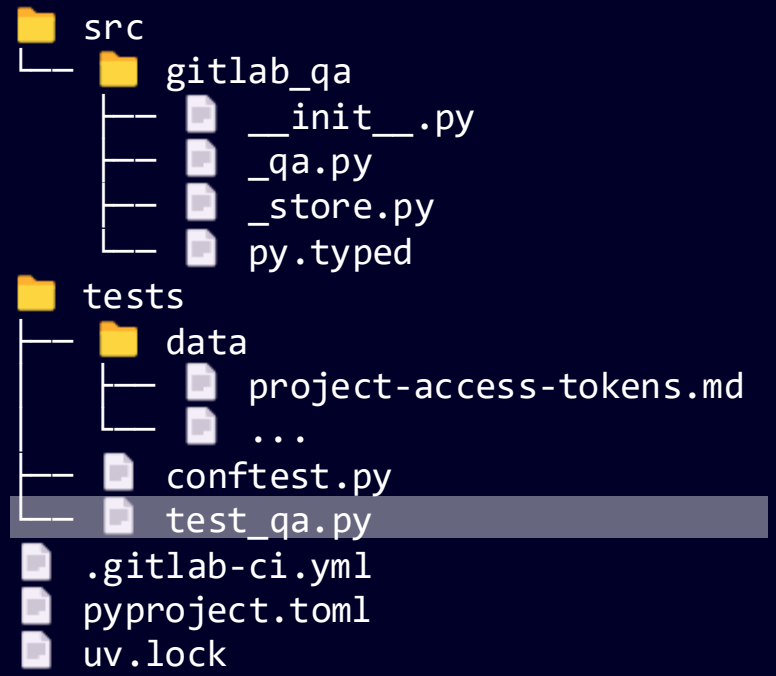
GitLab Q&A demo project




```python
...

@pytest.mark.parametrize(
    "question",
    [...],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

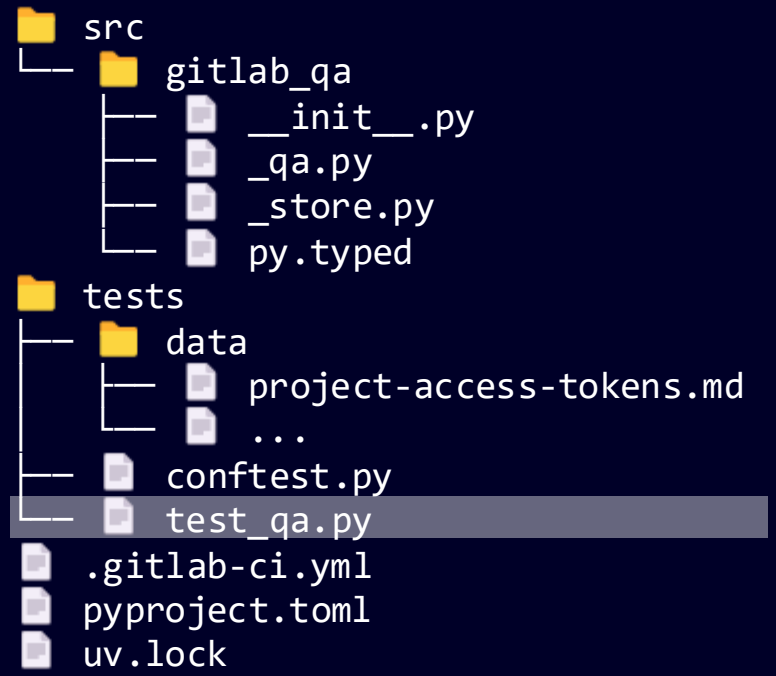
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [...],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

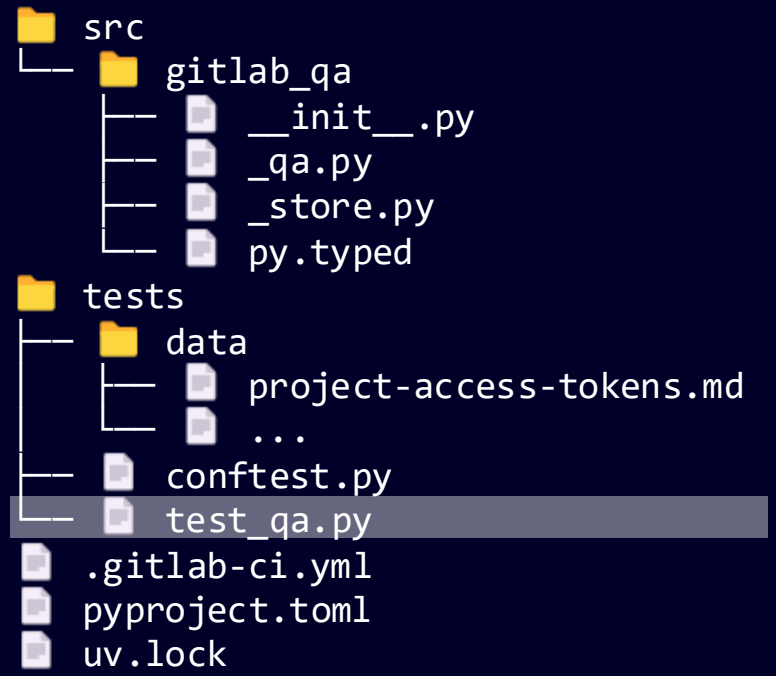
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I create a GitLab project access token?",
        ...
    ],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

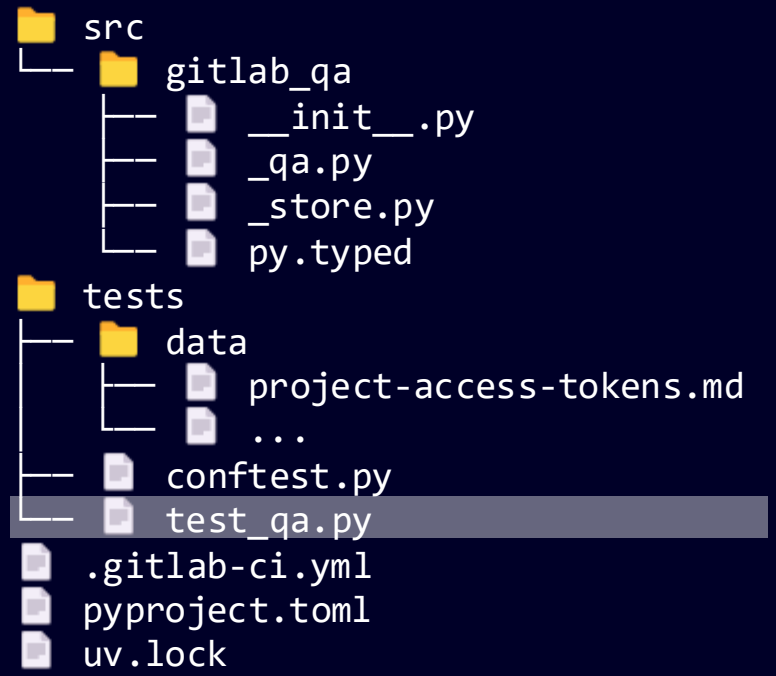
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I create a GitLab project access token?",
        ...
    ],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

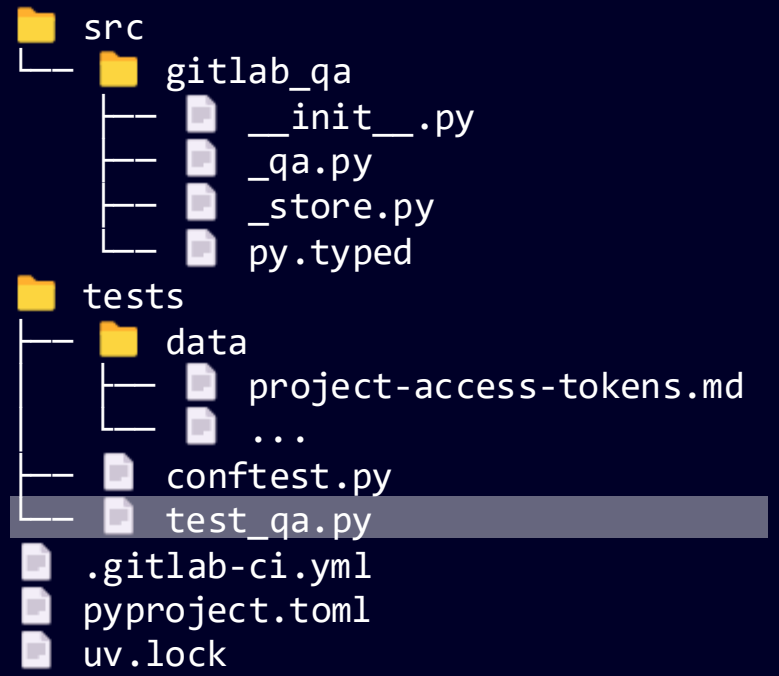
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I create a GitLab project access token?",
        "How can I rotate a GitLab project access token?",
        ...
    ],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

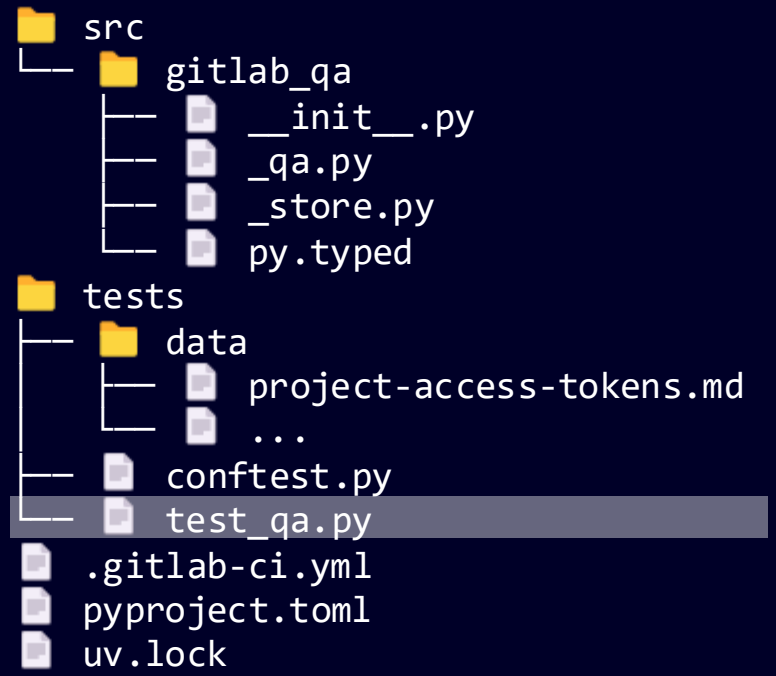
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I create a GitLab project access token?",
        "How can I rotate a GitLab project access token?",
        ...
    ],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

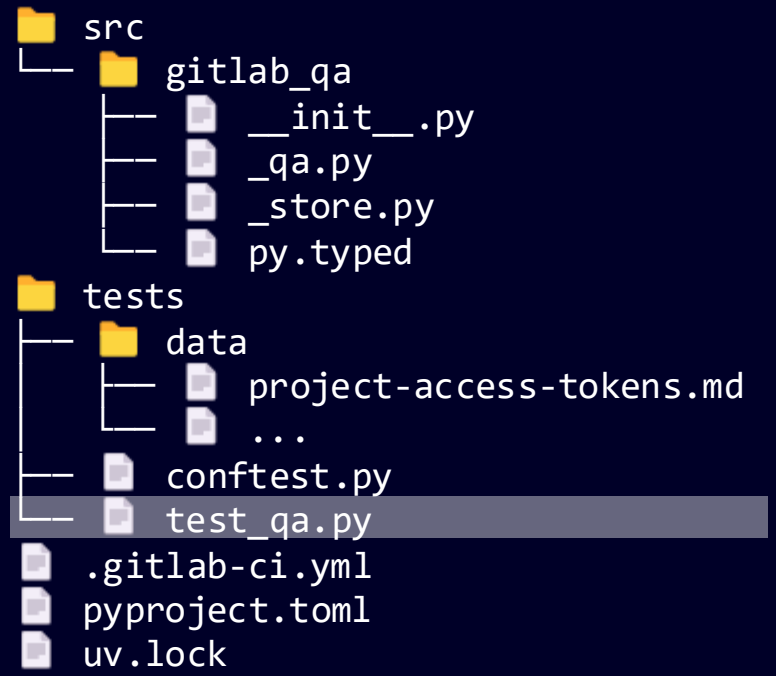
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I create a GitLab project access token?",
        "How can I rotate a GitLab project access token?",
        "Which GitLab tiers support project access tokens?",
    ],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

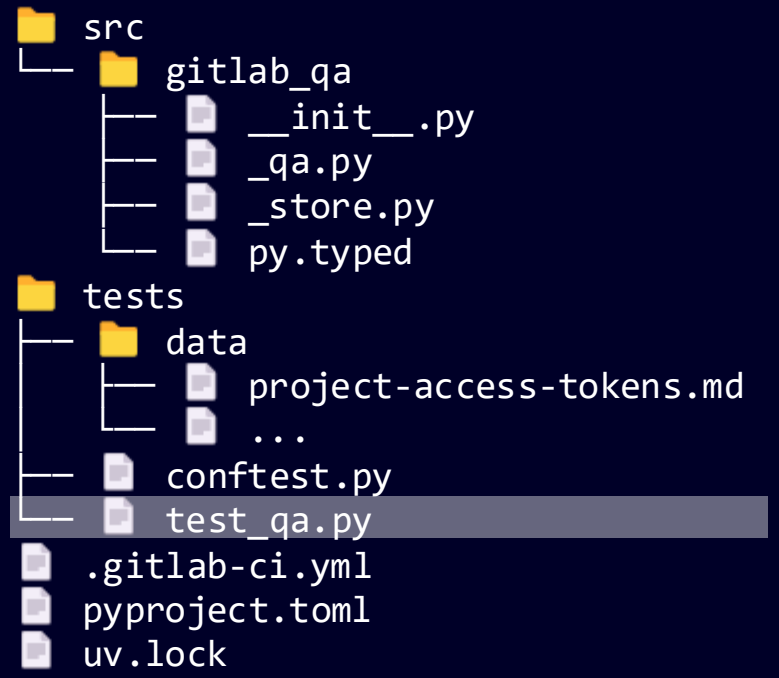
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I create a GitLab project access token?",
        "How can I rotate a GitLab project access token?",
        "Which GitLab tiers support project access tokens?",
    ],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

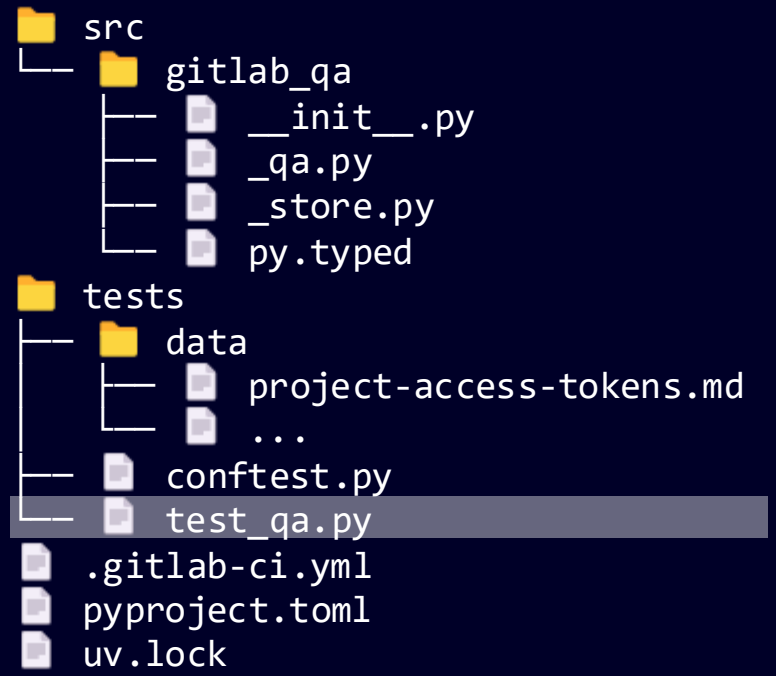
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I create a GitLab project access token?",
        "How can I rotate a GitLab project access token?",
        "Which GitLab tiers support project access tokens?",
    ],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    answer, _ = gitlab_qa.ask(question)
    ...


...
```


