

## **Zusammenfassung** *(automatisiert erstellt)*

Sie möchten, dass bei jedem Containerlauf eine eindeutige Ergebnisdatei wie ``result_<inputname>.txt`` erzeugt wird, um Überschreibungen zu vermeiden. Dafür ist im Python-Skript im Container (z.B. ``test.py``) erforderlich, den Ausgabedateinamen als Argument zu akzeptieren. Im ``main()``-Bereich muss für jede Eingabedatei ein individueller Name generiert und beim Start des Containers übergeben werden, z.B. via ``launch_container_rest`` mit ``container_args=[input_filename, output_filename]``. Das Container-Skript liest diese Argumente (``sys.argv[1]``, ``sys.argv[2]``) und schreibt die Resultate in die jeweilige Datei. So wird die Nachvollziehbarkeit und Reproduzierbarkeit des Workflows gesichert.

## **Systemprompt**

Du bist ein intelligentes und unterstützendes KI-Assistenzsystem für alle Hochschulangehörigen der HAWK Hildesheim/Holzminden/Göttingen. Dein Ziel ist es, Studierende, Lehrende, Forschende und Mitarbeitende in ihrer akademischen Arbeit, beim Lernen, Forschen, Lehren und verwalterischen Aufgaben zu unterstützen. Dabei förderst du kollaboratives Arbeiten, wissenschaftliches Denken und eine kreative Problemlösung. Beziehe dich auf wissenschaftliche Methoden und Theorien, argumentiere sachlich und reflektiere kritisch. Sei objektiv und verzichte auf unbegründete Meinungen. Fördere akademische Integrität und unterstütze keine Plagiate. Sei inklusiv, wertschätzend und respektiere Vielfalt.

## Verlauf:

### Bousfiha, Tanae:

```
while True:
    choice = input("Welche Datei soll verwendet
werden? Gib die Nummer ein: ")
    if choice.isdigit() and 1 <= int(choice) <=
len(all_files):#Sicherstellt die Eingabe innerhalb der
möglichen Dateiindizes liegt.
        selected = all_files[int(choice) - 1]#Die Liste
all_files ist nullbasiert (erster Eintrag ist an Index 0),
daher wird 1 abgezogen.
        print(f"select one or more Datei:
{selected['name']}")

    return selected
else:
    print("Ungültige Auswahl.")
```

what i want him to make it possible too the user to choose more than 1 file

### HAWKI (gpt-4.1):

Um den Benutzer zu ermöglichen, mehr als eine Datei auszuwählen, musst du die Eingabeverarbeitung anpassen, sodass der Nutzer mehrere Dateinummern (zum Beispiel durch Komma getrennt) eingeben kann. Danach validierst du jede Eingabe, sammelst die ausgewählten Dateien in einer Liste und gibst sie dann zurück.

Hier ist eine überarbeitete Variante deines Codes auf Python, die mehrere Auswahlen ermöglicht:

```
python
while True:
    choice = input("Welche Datei(en) sollen verwendet werden? Gib die Nummern ein
(z.B. 1,3,5): ")
    # Teile die Eingabe am Komma und entferne überflüssige Leerzeichen
    indices = [c.strip() for c in choice.split(",")]
```

```

# Prüfe, ob alle Eingaben gültige Dateinummern sind
if all(i.isdigit() and 1 <= int(i) <= len(all_files) for i in indices):
    # Entferne Duplikate & wandle in int um
    unique_indices = list(set(int(i) for i in indices))
    selected = [all_files[i - 1] for i in unique_indices]
    print("Ausgewählte Dateien:")
    for s in selected:
        print(f"- {s['name']}")
    return selected
else:
    print("Ungültige Auswahl. Bitte gib gültige Nummern an.")
...

```

**\*\*Erläuterung:\*\***

- Mehrere Nummern sind durch Kommas getrennt möglich.
- Leerzeichen werden entfernt und jeder Index wird überprüft.
- Doppelte Einträge werden vermieden (`set()`).
- Es werden alle ausgewählten Dateien übersichtlich ausgegeben.

**\*\*Hinweis:\*\***

Falls du unterschiedliche Auswahlformate erlauben willst (z.B. „1 3 5“ oder „1,2,3“), kannst du mit regulären Ausdrücken oder `re.split(r'\D+', choice)` arbeiten. Sag gern Bescheid, wenn du das möchtest!

Melde dich, falls du noch Anpassungen vornehmen oder eine noch komfortablere Auswahl möchtest (z.B. per Bereichseingabe wie `1-3,5`).

**Bousfiha, Tanae:**

und auf main

**HAWKI (gpt-4.1):**

Natürlich! Wenn du in deiner `main()`-Funktion die Auswahl mehrerer Dateien benutzen möchtest, kannst du die überarbeitete Auswahlfunktion (z. B. `select\_files`) einsetzen und ihr das `all\_files`-Array übergeben. Beispielweise so:

### 1. Erstelle die Auswahlfunktion:

Schreibe die Funktion ausgelagert, damit du sie einfach aus `main()` aufrufen kannst:

```
```python
```

```
def select_files(all_files):
```

```
    while True:
```

```
        choice = input("Welche Datei(en) sollen verwendet werden? Gib die Nummern  
ein (z.B. 1,3,5): ")
```

```
        indices = [c.strip() for c in choice.split(',')]
```

```
        if all(i.isdigit() and 1 <= int(i) <= len(all_files) for i in indices):
```

```
            unique_indices = list(set(int(i) for i in indices))
```

```
            selected = [all_files[i - 1] for i in unique_indices]
```

```
            print("Ausgewählte Dateien:")
```

```
            for s in selected:
```

```
                print(f"- {s['name']}")
```

```
            return selected
```

```
        else:
```

```
            print("Ungültige Auswahl. Bitte gib gültige Nummern an.")
```

```
    ...
```

```
---
```

### 2. Integration in die `main()`-Funktion

```
```python
```

```
def main():
```

```
    # Beispiel-Daten:
```

```
    all_files = [
```

```
        {'name': 'datei1.txt'},
```

```
        {'name': 'datei2.txt'},
```

```
        {'name': 'datei3.txt'},
```

```
        # ... weitere Dateien
```

```
    ]
```

```
    # Zeige dem Nutzer die verfügbaren Dateien an
```

```
    print("Verfügbare Dateien:")
```

```
    for idx, file in enumerate(all_files, start=1):
```

```
        print(f"{idx}. {file['name']}")
```

```
# Nutzer wählt eine oder mehrere Dateien aus
selected_files = select_files(all_files)
```

```
# Weiterverarbeitung der ausgewählten Dateien
```

```
# Beispiel: Datei-Namen ausgeben
```

```
print("Weiterverarbeitung:")
```

```
for f in selected_files:
```

```
    print(f"- {f['name']}")
```

```
...
```

```
---
```

```
### 3. Starte das Programm
```

```
```python
```

```
if __name__ == "__main__":
```

```
    main()
```

```
```
```

```
---
```

```
**Wissenschaftlicher Hintergrund:**
```

Diese Vorgehensweise folgt den Prinzipien der Modularisierung und

Benutzerinteraktion nach gängigen Software-Engineering-Methoden: Eingabelisten

werden validiert, Feedback wird gegeben und eine fehlerrobuste Eingabe ist

gewährleistet.

```
---
```

```
** Hinweis: ** Sag Bescheid, falls du die Funktionalität um z.B.
andersartige Darstellung für viele Dateien oder weitere Optionen erweitern möchtest!
```

