```
import pytest

from swarms.structs.conversation import Conversation


@pytest.fixture
def conversation():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    conv.add("assistant", "Hello, user!")
    return conv


def test_add_message():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    assert len(conv.conversation_history) == 1
    assert conv.conversation_history[0]["role"] == "user"
    assert conv.conversation_history[0]["content"] == "Hello, world!"


def test_add_message_with_time():
    conv = Conversation(time_enabled=True)
    conv.add("user", "Hello, world!")
    assert len(conv.conversation_history) == 1
    assert conv.conversation_history[0]["role"] == "user"
```

```
    assert conv.conversation_history[0]["content"] == "Hello, world!"
    assert "timestamp" in conv.conversation_history[0]


def test_delete_message():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    conv.delete(0)
    assert len(conv.conversation_history) == 0


def test_delete_message_out_of_bounds():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    with pytest.raises(IndexError):
        conv.delete(1)


def test_update_message():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    conv.update(0, "assistant", "Hello, user!")
    assert len(conv.conversation_history) == 1
    assert conv.conversation_history[0]["role"] == "assistant"
    assert conv.conversation_history[0]["content"] == "Hello, user!"
```

```
def test_update_message_out_of_bounds():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    with pytest.raises(IndexError):
        conv.update(1, "assistant", "Hello, user!")


def test_return_history_as_string_with_messages(conversation):
    result = conversation.return_history_as_string()
    assert result is not None


def test_return_history_as_string_with_no_messages():
    conv = Conversation()
    result = conv.return_history_as_string()
    assert result == ""


@pytest.mark.parametrize(
    "role, content",
    [
        ("user", "Hello, world!"),
        ("assistant", "Hello, user!"),
        ("system", "System message"),
        ("function", "Function message"),
```

```
    ],
)
def test_return_history_as_string_with_different_roles(role, content):
    conv = Conversation()
    conv.add(role, content)
    result = conv.return_history_as_string()
    expected = f"{role}: {content}\n\n"
    assert result == expected


@pytest.mark.parametrize("message_count", range(1, 11))
def test_return_history_as_string_with_multiple_messages(
    message_count,
):
    conv = Conversation()
    for i in range(message_count):
        conv.add("user", f"Message {i + 1}")
    result = conv.return_history_as_string()
    expected = "".join(
        [f"user: Message {i + 1}\n\n" for i in range(message_count)]
    )
    assert result == expected


@pytest.mark.parametrize(
    "content",
```

```
    [
        "Hello, world!",
        "This is a longer message with multiple words.",
        "This message\nhas multiple\nlines.",
        "This message has special characters: !@#$%^&*()",
        "This message has unicode characters: ",
    ],
)
def test_return_history_as_string_with_different_contents(content):
    conv = Conversation()
    conv.add("user", content)
    result = conv.return_history_as_string()
    expected = f"user: {content}\n\n"
    assert result == expected



def test_return_history_as_string_with_large_message(conversation):
    large_message = "Hello, world! " * 10000  # 10,000 repetitions
    conversation.add("user", large_message)
    result = conversation.return_history_as_string()
    expected = (
        "user: Hello, world!\n\nassistant: Hello, user!\n\nuser:"
        f" {large_message}\n\n"
    )
    assert result == expected
```

```
def test_search_keyword_in_conversation(conversation):
    result = conversation.search_keyword_in_conversation("Hello")
    assert len(result) == 2
    assert result[0]["content"] == "Hello, world!"
    assert result[1]["content"] == "Hello, user!"


def test_export_import_conversation(conversation, tmp_path):
    filename = tmp_path / "conversation.txt"
    conversation.export_conversation(filename)
    new_conversation = Conversation()
    new_conversation.import_conversation(filename)
    assert (
        new_conversation.return_history_as_string()
        == conversation.return_history_as_string()
    )


def test_count_messages_by_role(conversation):
    counts = conversation.count_messages_by_role()
    assert counts["user"] == 1
    assert counts["assistant"] == 1


def test_display_conversation(capsys, conversation):
```

```
    conversation.display_conversation()
    captured = capsys.readouterr()
    assert "user: Hello, world!\n\n" in captured.out
    assert "assistant: Hello, user!\n\n" in captured.out


def test_display_conversation_detailed(capsys, conversation):
    conversation.display_conversation(detailed=True)
    captured = capsys.readouterr()
    assert "user: Hello, world!\n\n" in captured.out
    assert "assistant: Hello, user!\n\n" in captured.out


def test_search():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    conv.add("assistant", "Hello, user!")
    results = conv.search("Hello")
    assert len(results) == 2
    assert results[0]["content"] == "Hello, world!"
    assert results[1]["content"] == "Hello, user!"


def test_return_history_as_string():
    conv = Conversation()
    conv.add("user", "Hello, world!")
```

```
    conv.add("assistant", "Hello, user!")
    result = conv.return_history_as_string()
    expected = "user: Hello, world!\n\nassistant: Hello, user!\n\n"
    assert result == expected


def test_search_no_results():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    conv.add("assistant", "Hello, user!")
    results = conv.search("Goodbye")
    assert len(results) == 0


def test_search_case_insensitive():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    conv.add("assistant", "Hello, user!")
    results = conv.search("hello")
    assert len(results) == 2
    assert results[0]["content"] == "Hello, world!"
    assert results[1]["content"] == "Hello, user!"


def test_search_multiple_occurrences():
    conv = Conversation()
```

```
    conv.add("user", "Hello, world! Hello, world!")
    conv.add("assistant", "Hello, user!")
    results = conv.search("Hello")
    assert len(results) == 2
    assert results[0]["content"] == "Hello, world! Hello, world!"
    assert results[1]["content"] == "Hello, user!"


def test_query_no_results():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    conv.add("assistant", "Hello, user!")
    results = conv.query("Goodbye")
    assert len(results) == 0


def test_query_case_insensitive():
    conv = Conversation()
    conv.add("user", "Hello, world!")
    conv.add("assistant", "Hello, user!")
    results = conv.query("hello")
    assert len(results) == 2
    assert results[0]["content"] == "Hello, world!"
    assert results[1]["content"] == "Hello, user!"
```

```
def test_query_multiple_occurrences():
    conv = Conversation()
    conv.add("user", "Hello, world! Hello, world!")
    conv.add("assistant", "Hello, user!")
    results = conv.query("Hello")
    assert len(results) == 2
    assert results[0]["content"] == "Hello, world! Hello, world!"
    assert results[1]["content"] == "Hello, user!"
```