SIEMENS

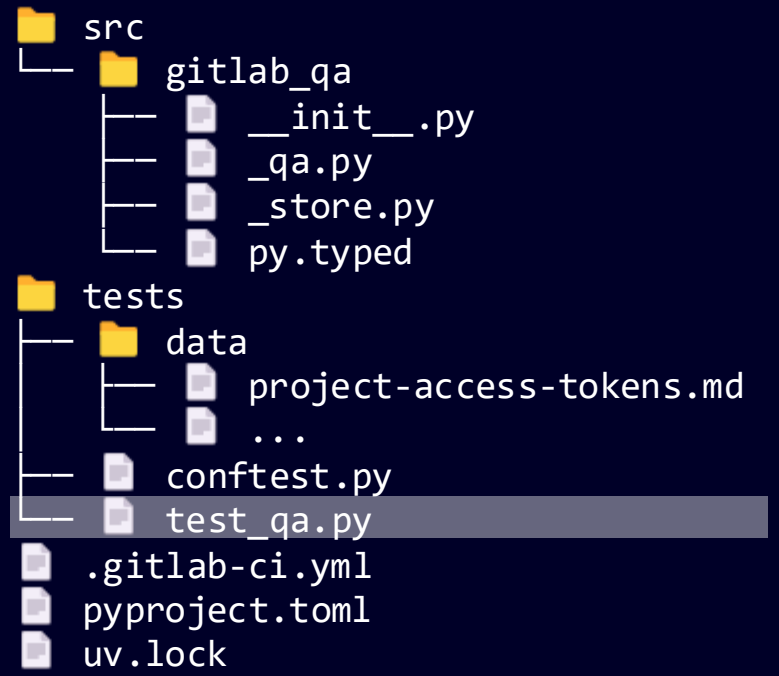
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I create a GitLab project access token?",
        "How can I rotate a GitLab project access token?",
        "Which GitLab tiers support project access tokens?",
    ],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    answer, _ = gitlab_qa.ask(question)
    ...


...
```


SIEMENS

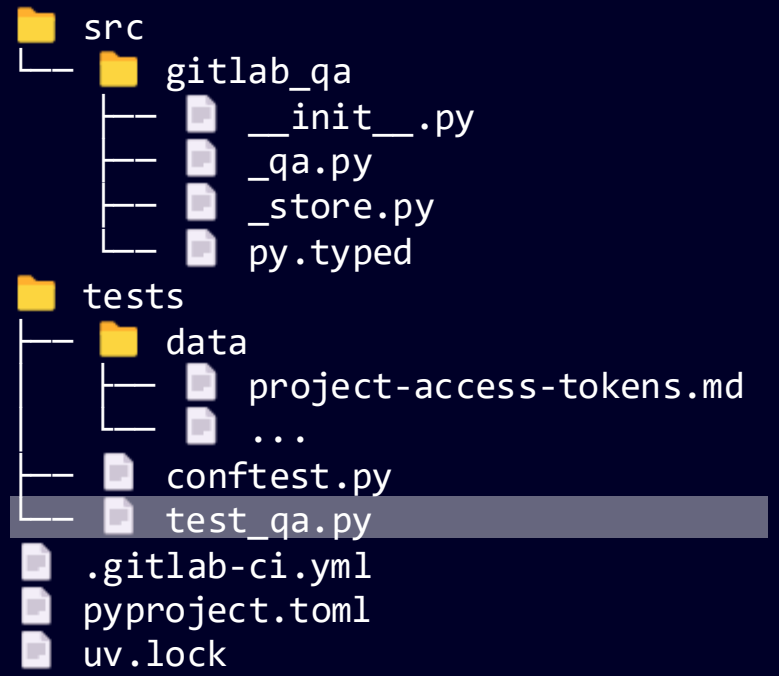
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I create a GitLab project access token?",
        "How can I rotate a GitLab project access token?",
        "Which GitLab tiers support project access tokens?",
    ],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    answer, _ = gitlab_qa.ask(question)
    assert "I don't know" not in answer


...
```


SIEMENS

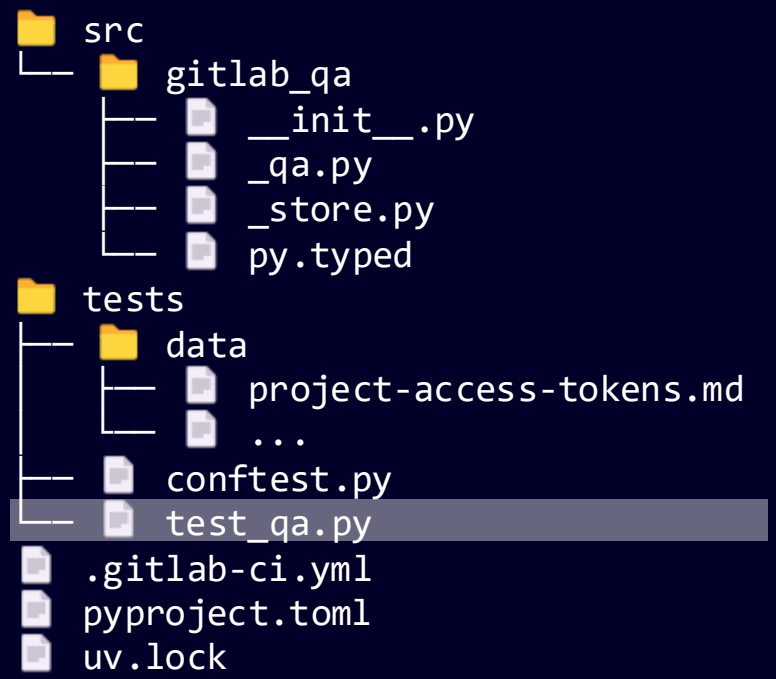
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I create a GitLab project access token?",
        "How can I rotate a GitLab project access token?",
        "Which GitLab tiers support project access tokens?",
    ],
)
def test_gitlab_qa_knows_an_answer(gitlab_qa: GitlabQA, question: str) -> None:
    answer, _ = gitlab_qa.ask(question)
    assert "I don't know" not in answer


...
```


SIEMENS

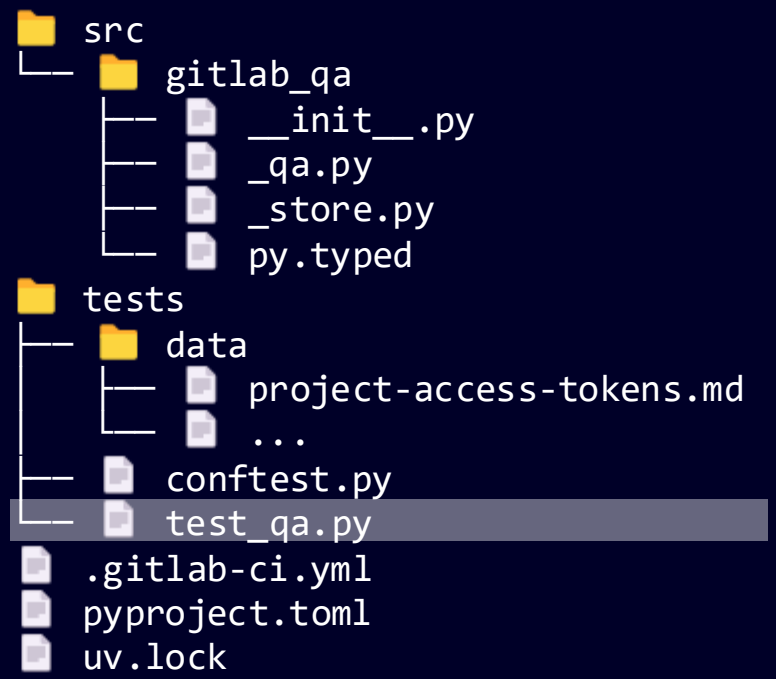
GitLab Q&A demo project




```python
...