**Bousfiha, Tanae:**

das ist mein emain part

```
def main():
```

```
    xn Timer host = &quot;https://xn Timer dev.gwdg.de&quot;
```

```
    docker_base_image = &quot;python:3.10&quot;
```

```
xnat_user = get_input("&quot;XNAT Username: &quot;")
xnat_password = getpass.getpass("&quot;XNAT Password: &quot;")
project_id = get_input("&quot;Project ID: &quot;").strip()
script_path = get_input("&quot;Path to the Python script: &quot;")
```

# 1. Hole die Basisdaten/Eingaben

```
mod_data = modification()
```

# 2. Ergänze die nötigen Felder für downstream

```
mod_data["contexts"] = [mod_data["selected_context"]]
mod_data["label_name"] = mod_data["command_name"]
mod_data["label_description"] =
mod_data["command_description"]
wrapper_name = mod_data["command_name"].replace("&quot; &quot;",
"&quot;_&quot;").lower() + "&quot;_wrapper"&quot;;#Generiert technische Bezeichnung
fuer einen xnat wrapper
```

# 3. Jetzt kann überall darauf zugegriffen werden

```
dockerfile_path = write_dockerfile("&quot;.&quot;", os.path.basename(script_path),
docker_base_image)
local_image_name =
f"&quot;{mod_data['command_name'].lower().replace(' ',
'_'&quot;);latest&quot;";
full_image_name = build_and_push_docker_image(dockerfile_path,
local_image_name)
```

```
json_file_path = create_json_file(full_image_name, os.path.basename(script_path),
mod_data)
```

```
send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)
```

```
command_id = get_command_id_by_name(xnat_host, xnat_user, xnat_password,
mod_data["command_name"])
```

try:

```
wrapper_id = get_wrapper_id_by_command_name(
    xnat_host, xnat_user, xnat_password,
    mod_data["command_name"], wrapper_name
```

```

    )
    print(f'Wrapper already exists: {wrapper_id}')
except SystemExit:
    print('Wrapper not found')
    outputs, external_inputs, derived_inputs = get_command_id(
        xnat_host, xnat_user, xnat_password, command_id
    )

    enable_wrapper_sitewide(xnat_host, command_id, wrapper_name, xnat_user,
                             xnat_password)
    enable_wrapper_for_project(xnat_host, project_id, command_id, wrapper_name,
                               xnat_user, xnat_password)

CONTEXT_ENTITY_MAPPING = {
    'xnat:projectData': {'entity_type': 'project',
                        'id_label': 'Project ID'},
    'xnat:subjectData': {'entity_type': 'subject',
                        'id_label': 'Subject ID'},
    'xnat:mrSessionData': {'entity_type':
                        'session', 'id_label': 'Session ID'},
    'xnat:petSessionData': {'entity_type':
                        'session', 'id_label': 'Session ID'},
    'xnat:ctSessionData': {'entity_type':
                        'session', 'id_label': 'Session ID'},
    'xnat:sessionData': {'entity_type': 'session',
                        'id_label': 'Session ID'},
    'xnat:imageScanData': {'entity_type': 'scan',
                        'id_label': 'Session ID + Scan ID'},
    'xnat:resource': {'entity_type': 'resource',
                        'id_label': 'Resource (not supported)'}
}

first_context = mod_data['contexts'][0]
context_info = CONTEXT_ENTITY_MAPPING.get(first_context)

if not context_info:
    print(f'Unbekannter Kontext: {first_context}. Abbruch.')
    return

```

```

entity_type = context_info["entity_type"]

if entity_type == "scan":
    session_id = get_input("Gib die Session-ID ein: ")
    scan_id = get_input("Gib die Scan-ID ein: ")
    input_file_info = get_input_file(
        xnat_host, session_id, entity_type, xnat_user, xnat_password,
scan_id=scan_id
    )
    container_entity_id = session_id

elif entity_type in ["session", "subject", "project"]:
    entity_id = get_input(f"Gib die {context_info['id_label']} ein: ")
    input_file_info = get_input_file(
        xnat_host, entity_id, entity_type, xnat_user, xnat_password
    )
    container_entity_id = entity_id
    scan_id = None # wird nicht gebraucht

else:
    print("Dieser Kontext wird aktuell nicht unterstützt.")
    return

if input_file_info:
    launch_container_rest(
        xnat_host,
        project_id,
        command_id,
        wrapper_name,
        container_entity_id,
        xnat_user,
        xnat_password,
        input_file_info,
        entity_type=entity_type,
        scan_id=scan_id if entity_type == "scan" else None
    )

```



else:

```
print("&quot;Keine Datei ausgewählt. Containerstart abgebrochen.&quot;)
```

```
if __name__ == "&quot;__main__&quot;:
```

```
    main()
```

### **HAWKI (gpt-4.1):**

Dein Ziel ist es, im `main()`-Teil eine Mehrfachauswahl von Dateien zu ermöglichen und darauf basierend dein weiteres Vorgehen (etwa mit der Funktion `launch\_container\_rest`) zu steuern.

#### ### Vorgehen

1. **Hole** die Liste aller auswählbaren Dateien (z.B. `all\_files`).
2. **Biete die Auswahl** mehrerer Dateien mit einer Funktion wie `select\_files(all\_files)` an.
3. **Übergebe die Auswahlliste** an den nächsten Verarbeitungsschritt (in deinem Fall vermutlich an `launch\_container\_rest` & Co.).

---

#### ### Beispielhafte Integration

Nehmen wir an, deine Funktion `get\_input\_file(...)` liefert eine Liste von Dateien (z.B. in Form von Dictionaries mit Dateiinformationen):

Du kannst dann ein Auswahlmenü schreiben (ähnlich wie oben), z.B. so:

```
```python
def select_files(all_files):
    if not all_files:
        print("Keine verfügbaren Dateien gefunden.")
        return []
    print("Verfügbare Dateien:")
    for idx, f in enumerate(all_files, start=1):
        print(f"{idx}: {f['name']}")
```

```

while True:
    choice = input("Nummern der Dateien auswählen (z.B. 1,3,5): ")
    indices = [c.strip() for c in choice.split(',')]
    if all(i.isdigit() and 1 <= int(i) <= len(all_files) for i in indices):
        unique_indices = list(set(int(i) for i in indices))
        selected_files = [all_files[i-1] for i in unique_indices]
        print("Ausgewählte Dateien:")
        for s in selected_files:
            print(f"- {s['name']}")
        return selected_files
    else:
        print("Ungültige Auswahl. Bitte wiederholen.")

```

# Beispielhafte Verwendung im main()

```

if entity_type == "scan":
    session_id = get_input("Gib die Session-ID ein: ")
    scan_id = get_input("Gib die Scan-ID ein: ")
    all_files = get_input_file(
        xnat_host, session_id, entity_type, xnat_user, xnat_password, scan_id=scan_id
    )
    container_entity_id = session_id

elif entity_type in ["session", "subject", "project"]:
    entity_id = get_input(f"Gib die {context_info['id_label']} ein: ")
    all_files = get_input_file(
        xnat_host, entity_id, entity_type, xnat_user, xnat_password
    )
    container_entity_id = entity_id
    scan_id = None

```

```

else:
    print("Dieser Kontext wird aktuell nicht unterstützt.")
    return

```

# Hier kommt die Modifikation!