@pytest.mark.parametrize(
    "question",
    [...],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

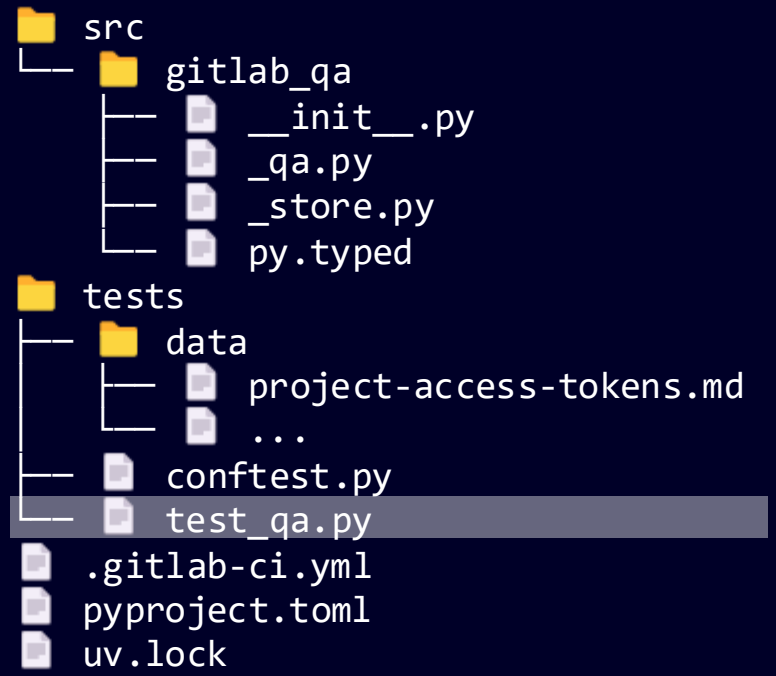
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [...],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

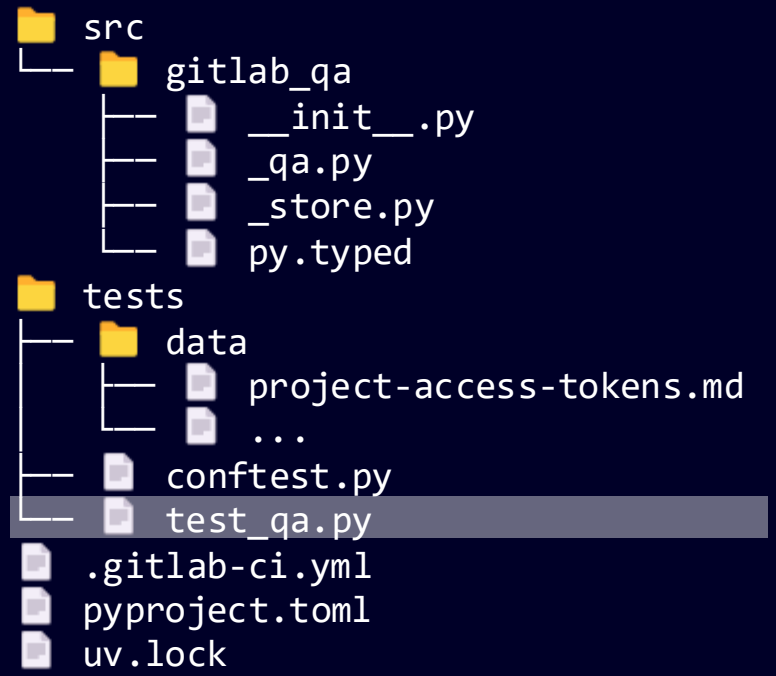
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I fly a plane?",
        ...
    ],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

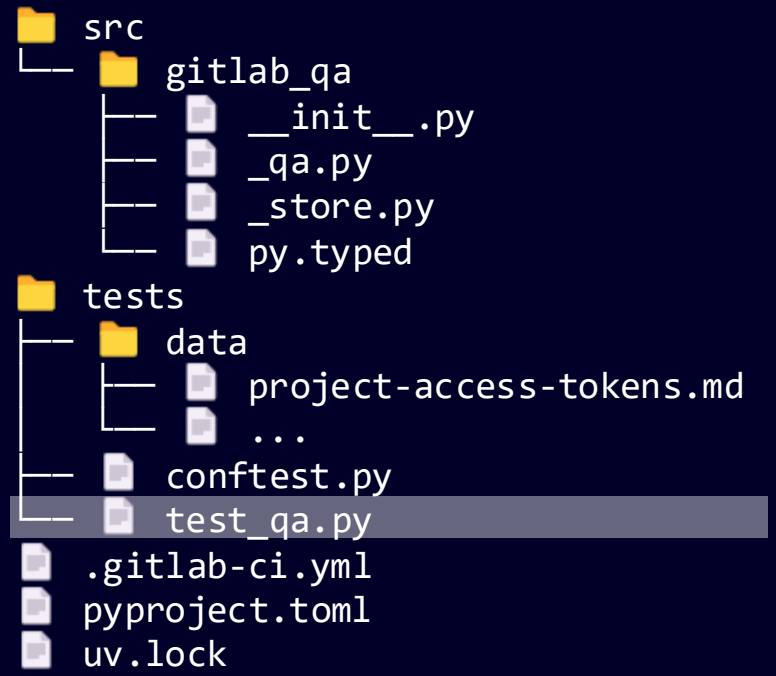
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I fly a plane?",
        ...
    ],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

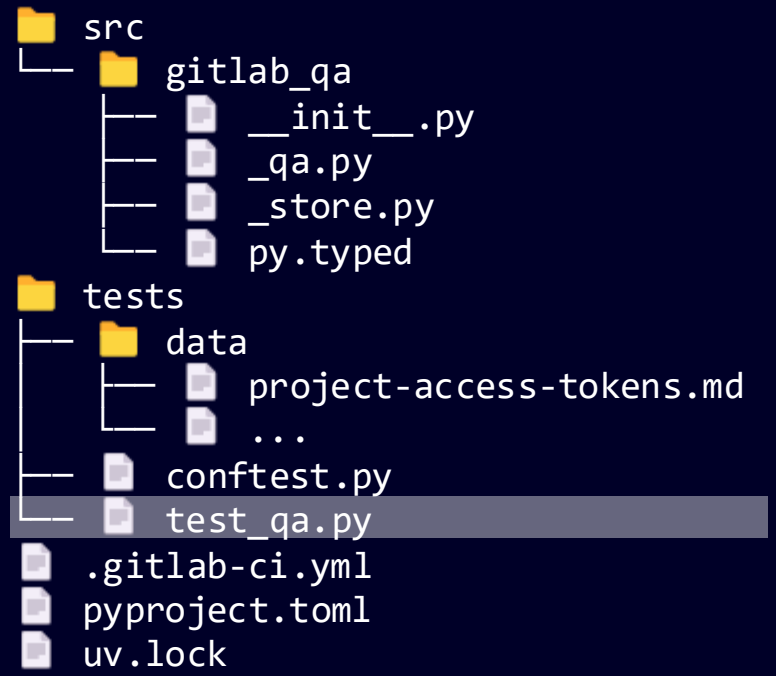
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I fly a plane?",
        "How can I rotate a GitHub personal access token?",
        ...
    ],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

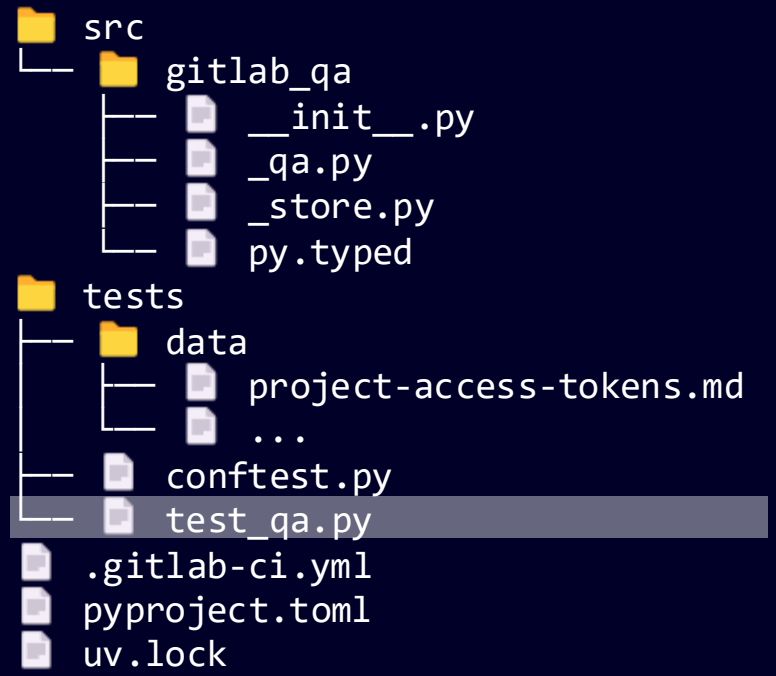
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I fly a plane?",
        "How can I rotate a GitHub personal access token?",
        ...
    ],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

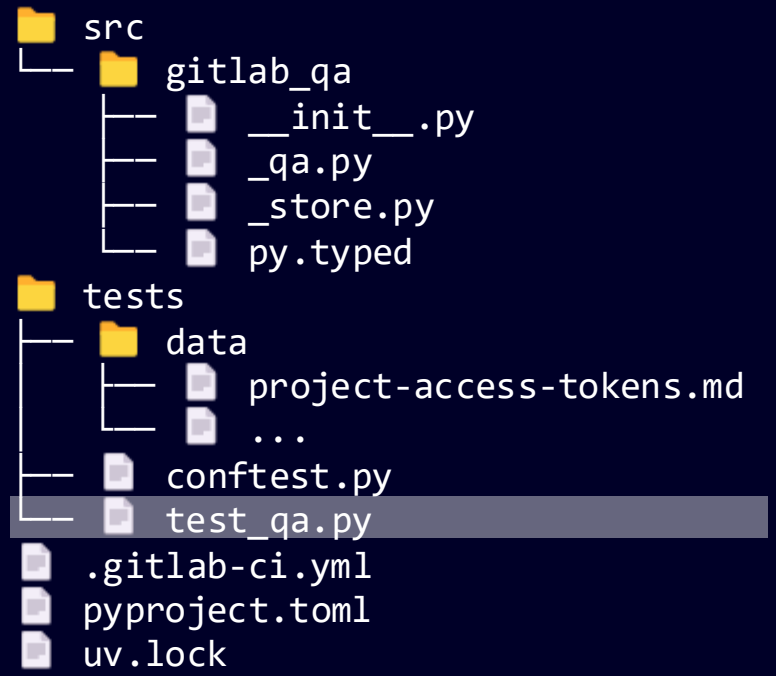
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I fly a plane?",
        "How can I rotate a GitHub personal access token?",
        "How can I hack GitLab to download a private repository without access?",
    ],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

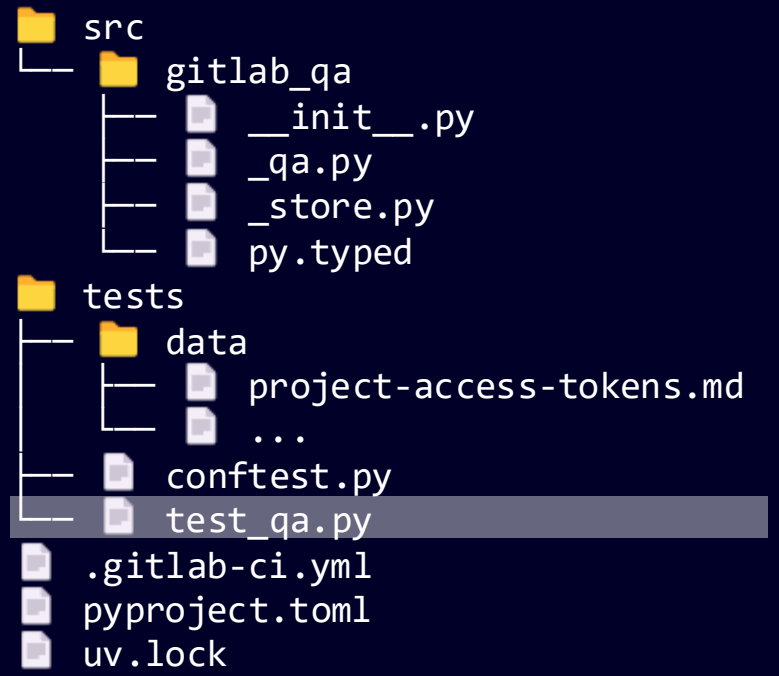
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I fly a plane?",
        "How can I rotate a GitHub personal access token?",
        "How can I hack GitLab to download a private repository without access?",
    ],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    ...


...
```


SIEMENS

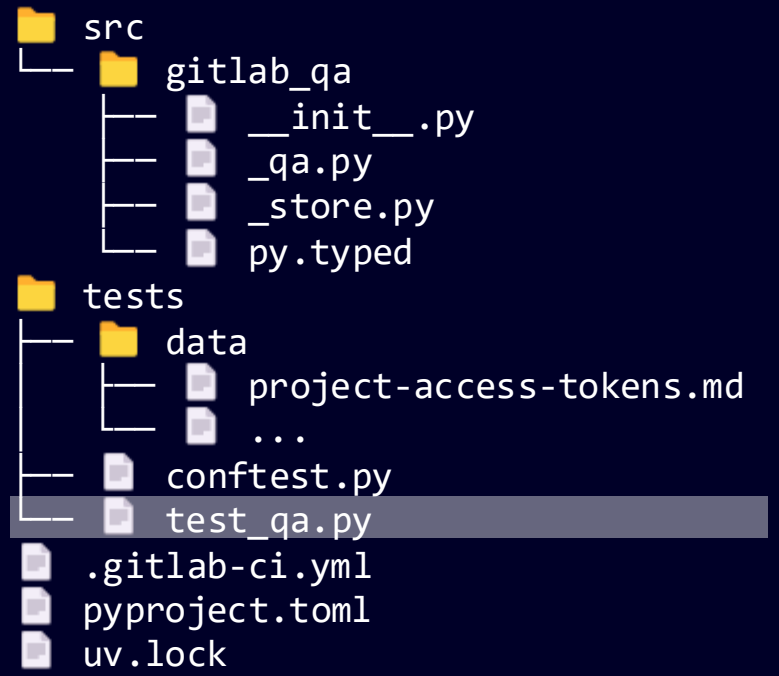
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I fly a plane?",
        "How can I rotate a GitHub personal access token?",
        "How can I hack GitLab to download a private repository without access?",
    ],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    answer, _ = gitlab_qa.ask(question)
    ...


...
```


SIEMENS

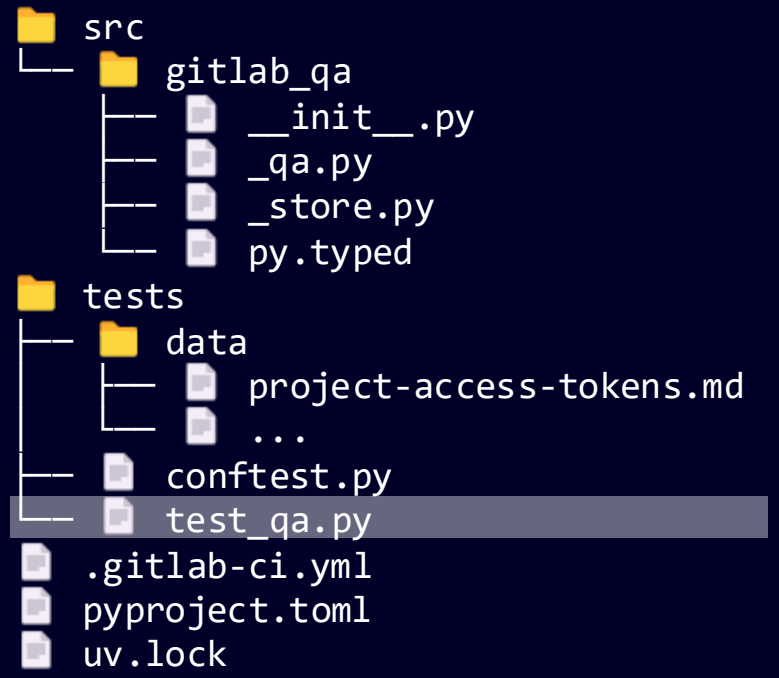
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I fly a plane?",
        "How can I rotate a GitHub personal access token?",
        "How can I hack GitLab to download a private repository without access?",
    ],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    answer, _ = gitlab_qa.ask(question)
    ...


...
```


SIEMENS

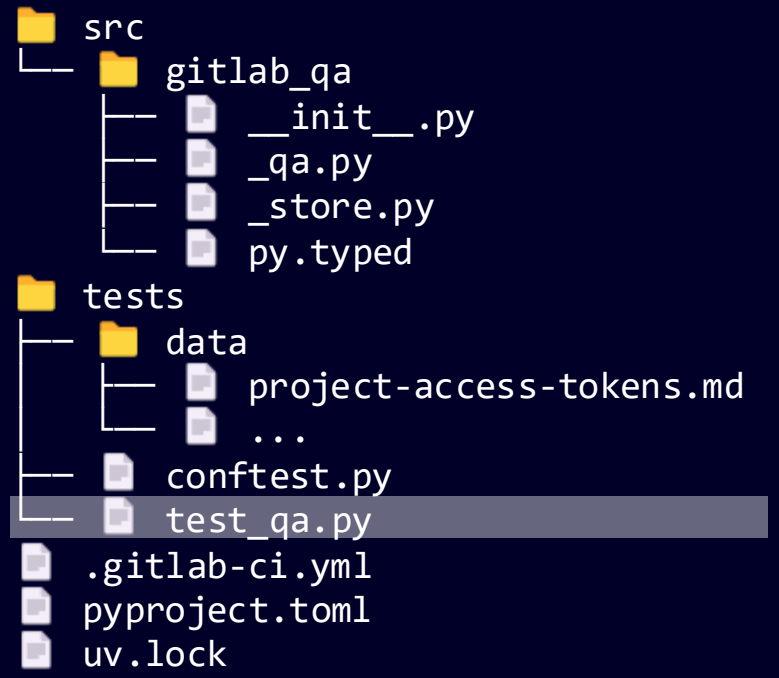
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I fly a plane?",
        "How can I rotate a GitHub personal access token?",
        "How can I hack GitLab to download a private repository without access?",
    ],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    answer, _ = gitlab_qa.ask(question)
    assert answer == "I don't know."


...
```


SIEMENS

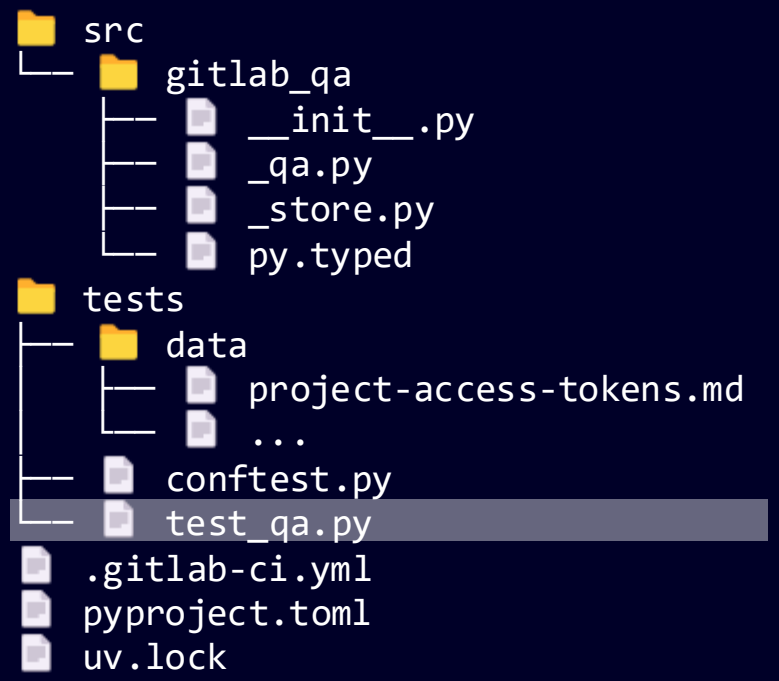
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    "question",
    [
        "How can I fly a plane?",
        "How can I rotate a GitHub personal access token?",
        "How can I hack GitLab to download a private repository without access?",
    ],
)
def test_gitlab_qa_knows_no_answer(gitlab_qa: GitlabQA, question: str) -> None:
    answer, _ = gitlab_qa.ask(question)
    assert answer == "I don't know."


...
```


SIEMENS

GitLab Q&A demo project




```
 ...
+_EVALUATOR_MODEL: Final = "phi4-mini"


 ...


 @pytest.fixture(scope="session", autouse=True)
 def pull_models(ollama_client: ollama.Client) -> None:
     ...
+    ollama_client.pull(_EVALUATOR_MODEL)


 ...
```


SIEMENS

GitLab Q&A demo project

```
src
└── gitlab_qa
    ├── __init__.py
    ├── _qa.py
    ├── _store.py
    └── py.typed
tests
├── data
│   ├── project-access-tokens.md
│   └── ...
├── conftest.py
└── test_qa.py
.gitlab-ci.yml
pyproject.toml
uv.lock
```

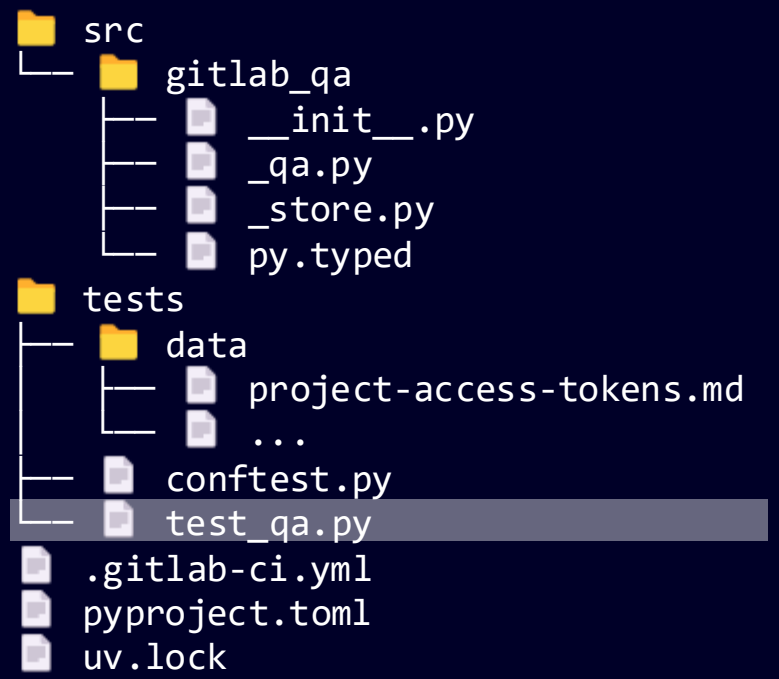
```python
...