```

selected_files = select_files(all_files)

```

```

if selected_files:

```

```

# Hier setzt du ggf. deine Logik für jede ausgewählte Datei an:
for input_file_info in selected_files:
    launch_container_rest(
        xnat_host,
        project_id,
        command_id,
        wrapper_name,
        container_entity_id,
        xnat_user,
        xnat_password,
        input_file_info,
        entity_type=entity_type,
        scan_id=scan_id if entity_type == "scan" else None
    )
else:
    print("Keine Datei ausgewählt. Containerstart abgebrochen.")
...

---

### **Hinweise zur Umsetzung und zu wissenschaftlichen Prinzipien**

- **Modularisierung:** Die Auswahlfunktion ist vom restlichen Code getrennt und gut testbar.
- **Validierung:** Die Eingaben werden abgesichert und auf Plausibilität überprüft.
- **Benutzerzentrierung:** Mehrfachauswahl ist benutzungsfreundlich — effizient bei wiederkehrenden Aufgaben (vgl. Human-Computer Interaction Principles).
- **Iterativer Durchlauf:** Pro ausgewählter Datei kann ein Container gestartet oder jeder andere notwendige Prozess ausgeführt werden.
- **Fehlerrobustheit:** Bei keiner Auswahl erfolgt eine nachvollziehbare Rückmeldung.

---

```

Gibt es besondere Anforderungen an das weitere Handling der mehrfach ausgewählten Dateien (z.B. gemeinsames Bearbeiten in einem Schritt, einzelne Durchläufe pro Datei etc.)? Teile dies bitte mit, damit die Lösung optimal auf deinen Use-Case passt!

**Bousfiha, Tanae:**

so ist richtig def main():

```
xnat_host = "https://xnat-dev.gwdg.de"
```

```
docker_base_image = "python:3.10"
```

```
xnat_user = get_input("XNAT Username: ")
```

```
xnat_password = getpass.getpass("XNAT Password: ")
```

```
project_id = get_input("Project ID: ").strip()
```

```
script_path = get_input("Path to the Python script: ")
```

```
# 1. Hole die Basisdaten/Eingaben
```

```
mod_data = modification()
```

```
# 2. Ergänze die nötigen Felder für downstream
```

```
mod_data["contexts"] = [mod_data["selected_context"]]
```

```
mod_data["label_name"] = mod_data["command_name"]
```

```
mod_data["label_description"] =
```

```
mod_data["command_description"]
```

```
wrapper_name = mod_data["command_name"].replace(" ",
"_").lower() + "_wrapper"#Generiert technische Bezeichnung
fuer einen xnat wrapper
```

```
# 3. Jetzt kann überall darauf zugegriffen werden
```

```
dockerfile_path = write_dockerfile(",", os.path.basename(script_path),
docker_base_image)
```

```
local_image_name =
```

```
f"{mod_data["command_name"].lower().replace(" ",
"_")}":latest
```

```
full_image_name = build_and_push_docker_image(dockerfile_path,
local_image_name)
```

```
json_file_path = create_json_file(full_image_name, os.path.basename(script_path),
mod_data)
```

```
send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)
```

```
command_id = get_command_id_by_name(xnat_host, xnat_user, xnat_password,
mod_data["command_name"])
```

```
try:
```

```

        wrapper_id = get_wrapper_id_by_command_name(
            xnat_host, xnat_user, xnat_password,
mod_data[&quot;command_name&quot;], wrapper_name
        )
        print(f&quot;Wrapper already exists: {wrapper_id}&quot;)
except SystemExit:
    print(&quot; Wrapper not found &quot;)
    outputs, external_inputs, derived_inputs = get_command_id(
        xnat_host, xnat_user, xnat_password, command_id
    )

    enable_wrapper_sitewide(xnat_host, command_id, wrapper_name, xnat_user,
xnat_password)
    enable_wrapper_for_project(xnat_host, project_id, command_id, wrapper_name,
xnat_user, xnat_password)

```

```

CONTEXT_ENTITY_MAPPING = {
    &quot;xnat:projectData&quot;:: {&quot;entity_type&quot;:: &quot;project&quot;,
&quot;id_label&quot;:: &quot;Project ID&quot;},
    &quot;xnat:subjectData&quot;:: {&quot;entity_type&quot;:: &quot;subject&quot;,
&quot;id_label&quot;:: &quot;Subject ID&quot;},
    &quot;xnat:mrSessionData&quot;:: {&quot;entity_type&quot;::
&quot;session&quot;, &quot;id_label&quot;:: &quot;Session ID&quot;},
    &quot;xnat:petSessionData&quot;:: {&quot;entity_type&quot;::
&quot;session&quot;, &quot;id_label&quot;:: &quot;Session ID&quot;},
    &quot;xnat:ctSessionData&quot;:: {&quot;entity_type&quot;::
&quot;session&quot;, &quot;id_label&quot;:: &quot;Session ID&quot;},
    &quot;xnat:sessionData&quot;:: {&quot;entity_type&quot;:: &quot;session&quot;,
&quot;id_label&quot;:: &quot;Session ID&quot;},
    &quot;xnat:imageScanData&quot;:: {&quot;entity_type&quot;:: &quot;scan&quot;,
&quot;id_label&quot;:: &quot;Session ID + Scan ID&quot;},
    &quot;xnat:resource&quot;:: {&quot;entity_type&quot;:: &quot;resource&quot;,
&quot;id_label&quot;:: &quot;Resource (not supported)&quot;}
}

```

```

first_context = mod_data[&quot;contexts&quot;][0]
context_info = CONTEXT_ENTITY_MAPPING.get(first_context)

```

```

if not context_info:
    print(f"Unbekannter Kontext: {first_context}. Abbruch.")
    return

entity_type = context_info["entity_type"]

if entity_type == "scan":
    session_id = get_input("Gib die Session-ID ein: ")
    scan_id = get_input("Gib die Scan-ID ein: ")
    input_file_info = get_input_file(
        xnat_host, session_id, entity_type, xnat_user, xnat_password,
scan_id=scan_id
    )
    container_entity_id = session_id

elif entity_type in ["session", "subject", "project"]:
    entity_id = get_input(f"Gib die {context_info[&#039;id_label&#039;]} ein:
")
    input_file_info = get_input_file(
        xnat_host, entity_id, entity_type, xnat_user, xnat_password
    )
    container_entity_id = entity_id
    scan_id = None # wird nicht gebraucht

else:
    print("Dieser Kontext wird aktuell nicht unterstützt.")
    return

def select_files(all_files):
    if not all_files:
        print("Keine verfügbaren Dateien gefunden.")
        return []
    print("Verfügbare Dateien:")
    for idx, f in enumerate(all_files, start=1):
        print(f"{idx}: {f[&#039;name&#039;]}")
    while True:
        choice = input("Nummern der Dateien auswählen (z.B. 1,3,5): ")
        indices = [c.strip() for c in choice.split(&#039;,&#039;)]

```

```

if all(i.isdigit() and 1 <= int(i) <= len(all_files) for i in indices):
    unique_indices = list(set(int(i) for i in indices))
    selected_files = [all_files[i-1] for i in unique_indices]
    print(""Ausgewählte Dateien:"")
    for s in selected_files:
        print(f""- {s['name']}"")
    return selected_files
else:
    print(""Ungültige Auswahl. Bitte wiederholen."")

# Beispielhafte Verwendung im main()
if entity_type == ""scan"":
    session_id = get_input(""Gib die Session-ID ein: "")
    scan_id = get_input(""Gib die Scan-ID ein: "")
    all_files = get_input_file(
        xnat_host, session_id, entity_type, xnat_user, xnat_password, scan_id=scan_id
    )
    container_entity_id = session_id

elif entity_type in ["session", "subject", "project"]:
    entity_id = get_input(f""Gib die {context_info['id_label']} ein: "")
    all_files = get_input_file(
        xnat_host, entity_id, entity_type, xnat_user, xnat_password
    )
    container_entity_id = entity_id
    scan_id = None

else:
    print(""Dieser Kontext wird aktuell nicht unterstützt."")

return

# Hier kommt die Modifikation!
selected_files = select_files(all_files)

if selected_files:
    # Hier setzt du ggf. deine Logik für jede ausgewählte Datei an:

```

```

for input_file_info in selected_files:
    launch_container_rest(
        xnat_host,
        project_id,
        command_id,
        wrapper_name,
        container_entity_id,
        xnat_user,
        xnat_password,
        input_file_info,
        entity_type=entity_type,
        scan_id=scan_id if entity_type == "scan" else None
    )
else:
    print(""Keine Datei ausgewählt. Containerstart abgebrochen."")

if __name__ == "__main__":
    main()

```

### **HAWKI (gpt-4.1):**

Du bist schon sehr nah an der funktionalen Lösung für Mehrfachauswahl. Es gibt aber einige **\*\*strukturelle und syntaktische Probleme\*\*** im von dir geposteten Code, insbesondere bezüglich der Funktionseinbettung und des mehrmaligen Aufrufs der Eingabe.

### **### Optimierte und korrekte Version**

Hier ist ein **\*\*bereinigter und funktionsfähiger\*\*** Vorschlag, der die Auswahl mehrerer Dateien integriert und strukturell korrekt ist:

```

```python
def select_files(all_files):
    if not all_files:
        print("Keine verfügbaren Dateien gefunden.")
        return []
    print("Verfügbare Dateien:")
    for idx, f in enumerate(all_files, start=1):

```



```

    print(f"{idx}: {f['name']}")
while True:
    choice = input("Nummern der Dateien auswählen (z.B. 1,3,5): ")
    indices = [c.strip() for c in choice.split(',') if c.strip()]
    if all(i.isdigit() and 1 <= int(i) <= len(all_files) for i in indices):
        unique_indices = list(set(int(i) for i in indices))
        selected_files = [all_files[i-1] for i in unique_indices]
        print("Ausgewählte Dateien:")
        for s in selected_files:
            print(f"- {s['name']}")
        return selected_files
    else:
        print("Ungültige Auswahl. Bitte wiederholen.")

def main():
    xnat_host = "https://xnat-dev.gwdg.de"
    docker_base_image = "python:3.10"

    xnat_user = get_input("XNAT Username: ")
    xnat_password = getpass.getpass("XNAT Password: ")
    project_id = get_input("Project ID: ").strip()
    script_path = get_input("Path to the Python script: ")

    # 1. Hole die Basisdaten/Eingaben
    mod_data = modification()

    # 2. Ergänze die nötigen Felder für downstream
    mod_data["contexts"] = [mod_data["selected_context"]]
    mod_data["label_name"] = mod_data["command_name"]
    mod_data["label_description"] = mod_data["command_description"]
    wrapper_name = mod_data["command_name"].replace(" ", "_").lower() +
    "_wrapper"

    # 3. Jetzt kann überall darauf zugegriffen werden
    dockerfile_path = write_dockerfile(".", os.path.basename(script_path),
docker_base_image)
    local_image_name = f"{mod_data['command_name'].lower().replace(' ', '_')}:latest"
    full_image_name = build_and_push_docker_image(dockerfile_path,

```

```
local_image_name)
```

```
    json_file_path = create_json_file(full_image_name, os.path.basename(script_path),  
mod_data)
```

```
    send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)
```

```
    command_id = get_command_id_by_name(xnat_host, xnat_user, xnat_password,  
mod_data["command_name"])
```

```
    try:
```

```
        wrapper_id = get_wrapper_id_by_command_name(  
            xnat_host, xnat_user, xnat_password, mod_data["command_name"],  
wrapper_name  
        )
```

```
        print(f"Wrapper already exists: {wrapper_id}")
```

```
    except SystemExit:
```

```
        print("Wrapper not found")
```

```
        outputs, external_inputs, derived_inputs = get_command_id(  
            xnat_host, xnat_user, xnat_password, command_id  
        )
```

```
    enable_wrapper_sitewide(xnat_host, command_id, wrapper_name, xnat_user,  
xnat_password)
```

```
    enable_wrapper_for_project(xnat_host, project_id, command_id, wrapper_name,  
xnat_user, xnat_password)
```

```
CONTEXT_ENTITY_MAPPING = {
```

```
    "xnat:projectData": {"entity_type": "project", "id_label": "Project ID"},
```

```
    "xnat:subjectData": {"entity_type": "subject", "id_label": "Subject ID"},
```

```
    "xnat:mrSessionData": {"entity_type": "session", "id_label": "Session ID"},
```

```
    "xnat:petSessionData": {"entity_type": "session", "id_label": "Session ID"},
```

```
    "xnat:ctSessionData": {"entity_type": "session", "id_label": "Session ID"},
```

```
    "xnat:sessionData": {"entity_type": "session", "id_label": "Session ID"},
```

```
    "xnat:imageScanData": {"entity_type": "scan", "id_label": "Session ID + Scan  
ID"},
```

```
    "xnat:resource": {"entity_type": "resource", "id_label": "Resource (not  
supported)"}  
}
```

```

first_context = mod_data["contexts"][0]
context_info = CONTEXT_ENTITY_MAPPING.get(first_context)

if not context_info:
    print(f"Unbekannter Kontext: {first_context}. Abbruch.")
    return

entity_type = context_info["entity_type"]

if entity_type == "scan":
    session_id = get_input("Gib die Session-ID ein: ")
    scan_id = get_input("Gib die Scan-ID ein: ")
    all_files = get_input_file(
        xnat_host, session_id, entity_type, xnat_user, xnat_password,
scan_id=scan_id
    )
    container_entity_id = session_id

elif entity_type in ["session", "subject", "project"]:
    entity_id = get_input(f"Gib die {context_info['id_label']} ein: ")
    all_files = get_input_file(
        xnat_host, entity_id, entity_type, xnat_user, xnat_password
    )
    container_entity_id = entity_id
    scan_id = None

else:
    print("Dieser Kontext wird aktuell nicht unterstützt.")
    return

# Mehrfachauswahl und Verarbeitung
selected_files = select_files(all_files)

if selected_files:
    for input_file_info in selected_files:
        launch_container_rest(
            xnat_host,

```

```

        project_id,
        command_id,
        wrapper_name,
        container_entity_id,
        xnat_user,
        xnat_password,
        input_file_info,
        entity_type=entity_type,
        scan_id=scan_id if entity_type == "scan" else None
    )
else:
    print("Keine Datei ausgewählt. Containerstart abgebrochen.")

if __name__ == "__main__":
    main()

```

**\*\*Wichtige Hinweise:\*\***

- Die Funktion `select\_files` steht **\*\*außerhalb\*\*** von `main` und wird auf die im jeweiligen Kontext ermittelten Datei-Listen angewendet.
- Der Inhalt von `main()` wurde korrigiert und auf die neue Logik angepasst.
- Doppelte Eingabeschritte, die du versehentlich übernommen hattest, sind entfernt.
- Syntaktische Fehler wie falsch eingerückte oder falsch platzierte Teile wurden bereinigt.