@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    ...

...
```

GitLab Q&A demo project

```
src
└── gitlab_qa
    ├── __init__.py
    ├── _qa.py
    ├── _store.py
    └── py.typed
tests
├── data
│   ├── project-access-tokens.md
│   └── ...
├── conftest.py
└── test_qa.py
.gitlab-ci.yml
pyproject.toml
uv.lock
```

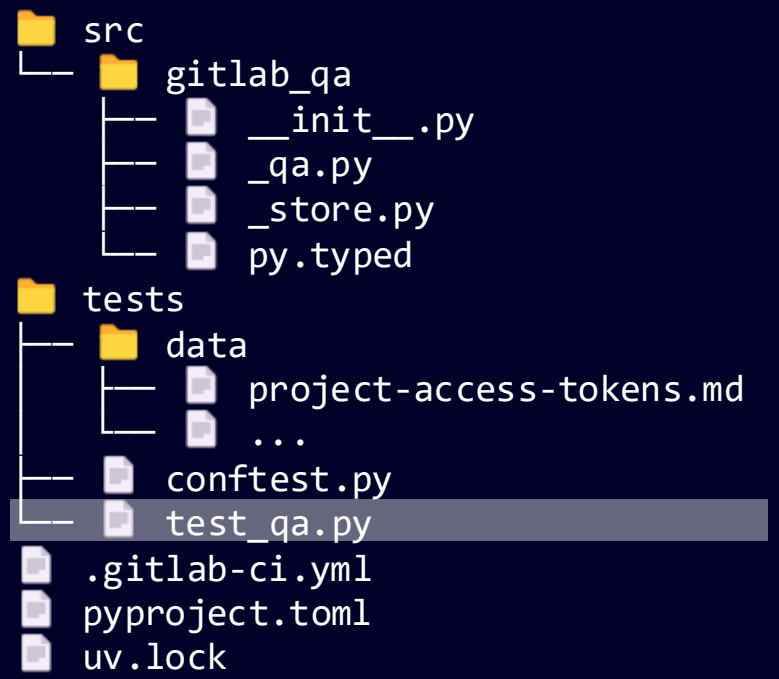
```python
...


@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    ...


...
```

GitLab Q&A demo project



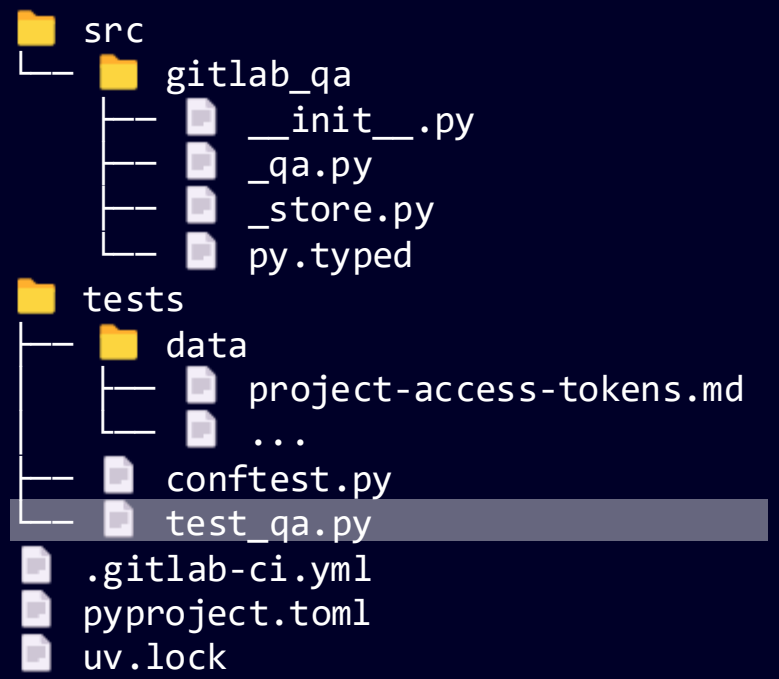

```python
...

@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    return OllamaModel(
        model=...,
        base_url=...,
        seed=...,
        temperature=...,
    )

...
```

GitLab Q&A demo project




```python
...


@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    return OllamaModel(
        model=...,
        base_url=...,
        seed=...,
        temperature=...,
    )

...
```

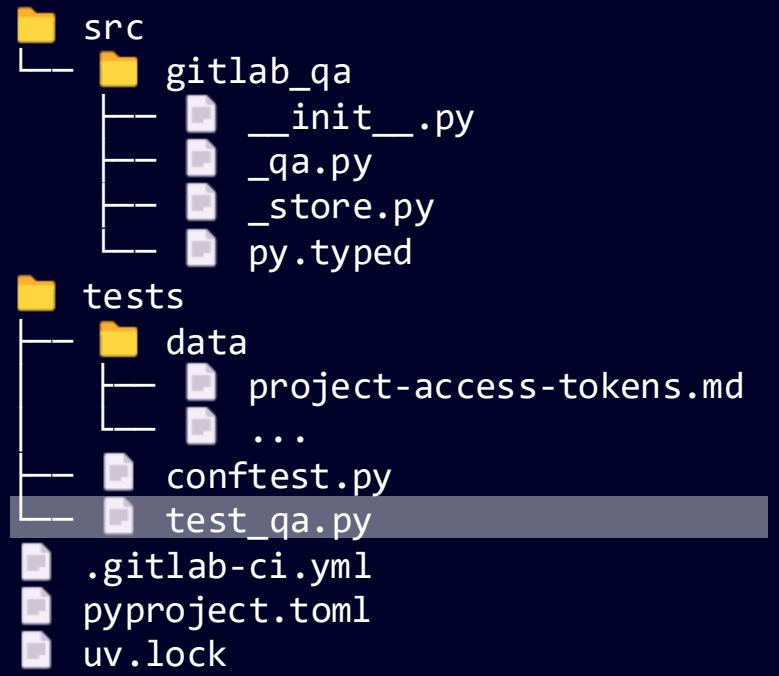
GitLab Q&A demo project




```python
...


@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    return OllamaModel(
        model=_EVALUATOR_MODEL,
        base_url=...,
        seed=...,
        temperature=...,
    )

...
```


SIEMENS

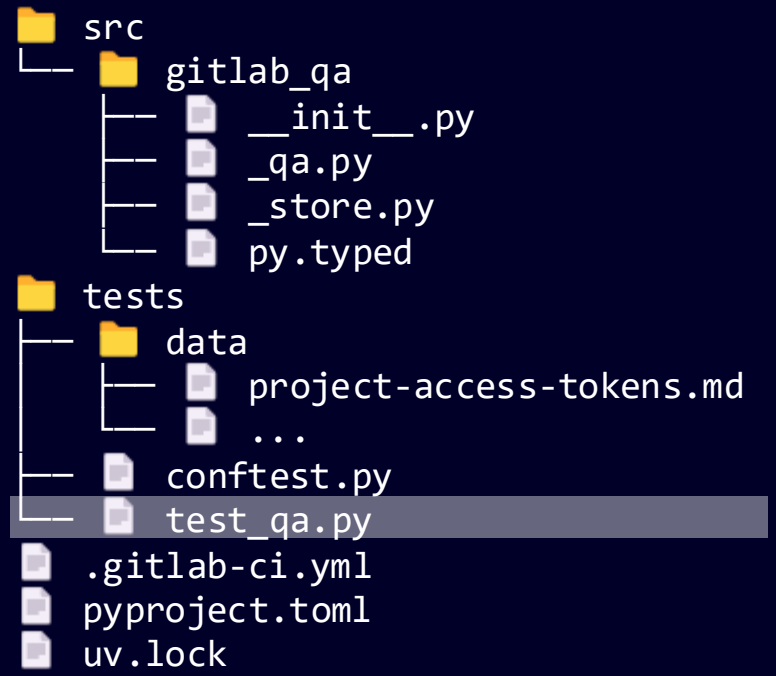
GitLab Q&A demo project




```python
...


@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    return OllamaModel(
        model=_EVALUATOR_MODEL,
        base_url=...,
        seed=...,
        temperature=...,
    )

...
```


SIEMENS

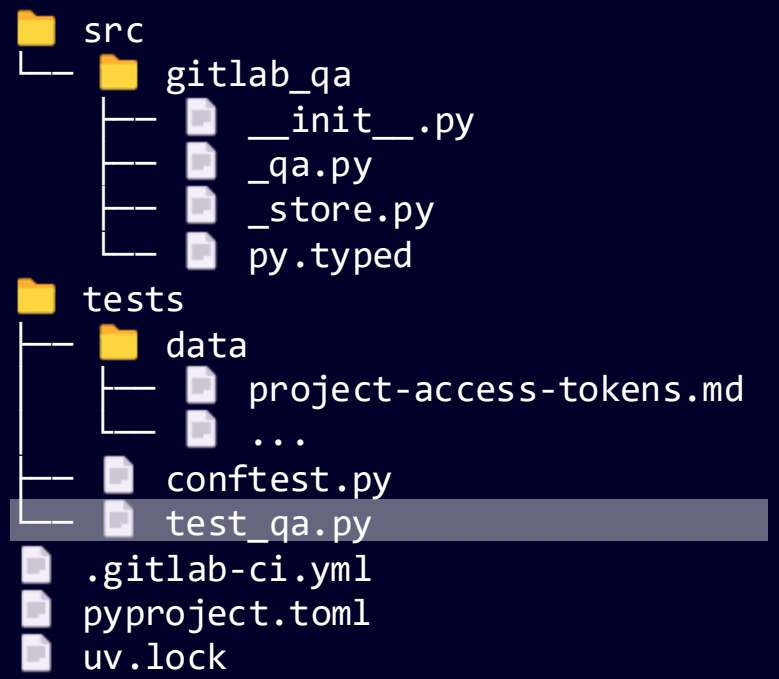
GitLab Q&A demo project




```python
...


@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    return OllamaModel(
        model=_EVALUATOR_MODEL,
        base_url=request.config.getoption("--ollama-url"),
        seed=...,
        temperature=...,
    )

...
```


SIEMENS

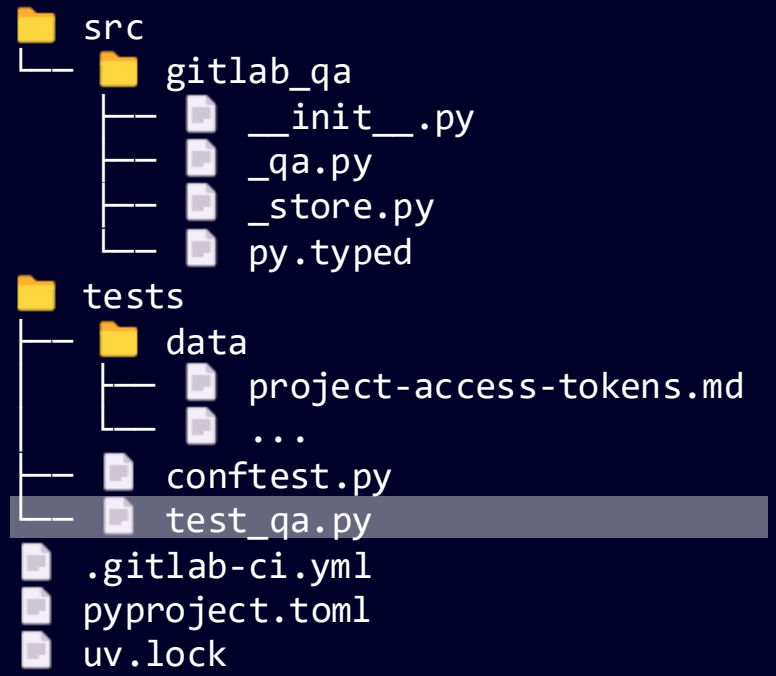
GitLab Q&A demo project




```python
...