---

**\*\*Didaktisch:\*\***

Die saubere Trennung von Auswahl- und Hauptlogik folgt Grundsätzen der **\*\*funktionalen Dekomposition\*\*** und wirkt sich positiv auf Lesbarkeit und Wartbarkeit aus (z.B. nach Clean Code und Softwaretechnik-Prinzipien). Die Implementierung schützt zudem vor ungültigen Benutzereingaben und erhöht so die Robustheit.

Melde dich gern, wenn der Ablauf noch nicht exakt deinen Anforderungen entspricht oder du weitere Usability-Verbesserungen wünschst!

## Bousfiha, Tanae:

can i modify it so that he showes to the users all the data that in the context are and then give the opotunity to the user tio choose

## HAWKI (gpt-4.1):

Absolutely! That's a very user-friendly improvement.

**\*\*Instead of\*\***: Erst nach Eingabe der ID die Files zu holen und anzeigen...

**\*\*Besser\*\***: Zuerst alle möglichen "Entities" (z./B. alle Sessi Scans) im Kontext von Project oder Subject anzeigen, dann lässt du den\*die Nutzer\*in auswählen, und erst dann wählst du Dateien daraus aus.

**\*\*Wie geht das?\*\***

Du brauchst dazu:

1. **\*\*Eine Funktion, die die Entities im Kontext holt\*\*** (z./B. a Project, alle Subjects eines Projekts usw.)
2. **\*\*Eine Präsentation und Auswahl dieser Entities\*\*** via Menu
3. Dann, für die gewählten Entity/Entities: Wieder das File-Auswahl-Menü wie bisher

---

### ### Schritt-für-Schritt-Beispiel

#### #### 1. Entities holen und anzeigen

Angenommen, du hast für jede Entity-Tiefe eine Funktion, die die passende Liste zurückgibt. Beispiel:

```
- `get_sessions_for_project(xnat_host, project_id, xnat_user, xnat_password)`  
- `get_subjects_for_project(..)`
```

Diese Funktionen müssten im XNAT-API-Modul implementiert sein.

#### #### 2. Menü für Entities & für Dateien

Hier ist ein beispielhaftes (vereinfachtes) Auswahlmenü für Sessions in einem Projekt:

```

```python
def select_entity(entities, id_key='id', name_key='label'):
    if not entities:
        print("Keine verfügbaren Entities gefunden.")
        return None
    print("Verfügbare Entities:")
    for idx, ent in enumerate(entities, start=1):
        print(f"{idx}. {ent.get(id_key, '(no id)')} - {ent.get(name_key, '(no label)')}")
    while True:
        choice = input(f"Wähle eine Nummer (1-{len(entities)}): ")
        if choice.isdigit() and 1 <= int(choice) <= len(entities):
            return entities[int(choice)-1]
        print("Ungültige Auswahl.")
...

```

#### 3. Dein Ablauf am Beispiel „Alle Sessions eines Projekts anzeigen“

```

```python
if entity_type == "session":
    # Alle Sessions im Project holen
    all_sessions = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
    selected_session = select_entity(all_sessions, id_key='id', name_key='label')
    if not selected_session:
        print("Keine Session gewählt.")
        return

    session_id = selected_session['id']
    # Jetzt wie vorher: Datei-Auswahl
    all_files = get_input_file(xnat_host, session_id, "session", xnat_user,
xnat_password)
    selected_files = select_files(all_files)
    # ...restlicher Ablauf...
...

```

**\*\*Das gleiche Muster geht für Subjects oder andere Entities.\*\***

---

### ### Prinzip: Generische Lösung

Mache dir eine allgemeine select\_entity-Funktion.

Hole jeweils die Liste aus XNAT, dann wähle entity, dann zeige dort die Dateien.

---

### ### Fazit

- Ja, du kannst deine GUI so gestalten, dass Nutzende zuerst \*alle relevanten Entities\* sehen (Sessions, Subjects etc.).

- Die Struktur ist:

1. Kontext festlegen (entity\_type bestimmen)
2. Alle möglichen Entities holen und anzeigen
3. Entity auswählen lassen
4. Für die ausgewählte Entity die Files anzeigen und auswählen lassen

Das entspricht auch den Prinzipien von Usability Engineering (Nielsen: Konkrete Auswahl anstatt freie Eingabe; Fehlerminimierung, guided flow).

**\*\*Wissenschaftlicher Hinweis:\*\*** Je mehr der Arbeitsablauf „guided“ ist und alle Auswahloptionen sichtbar gemacht werden, desto geringer ist die Fehleranfälligkeit und der Frustrationsgrad. (Vgl. Nielsen, Usability Heuristics, 1994)

---

Melde dich gern, falls du ein Beispiel mit mehreren Auswahl-Multi-Choice für mehrere Sessions/Subjects und danach File-Auswahl pro Entity) oder ein Beispiel für die Implementierung deiner get\_Xxx-Funktionen brauchst!

### **Bousfiha, Tanae:**

here is my part what should be modified in order to get that

```
def get_input_file(xnat_host, entity_id, entity_type, xnat_user, xnat_password, scan_id=None):
```

```
    &quot;&quot;&quot;
```

Holt alle Dateien vom angegebenen XNAT-Objekt (Session, Subject, Project, etc.) und lässt den Benutzer eine Datei auswählen.  
""

```
# REST-Pfad je nach Kontexttyp
if entity_type == "project":
    base_url =
f"{xnat_host.rstrip(&#039;/&#039;))/data/projects/{entity_id}/resources"
    elif entity_type == "subject":
        base_url =
f"{xnat_host.rstrip(&#039;/&#039;))/data/subjects/{entity_id}/resources"
    elif entity_type in ["session", "experiment"]:
        base_url = f"{xnat_host.rstrip(&#039;/&#039;))/data/experiments/{entity_id}
/resources"
    elif entity_type == "scan" and scan_id:
        base_url = f"{xnat_host.rstrip(&#039;/&#039;))/data/experiments/{entity_id}
/scans/{scan_id}/resources"
    else:
        print("Unbekannter oder nicht unterstützter Entitätstyp.")
        return None

# Ressourcen abfragen
resp = requests.get(base_url, auth=(xnat_user, xnat_password), verify=False)
if resp.status_code != 200:
    print(f"Fehler beim Abrufen der Ressourcen ({entity_type}):
{resp.status_code}")
    return None

resources = resp.json().get("ResultSet", {}).get("Result", [])
all_files = []

for resource in resources:
    res_label = resource["label"]#jede Ressource muss ein Label haben
    file_url = f"{base_url}/{res_label}/files"#URL wird so gebaut, dass sie
auf die Datei-API der jeweiligen Ressource zeigt.
    file_resp = requests.get(file_url, auth=(xnat_user, xnat_password), verify=False)
    if file_resp.status_code != 200:
        continue
```



```

files = file_resp.json().get(&quot;ResultSet&quot;, {}).get(&quot;Result&quot;,
[])#Jede Datei wird als Dictionary mit folgenden Feldern
for f in files:
    all_files.append({
        &quot;name&quot;: f[&quot;Name&quot;],
        &quot;uri&quot;:
f&quot;,{base_url}/{res_label}/files/{f[&#039;Name&#039;]}&quot;,
        &quot;resource&quot;: res_label
    })

if not all_files:
    print(&quot;Keine Dateien gefunden.&quot;)
    return None

while True:
    choice=input(&quot;Welche Datei(en) sollen verwendet werden? Gib die
Nummern ein (z.B. 1,3,5):&quot;)
    indices=[c.strip()for c in choice.split(&quot;,&quot;)]

    if all(i.isdigit() and 1 &lt;= int(i) &lt;= len(all_files) for i in indices):
        unique_indices = list(set(int(i) for i in indices))
        selected = [all_files[i - 1] for i in unique_indices]
        print(&quot;Ausgewählte Dateien:&quot;)
        for s in selected:
            print(f&quot;- {s[&#039;name&#039;]}&quot;)
            return selected
        else:
            print(&quot;Ungültige Auswahl. Bitte gib gültige Nummern an.&quot;)