@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    return OllamaModel(
        model=_EVALUATOR_MODEL,
        base_url=request.config.getoption("--ollama-url"),
        seed=...,
        temperature=...,
    )

...
```


SIEMENS

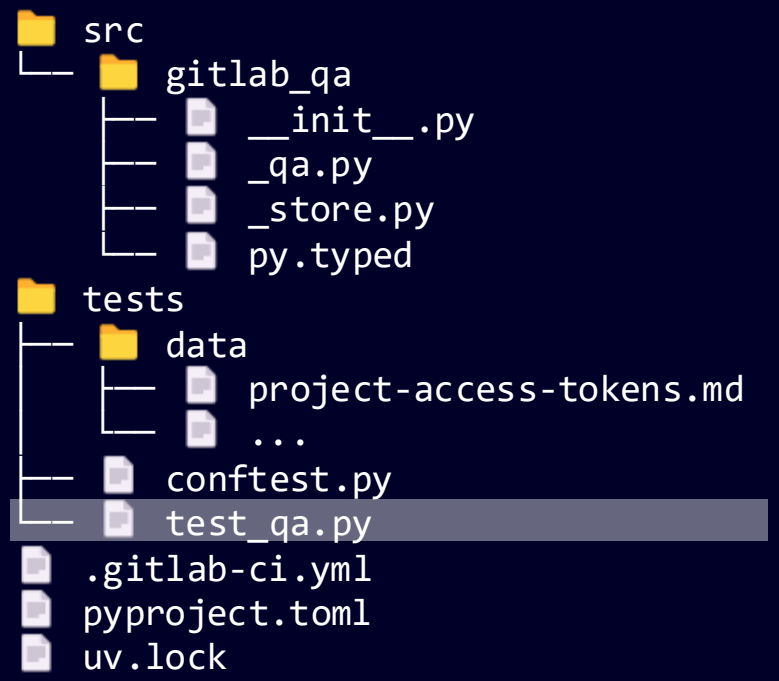
GitLab Q&A demo project




```python
...


@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    return OllamaModel(
        model=_EVALUATOR_MODEL,
        base_url=request.config.getoption("--ollama-url"),
        seed=0,
        temperature=...,
    )

...
```


SIEMENS

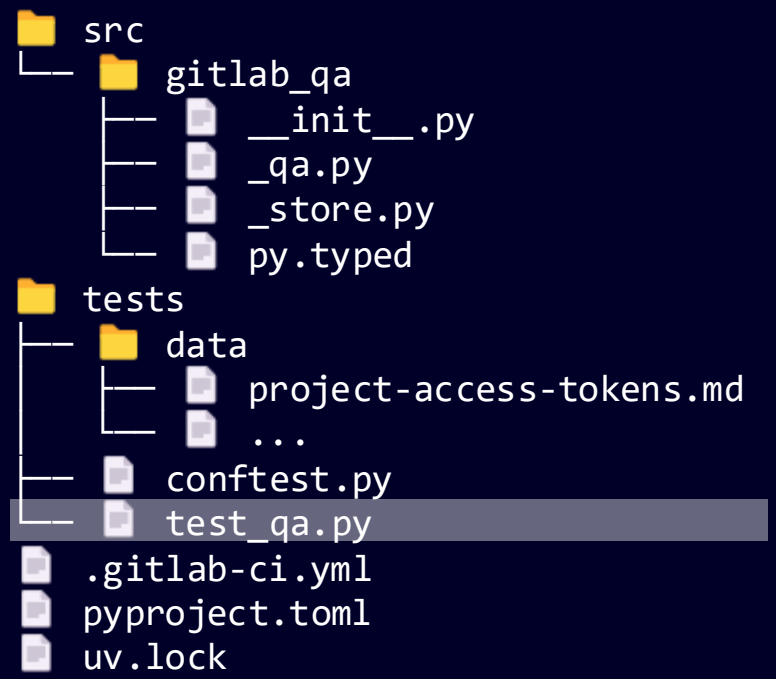
GitLab Q&A demo project




```python
...


@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    return OllamaModel(
        model=_EVALUATOR_MODEL,
        base_url=request.config.getoption("--ollama-url"),
        seed=0,
        temperature=0.0,
    )

...
```


SIEMENS

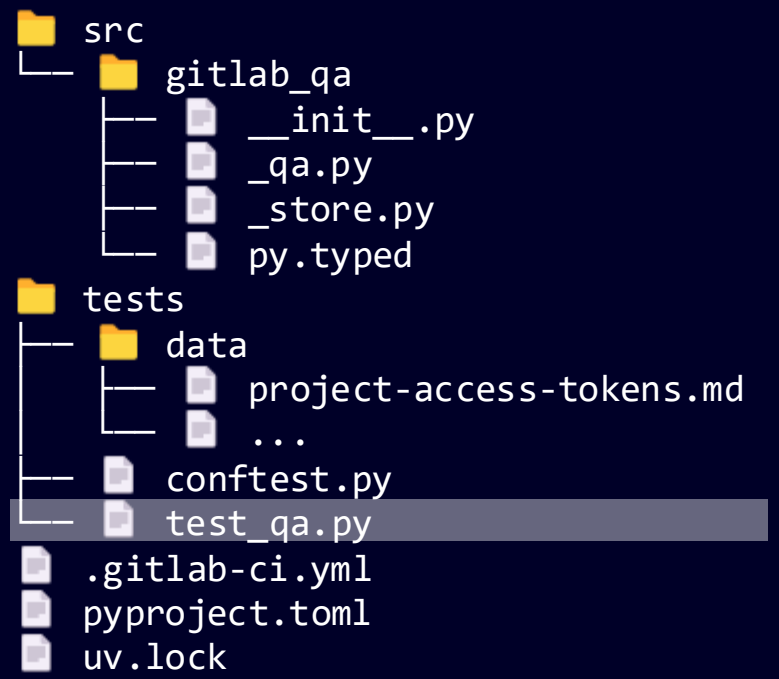
GitLab Q&A demo project




```python
...


@pytest.fixture(scope="session")
def evaluator(request: pytest.FixtureRequest) -> OllamaModel:
    return OllamaModel(
        model=_EVALUATOR_MODEL,
        base_url=request.config.getoption("--ollama-url"),
        seed=0,
        temperature=0.0,
    )

...
```


SIEMENS

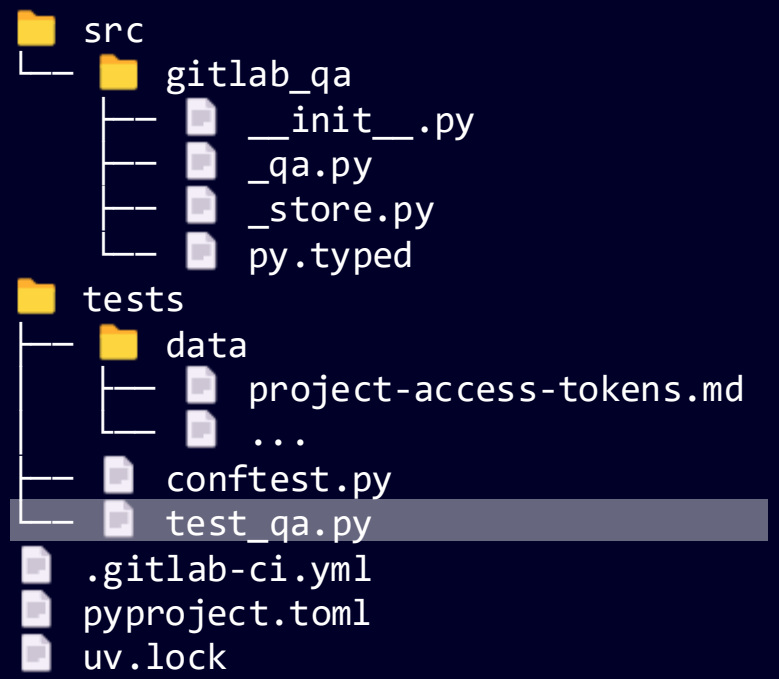
GitLab Q&A demo project




```
...

@pytest.mark.parametrize(
    ("question", "reference"),
    [...],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    ...


...
```


SIEMENS

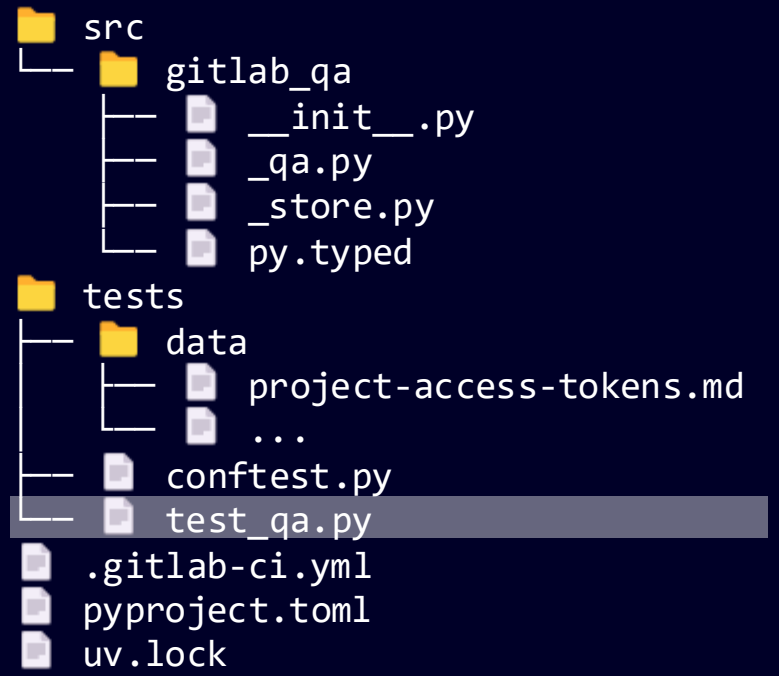
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [...],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    ...


...
```


SIEMENS

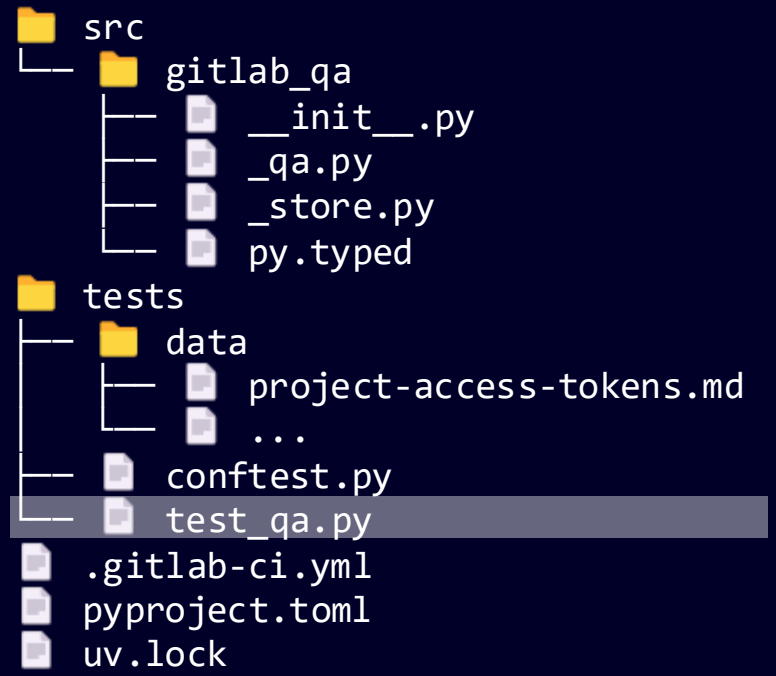
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            ...,
            ...,
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    ...


...
```


SIEMENS

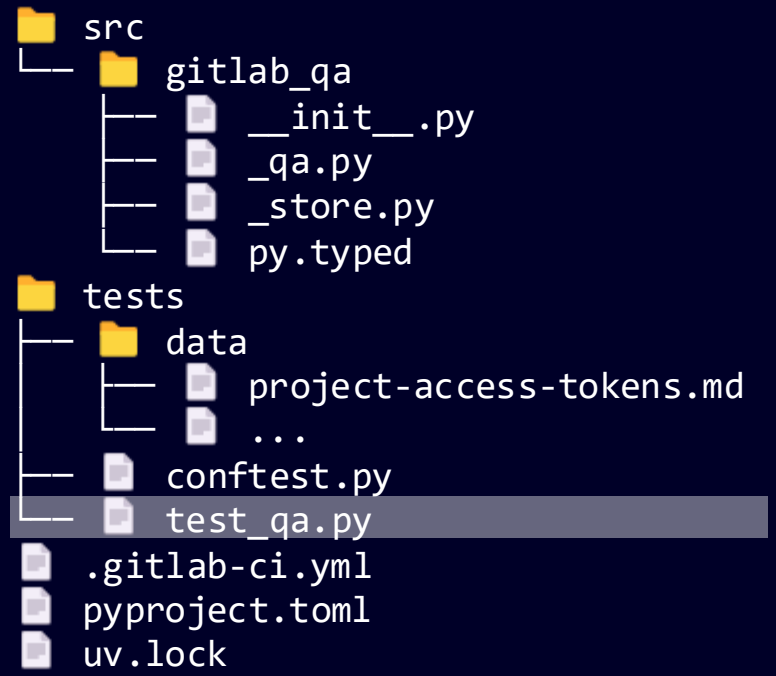
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            ...,
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    ...


...
```


SIEMENS

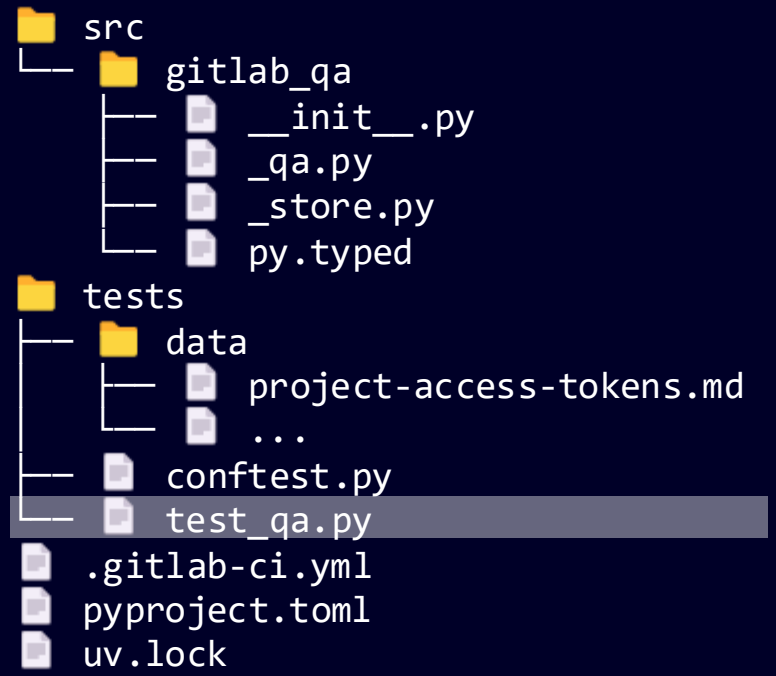
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            ...,
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    ...


...
```


SIEMENS

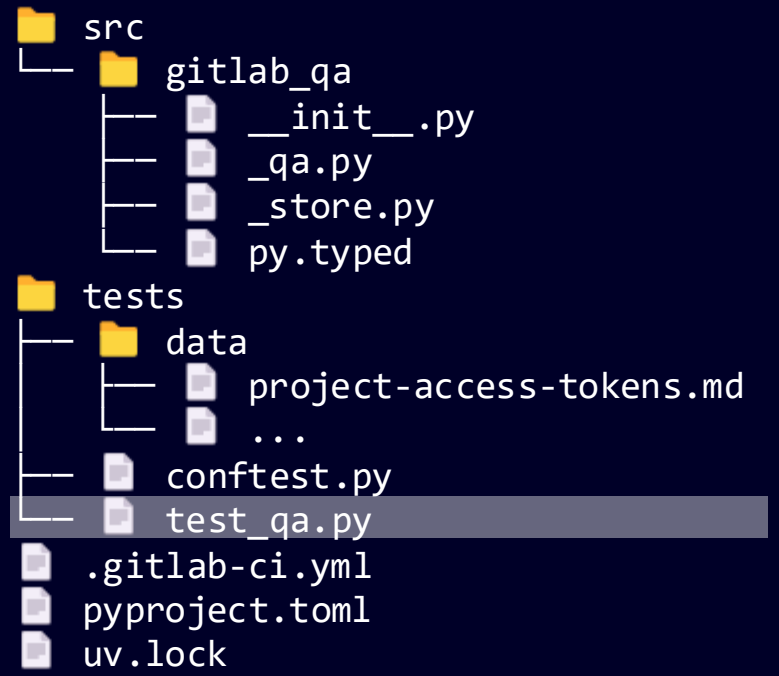
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            "Rotate a project access token in your project as follows:\n\n"
            "1. Go to `Settings > Access tokens`\n"
            '2. For the relevant token, click the "Rotate" button\n'
            '3. On the confirmation dialog, select "Rotate"',
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    ...


...
```


SIEMENS

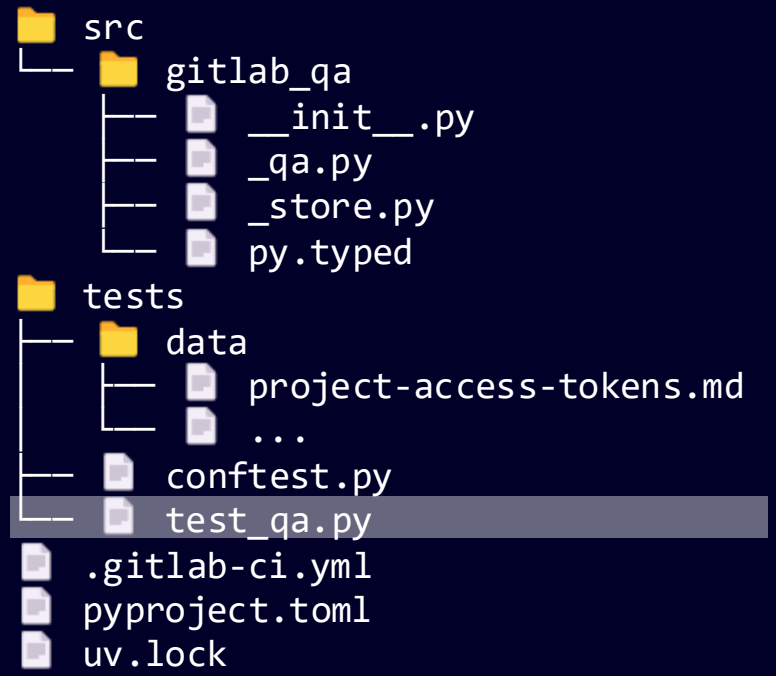
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            "Rotate a project access token in your project as follows:\n\n"
            "1. Go to `Settings > Access tokens`\n"
            '2. For the relevant token, click the "Rotate" button\n'
            '3. On the confirmation dialog, select "Rotate"',
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    ...


...
```


SIEMENS

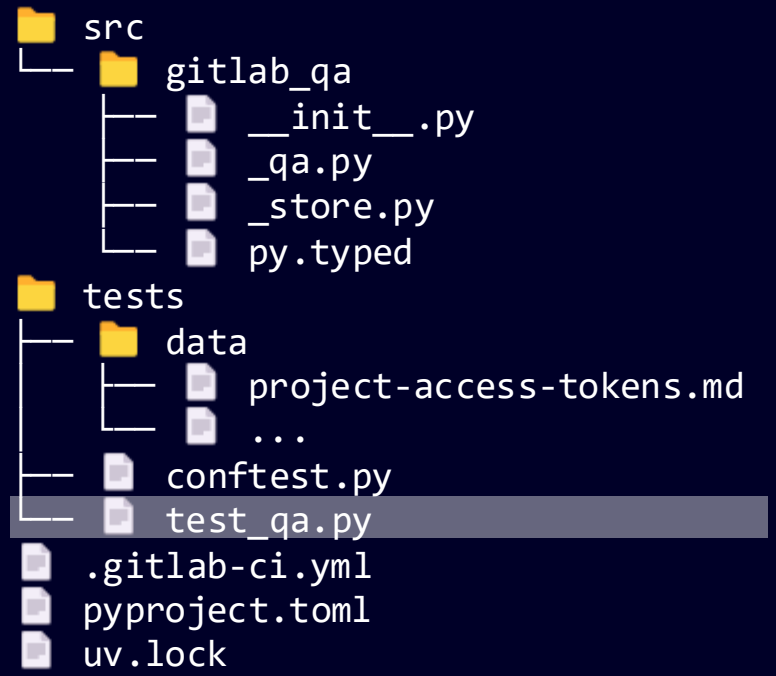
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            "Rotate a project access token in your project as follows:\n\n"
            "1. Go to `Settings > Access tokens`\n"
            '2. For the relevant token, click the "Rotate" button\n'
            '3. On the confirmation dialog, select "Rotate"',
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    answer, context = gitlab_qa.ask(question)
    ...


...
```


SIEMENS

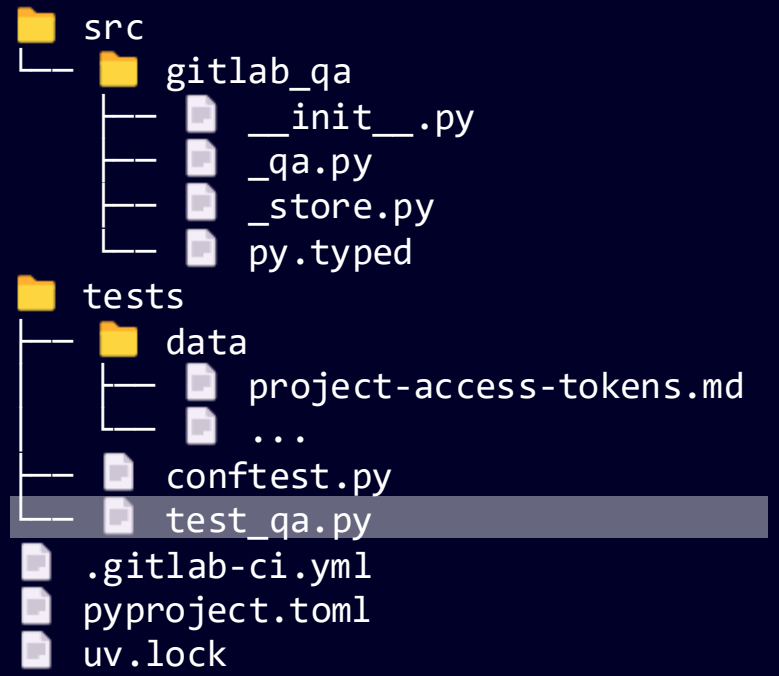
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            "Rotate a project access token in your project as follows:\n\n"
            "1. Go to `Settings > Access tokens`\n"
            '2. For the relevant token, click the "Rotate" button\n'
            '3. On the confirmation dialog, select "Rotate"',
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    answer, context = gitlab_qa.ask(question)
    ...