```

#### **HAWKI (gpt-4.1):**

Sehr gute Frage! Damit du **\*\*vor der Dateiauswahl erst alle relevanten Entities\*\*** (z.B. Sessions, Subjects, Scans) **\*\*anzeigen\*\*** und **\*\*auswählen\*\*** lassen kannst, musst du aus deiner Datei-Auswahl-Funktion (get\_input\_file) eine reine Datei-Liste-Funktion machen und das Entity-Menü in der Main-Logik oder einer separaten Funktion implementieren.

#### Schritt 1: `get\_input\_file` vereinfacht als Datei-Liste-Funktion

Deine Funktion holt jetzt **\*\*nur die Dateien\*\*** für eine übergebene Entity. Sie sollte **\*\*keine Benutzerauswahl mehr machen\*\***, sondern einfach die Daten liefern – das Menü machst du im Hauptablauf, nachdem du die Entities gezeigt und gewählt hast:

```
```python
def get_input_files(xnat_host, entity_id, entity_type, xnat_user, xnat_password,
scan_id=None):
    """
    Gibt alle Dateien vom angegebenen XNAT-Objekt zurück (OHNE
    Benutzerauswahl).
    """
    # REST-Pfad je nach Kontexttyp
    if entity_type == "project":
        base_url = f"{xnat_host.rstrip('/')}/data/projects/{entity_id}/resources"
    elif entity_type == "subject":
        base_url = f"{xnat_host.rstrip('/')}/data/subjects/{entity_id}/resources"
    elif entity_type in ["session", "experiment"]:
        base_url = f"{xnat_host.rstrip('/')}/data/experiments/{entity_id}/resources"
    elif entity_type == "scan" and scan_id:
        base_url =
f"{xnat_host.rstrip('/')}/data/experiments/{entity_id}/scans/{scan_id}/resources"
    else:
        print("Unbekannter oder nicht unterstützter Entitätstyp.")
        return []

    resp = requests.get(base_url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"Fehler beim Abrufen der Ressourcen ({entity_type}): {resp.status_code}")
        return []

    resources = resp.json().get("ResultSet", {}).get("Result", [])
    all_files = []
    for resource in resources:
        res_label = resource["label"]
        file_url = f"{base_url}/{res_label}/files"

```

```

file_resp = requests.get(file_url, auth=(xnat_user, xnat_password), verify=False)
if file_resp.status_code != 200:
    continue

files = file_resp.json().get("ResultSet", {}).get("Result", [])
for f in files:
    all_files.append({
        "name": f["Name"],
        "uri": f"{base_url}/{res_label}/files/{f['Name']}",
        "resource": res_label
    })
return all_files
...

```

----

### Schritt 2: Entities auflisten und auswählen lassen

Du brauchst Funktionen, die alle Entities im Kontext listen – z.B.  
``get_sessions_for_project()``. Dann ein Menü wie oben beschrieben:

```

```python
def select_entity(entities, id_key="ID", name_key="label"):
    if not entities:
        print("Keine Entities gefunden.")
        return None
    print("Verfügbare Entities:")
    for idx, entity in enumerate(entities, 1):
        print(f"{idx}: {entity.get(id_key, '---')} ({entity.get(name_key, '')})")
    while True:
        choice = input(f"Wähle eine Entity (1-{len(entities)}): ")
        if choice.isdigit() and 1 <= int(choice) <= len(entities):
            return entities[int(choice)-1]
        print("Ungültige Auswahl!")
...

```

### Schritt 3: Datei-Auswahl-Menü (kann wie bisher aussehen):

```

```python
def select_files_from_list(all_files):
    if not all_files:
        print("Keine Dateien gefunden.")
        return []
    print("Verfügbare Dateien:")
    for idx, f in enumerate(all_files, 1):
        print(f"{idx}: {f['name']} [Resource: {f['resource']}]")
    while True:
        choice = input("Welche Datei(en) sollen verwendet werden? Gib Nummern ein
(z.B. 1,3,5): ")
        indices = [c.strip() for c in choice.split(",") if c.strip()]
        if all(i.isdigit() and 1 <= int(i) <= len(all_files) for i in indices):
            unique_indices = list(set(int(i) for i in indices))
            selected = [all_files[i-1] for i in unique_indices]
            print("Ausgewählt:")
            for s in selected:
                print(f"- {s['name']}")
            return selected
        print("Ungültige Auswahl.")
...

```

### Schritt 4: Kombiniere den Ablauf in Deinem main

Vereinfachtes Beispiel — für `session`-Entities:

```

```python
# Beispiel: Auswahl einer Session im Projekt, dann darauf Dateiauswahl
all_sessions = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
chosen_session = select_entity(all_sessions, id_key='ID', name_key='label') # Passe
ggf. Felder an!
if chosen_session:
    session_id = chosen_session['ID']
    files = get_input_files(xnat_host, session_id, 'session', xnat_user, xnat_password)
    selected_files = select_files_from_list(files)
    # Eventuelle Weiterverarbeitung...
...

```

---

**\*\*Fazit:\*\***

- Die Auswahl der Entity-Ebene ist in deiner Main-Logik.
- Die Funktion `get\_input\_files` ist 'dumm' und gibt einfach nur ALLE Dateien zurück.
- Das Datei-Auswahl-Menü nutzt die Dateiliste dieser Funktion.

**\*\*Vorteil:\*\***

Sehr gute User Experience und klare Logik!

Diese Herangehensweise entspricht u.a. den Prinzipien der "Separation of Concerns" in der Softwaretechnik und verbessert den Bedienfluss laut Usability-Heuristiken.

**\*\*Gerne weitere Ausarbeitung bei komplexeren Kontexten (Scans in Sessions etc.)\*\***  
– sag gern, was dein Hauptanwendungsfall ist!

**Bousfiha, Tanae:**

```
def select_files(all_files):
    if not all_files:
        print(""Keine verfügbaren Dateien gefunden."")
        return []
    print(""Verfügbare Dateien:"")
    for idx, f in enumerate(all_files, start=1):
        print(f""{idx}: {f['name']}"")
    while True:
        choice = input(""Nummern der Dateien auswählen (z.B. 1,3,5): "")
        indices = [c.strip() for c in choice.split(',') if c.strip()]
        if all(i.isdigit() and 1 <= int(i) <= len(all_files) for i in indices):
            unique_indices = list(set(int(i) for i in indices))
            selected_files = [all_files[i-1] for i in unique_indices]
            print(""Ausgewählte Dateien:"")
            for s in selected_files:
                print(f""{s['name']}"")
            return selected_files
        else:
            print(""Ungültige Auswahl. Bitte wiederholen."")
```

```

def main():
    xnat_host = "https://xnat-dev.gwdg.de"
    docker_base_image = "python:3.10"

    xnat_user = get_input("XNAT Username: ")
    xnat_password = getpass.getpass("XNAT Password: ")
    project_id = get_input("Project ID: ").strip()
    script_path = get_input("Path to the Python script: ")

    # 1. Hole die Basisdaten/Eingaben
    mod_data = modification()

    # 2. Ergänze die nötigen Felder für downstream
    mod_data["contexts"] = [mod_data["selected_context"]]
    mod_data["label_name"] = mod_data["command_name"]
    mod_data["label_description"] =
mod_data["command_description"]
    wrapper_name = mod_data["command_name"].replace(" ",
    "_").lower() + "_wrapper"

    # 3. Jetzt kann überall darauf zugegriffen werden
    dockerfile_path = write_dockerfile(",", os.path.basename(script_path),
docker_base_image)
    local_image_name =
f"{mod_data[&#039;command_name&#039;].lower().replace(&#039; &#039;,
&#039;_&#039;)}:latest"
    full_image_name = build_and_push_docker_image(dockerfile_path,
local_image_name)

    json_file_path = create_json_file(full_image_name, os.path.basename(script_path),
mod_data)
    send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)

    command_id = get_command_id_by_name(xnat_host, xnat_user, xnat_password,
mod_data["command_name"])

    try:

```

```

        wrapper_id = get_wrapper_id_by_command_name(
            xnat_host, xnat_user, xnat_password,
mod_data[&quot;command_name&quot;], wrapper_name
        )
        print(f&quot;Wrapper already exists: {wrapper_id}&quot;)
except SystemExit:
    print(&quot;Wrapper not found&quot;)
    outputs, external_inputs, derived_inputs = get_command_id(
        xnat_host, xnat_user, xnat_password, command_id
    )

    enable_wrapper_sitewide(xnat_host, command_id, wrapper_name, xnat_user,
xnat_password)
    enable_wrapper_for_project(xnat_host, project_id, command_id, wrapper_name,
xnat_user, xnat_password)

```

```

CONTEXT_ENTITY_MAPPING = {
    &quot;xnat:projectData&quot;: {&quot;entity_type&quot;: &quot;project&quot;,
&quot;id_label&quot;: &quot;Project ID&quot;},
    &quot;xnat:subjectData&quot;: {&quot;entity_type&quot;: &quot;subject&quot;,
&quot;id_label&quot;: &quot;Subject ID&quot;},
    &quot;xnat:mrSessionData&quot;: {&quot;entity_type&quot;:
&quot;session&quot;, &quot;id_label&quot;: &quot;Session ID&quot;},
    &quot;xnat:petSessionData&quot;: {&quot;entity_type&quot;:
&quot;session&quot;, &quot;id_label&quot;: &quot;Session ID&quot;},
    &quot;xnat:ctSessionData&quot;: {&quot;entity_type&quot;:
&quot;session&quot;, &quot;id_label&quot;: &quot;Session ID&quot;},
    &quot;xnat:sessionData&quot;: {&quot;entity_type&quot;: &quot;session&quot;,
&quot;id_label&quot;: &quot;Session ID&quot;},
    &quot;xnat:imageScanData&quot;: {&quot;entity_type&quot;: &quot;scan&quot;,
&quot;id_label&quot;: &quot;Session ID + Scan ID&quot;},
    &quot;xnat:resource&quot;: {&quot;entity_type&quot;: &quot;resource&quot;,
&quot;id_label&quot;: &quot;Resource (not supported)&quot;},
}

```

```

first_context = mod_data[&quot;contexts&quot;][0]
context_info = CONTEXT_ENTITY_MAPPING.get(first_context)

```

```

if not context_info:
    print(f"Unbekannter Kontext: {first_context}. Abbruch.")
    return

entity_type = context_info["entity_type"]

if entity_type == "scan":
    session_id = get_input("Gib die Session-ID ein: ")
    scan_id = get_input("Gib die Scan-ID ein: ")
    all_files = get_input_file(
        xnat_host, session_id, entity_type, xnat_user, xnat_password,
scan_id=scan_id
    )
    container_entity_id = session_id

elif entity_type in ["session", "subject", "project"]:
    entity_id = get_input(f"Gib die {context_info['id_label']} ein: ")
    all_files = get_input_file(
        xnat_host, entity_id, entity_type, xnat_user, xnat_password
    )
    container_entity_id = entity_id
    scan_id = None

else:
    print("Dieser Kontext wird aktuell nicht unterstützt.")
    return

# Mehrfachauswahl und Verarbeitung
selected_files = select_files(all_files)

if selected_files:
    for input_file_info in selected_files:
        launch_container_rest(
            xnat_host,
            project_id,
            command_id,
            wrapper_name,

```



```

        container_entity_id,
        xnat_user,
        xnat_password,
        input_file_info,
        entity_type=entity_type,
        scan_id=scan_id if entity_type == "scan" else None
    )
else:
    print(""Keine Datei ausgewählt. Containerstart abgebrochen."")

if __name__ == "__main__":
    main()

```

how this part should be modified

### **HAWKI (gpt-4.1):**

Sehr gute Frage! Damit den Nutzenden zunächst alle relevanten "Entities" (also z.B. alle Sessions, Subjects, Scans) angezeigt werden – **anstatt eine ID einzugeben** – sollte die Logik so verändert werden:

---

**\*\*1. Schreibe für jede Entity-Ebene (Projekt, Subject, Session, Scan) eine Funktion, die diese Entities ausliest.\*\***

Zum Beispiel:

```

```python
def get_sessions_for_project(xnat_host, project_id, xnat_user, xnat_password):
    url = f"{xnat_host.rstrip('/')}/data/projects/{project_id}/experiments?format=json"
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    sessions = resp.json().get("ResultSet", {}).get("Result", [])
    return sessions # Liste von Dicts mit mindestens 'ID' oder 'label'
```

```

Analog für Subjects und ggf. Scans.

---

**\*\*2. Schreibe eine generische Auswahlfunktion für Entities:\*\***

```

```python
def select_entity(entities, id_key="ID", name_key="label"):
    if not entities:
        print("Keine Auswahl vorhanden.")
        return None
    print("Mögliche Auswahlen:")
    for idx, ent in enumerate(entities, 1):
        print(f"{idx}: {ent.get(id_key, '')} ({ent.get(name_key, '')})")
    while True:
        choice = input(f"Wähle eine Nummer (1-{len(entities)}): ")
        if choice.isdigit() and 1 <= int(choice) <= len(entities):
            return entities[int(choice)-1]
        print("Ungültige Auswahl.")
...

```

**\*\*3. Modifiziere den relevanten Part in Deinem `main()`:\*\***

Statt die ID abzufragen, holst du nun die Entities und lässt die Auswahl zu:

```

```python
if entity_type == "scan":
    session_id = get_input("Gib die Session-ID ein: ") # Optional: Auch hier
    auswählbar machen!
    scan_list = get_scans_for_session(xnat_host, session_id, xnat_user,
    xnat_password)
    chosen_scan = select_entity(scan_list, id_key="ID", name_key="type")
    if not chosen_scan:
        print("Keine Auswahl. Abbruch.")
        return
    scan_id = chosen_scan["ID"]
    all_files = get_input_file(
        xnat_host, session_id, entity_type, xnat_user, xnat_password,
    scan_id=scan_id
    )
    container_entity_id = session_id