...
```


SIEMENS

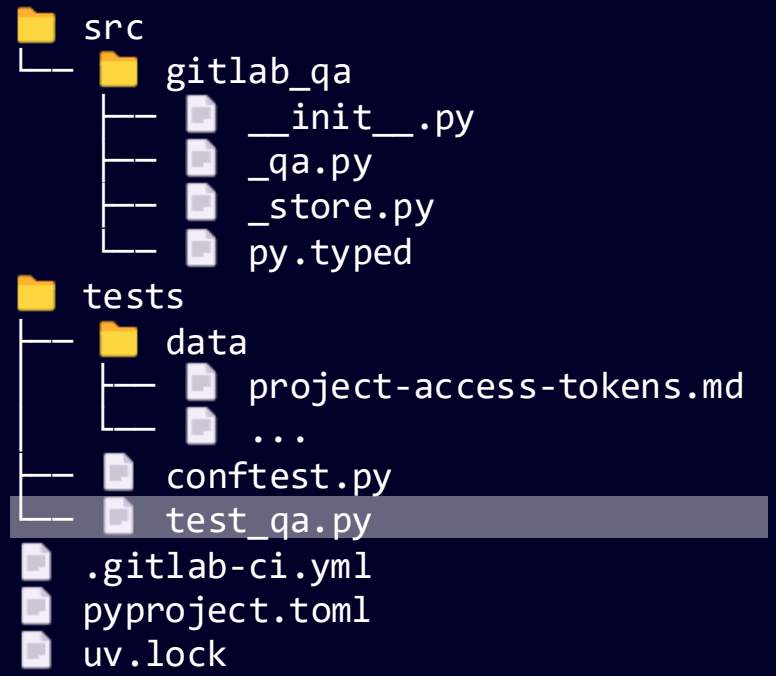
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            "Rotate a project access token in your project as follows:\n\n"
            "1. Go to `Settings > Access tokens`\n"
            '2. For the relevant token, click the "Rotate" button\n'
            '3. On the confirmation dialog, select "Rotate"',
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    answer, context = gitlab_qa.ask(question)
    assert_test(
        ...,
        metrics=[...],
    )


...
```


SIEMENS

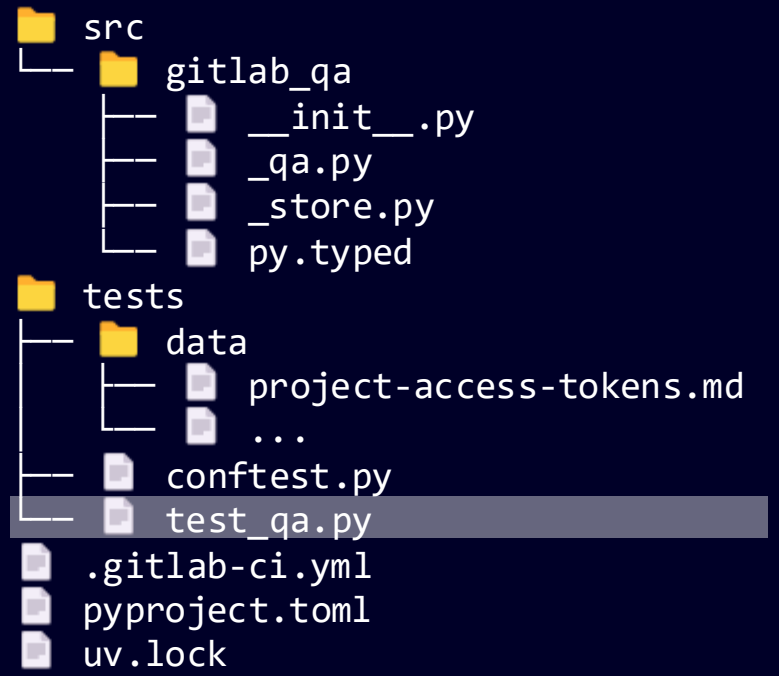
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            "Rotate a project access token in your project as follows:\n\n"
            "1. Go to `Settings > Access tokens`\n"
            '2. For the relevant token, click the "Rotate" button\n'
            '3. On the confirmation dialog, select "Rotate"',
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    answer, context = gitlab_qa.ask(question)
    assert_test(
        ...,
        metrics=[...],
    )


...
```


SIEMENS

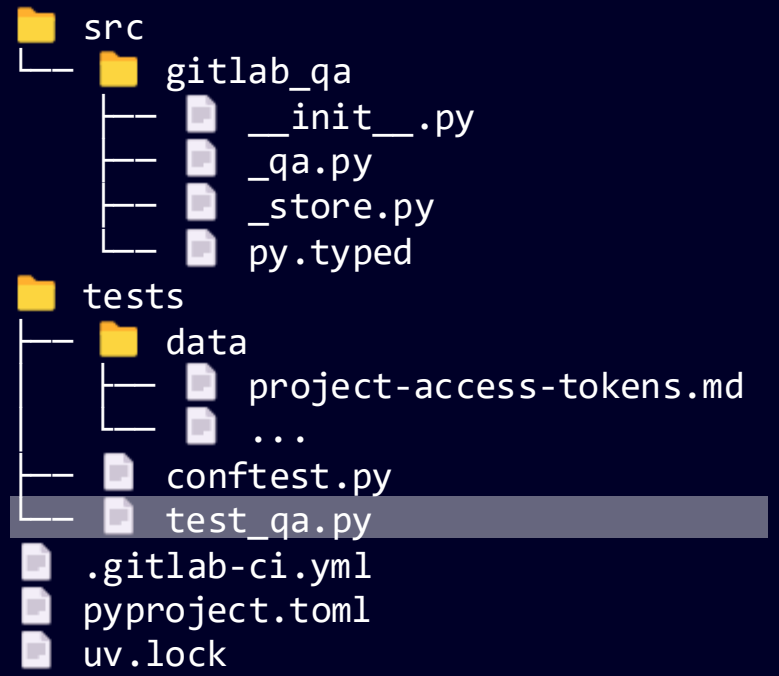
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            "Rotate a project access token in your project as follows:\n\n"
            "1. Go to `Settings > Access tokens`\n"
            '2. For the relevant token, click the "Rotate" button\n'
            '3. On the confirmation dialog, select "Rotate"',
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    answer, context = gitlab_qa.ask(question)
    assert_test(
        LLMTestCase(
            input=question,
            actual_output=answer,
            expected_output=reference,
            retrieval_context=list(context),
        ),
        metrics=[...],
    )


...
```


SIEMENS

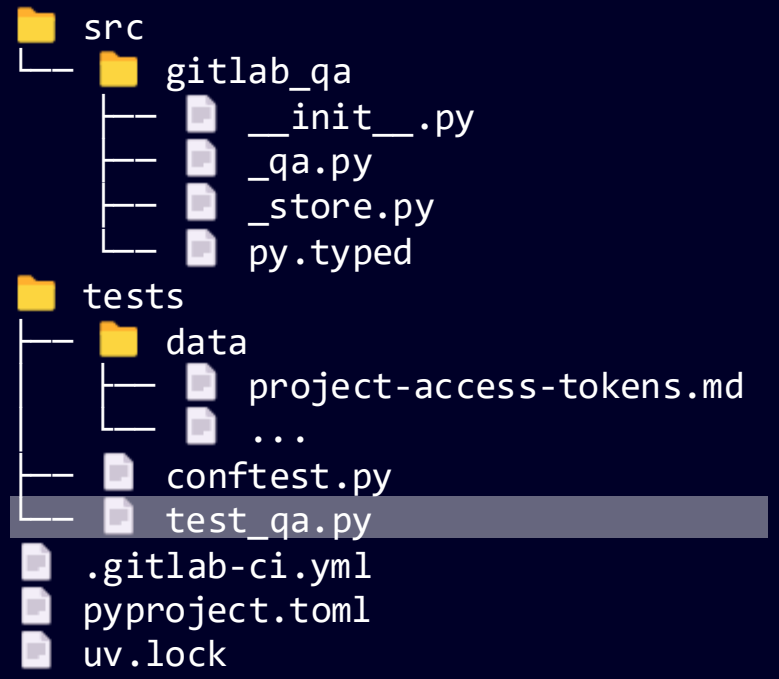
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            "Rotate a project access token in your project as follows:\n\n"
            "1. Go to `Settings > Access tokens`\n"
            '2. For the relevant token, click the "Rotate" button\n'
            '3. On the confirmation dialog, select "Rotate"',
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    answer, context = gitlab_qa.ask(question)
    assert_test(
        LLMTestCase(
            input=question,
            actual_output=answer,
            expected_output=reference,
            retrieval_context=list(context),
        ),
        metrics=[...],
    )


...
```


SIEMENS

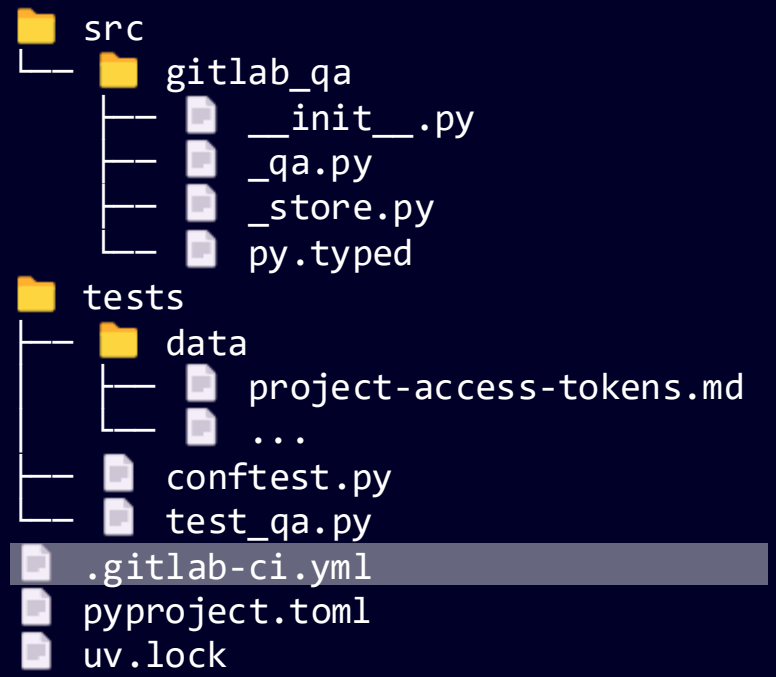
GitLab Q&A demo project




```python
...


@pytest.mark.parametrize(
    ("question", "reference"),
    [
        (
            "How can I manually rotate a project access token via UI?",
            "Rotate a project access token in your project as follows:\n\n"
            "1. Go to `Settings > Access tokens`\n"
            '2. For the relevant token, click the "Rotate" button\n'
            '3. On the confirmation dialog, select "Rotate"',
        )
    ],
)
def test_gitlab_qa(
    gitlab_qa: GitlabQA, evaluator: OllamaModel, question: str, reference: str
) -> None:
    answer, context = gitlab_qa.ask(question)
    assert_test(
        LLMTestCase(
            input=question,
            actual_output=answer,
            expected_output=reference,
            retrieval_context=list(context),
        ),
        metrics=[
            AnswerRelevancyMetric(model=evaluator),
            FaithfulnessMetric(model=evaluator),
            ContextualPrecisionMetric(model=evaluator),
            ContextualRecallMetric(model=evaluator),
            ContextualRelevancyMetric(model=evaluator),
        ],
    )
```


SIEMENS

GitLab Q&A demo project




```yaml
Integration tests:
  stage: test

  # Service containers
  services:
    - name: ollama/ollama:0.7.1
      alias: ollama
    - name: pgvector/pgvector:0.8.0-pg17
      alias: pgvector
      variables:
        POSTGRES_DB: test
        POSTGRES_USER: test
        POSTGRES_PASSWORD: test

  # Main job
  image: ghcr.io/astral-sh/uv:0.7.7-python3.13-bookworm
  before_script:
    - uv sync --frozen
  script:
    - uv run pytest
      --ollama-url http://ollama:11434
      --pgvector-uri postgresql+psycopg://test:test@pgvector:5432/test
```


SIEMENS

Copier for LLM Engineering on GitLab

Scaffolding and lifecycle management for the generative AI era

Copier

https://github.com/copier-org/copier

SIEMENS

If you like Copier …

… please consider giving a shiny GitHub star

Copier

https://github.com/copier-org/copier

SIEMENS

Questions?