```

```

elif entity_type == "session":
    # Statt ID: Sessions im Projekt auflisten!
    session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
    chosen_session = select_entity(session_list, id_key="ID", name_key="label")
    if not chosen_session:
        print("Keine Auswahl. Abbruch.")
        return
    session_id = chosen_session["ID"]
    all_files = get_input_file(
        xnat_host, session_id, entity_type, xnat_user, xnat_password
    )
    container_entity_id = session_id
    scan_id = None

elif entity_type == "subject":
    subject_list = get_subjects_for_project(xnat_host, project_id, xnat_user,
xnat_password)
    chosen_subject = select_entity(subject_list, id_key="ID", name_key="label")
    if not chosen_subject:
        print("Keine Auswahl. Abbruch.")
        return
    subject_id = chosen_subject["ID"]
    all_files = get_input_file(
        xnat_host, subject_id, entity_type, xnat_user, xnat_password
    )
    container_entity_id = subject_id
    scan_id = None

elif entity_type == "project":
    # Normalerweise gibt es nur ein Projekt, aber du kannst auch alle Projekte
anzeigen
    all_files = get_input_file(
        xnat_host, project_id, entity_type, xnat_user, xnat_password
    )
    container_entity_id = project_id
    scan_id = None

```

...

---

**\*\*Fazit:\*\***

- Dadurch wird der Auswahlprozess viel intuitiver!
- Die User sehen, was zur Auswahl steht, und geben NICHT mehr irgendwelche IDs "auf gut Glück" ein.
- Das fördert Usability und minimiert Eingabefehler.

**\*\*Wissenschaftlich fundiert:\*\*** Durch geführte Menüs und explorative Navigation werden Fehlerraten reduziert und Benutzer:innen erleben weniger Frustration, was durch zahlreiche Nutzerstudien bestätigt wurde (vgl. Nielsen, Usability Heuristics).

---

**\*\*Zusatz:\*\***

Passe ggf. die Namen/Felder ("ID", "label", "type") je nach dem von deiner XNAT-Version gelieferten JSON-Format an.

Wenn du meinst, dass die Auswahl für mehrere Entity-Ebenen (z.B. erst Subject, dann Session) verschachtelt sein soll, sag bitte Bescheid – dann kann ich dir auch dazu eine Vorlage machen!

---

**\*\*Zusammengefasst (Pseudocode):\*\***

1. Hole alle Entities (Sessions, Subjects, Scans) je nach Kontext.
2. Zeige sie als Menü an und lass wählen.
3. Hole dann erst die Dateien und lasse die Datei(en) wählen.

Soll ich dir ein Beispiel für ein vollständiges Sessions-Auswahl-Menü samt Datei-Menü zeigen?

**Bousfiha, Tanae:**

can you give me the main part

**HAWKI (gpt-4.1):**

Absolutely! Here's a practical **\*\*main()-Part\*\*** (in English) that implements the entity-selection-then-file-selection logic in a clear and reusable structure. It assumes that you have the following helper functions implemented (as sketched in the previous answers):

- ``get_sessions_for_project(xnat_host, project_id, xnat_user, xnat_password)``
- ``get_subjects_for_project(...)``
- ``get_scans_for_session(...)``
- ``select_entity(entities, id_key, name_key)``
- ``get_input_file(xnat_host, entity_id, entity_type, xnat_user, xnat_password, scan_id=None)`` (as you already have)
- ``select_files(all_files)``
- ... and your other domain-specific helpers from earlier

---

```
```python
```

```
def main():
```

```
    xnat_host = "https://xnat-dev.gwdg.de"
```

```
    docker_base_image = "python:3.10"
```

```
    xnat_user = get_input("XNAT Username: ")
```

```
    xnat_password = getpass.getpass("XNAT Password: ")
```

```
    project_id = get_input("Project ID: ").strip()
```

```
    script_path = get_input("Path to the Python script: ")
```

```
    # 1. Gather modification/command data
```

```
    mod_data = modification()
```

```
    mod_data["contexts"] = [mod_data["selected_context"]]
```

```
    mod_data["label_name"] = mod_data["command_name"]
```

```
    mod_data["label_description"] = mod_data["command_description"]
```

```
    wrapper_name = mod_data["command_name"].replace(" ", "_").lower() +
```

```
"_wrapper"
```

```
# 2. Prepare docker and command upload
```

```
dockerfile_path = write_dockerfile(".", os.path.basename(script_path),
docker_base_image)
local_image_name = f"{mod_data['command_name'].lower().replace(' ', '_')}:latest"
full_image_name = build_and_push_docker_image(dockerfile_path,
local_image_name)
```

```
json_file_path = create_json_file(full_image_name, os.path.basename(script_path),
mod_data)
```

```
send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)
```

```
command_id = get_command_id_by_name(xnat_host, xnat_user, xnat_password,
mod_data["command_name"])
```

```
try:
```

```
    wrapper_id = get_wrapper_id_by_command_name(
        xnat_host, xnat_user, xnat_password, mod_data["command_name"],
wrapper_name
    )
```

```
    print(f"Wrapper already exists: {wrapper_id}")
```

```
except SystemExit:
```

```
    print("Wrapper not found")
```

```
    outputs, external_inputs, derived_inputs = get_command_id(
        xnat_host, xnat_user, xnat_password, command_id
    )
```

```
enable_wrapper_sitewide(xnat_host, command_id, wrapper_name, xnat_user,
xnat_password)
```

```
enable_wrapper_for_project(xnat_host, project_id, command_id, wrapper_name,
xnat_user, xnat_password)
```

```
CONTEXT_ENTITY_MAPPING = {
```

```
    "xnat:projectData": {"entity_type": "project", "id_label": "Project ID"},
```

```
    "xnat:subjectData": {"entity_type": "subject", "id_label": "Subject ID"},
```

```
    "xnat:mrSessionData": {"entity_type": "session", "id_label": "Session ID"},
```

```
    "xnat:petSessionData": {"entity_type": "session", "id_label": "Session ID"},
```

```

"xnat:ctSessionData": {"entity_type": "session", "id_label": "Session ID"},
"xnat:sessionData": {"entity_type": "session", "id_label": "Session ID"},
"xnat:imageScanData": {"entity_type": "scan", "id_label": "Session ID + Scan
ID"},
"xnat:resource": {"entity_type": "resource", "id_label": "Resource (not
supported)"}
}

```

```

first_context = mod_data["contexts"][0]
context_info = CONTEXT_ENTITY_MAPPING.get(first_context)

```

```

if not context_info:
    print(f"Unknown context: {first_context}. Aborting.")
    return

```

```

entity_type = context_info["entity_type"]

```

```

# ----- ENTITY CHOOSER -----

```

```

if entity_type == "scan":
    # 1. Pick session first
    session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
    chosen_session = select_entity(session_list, id_key="ID", name_key="label")
    if not chosen_session:
        print("No session selected.")
        return
    session_id = chosen_session["ID"]

```

```

    # 2. Pick scan in session
    scan_list = get_scans_for_session(xnat_host, session_id, xnat_user,
xnat_password)
    chosen_scan = select_entity(scan_list, id_key="ID",
name_key="series_description")
    if not chosen_scan:
        print("No scan selected.")
        return
    scan_id = chosen_scan["ID"]
    container_entity_id = session_id

```

### # 3. File selection

```
all_files = get_input_file(
    xnat_host, session_id, 'scan', xnat_user, xnat_password, scan_id=scan_id
)

elif entity_type == "session":
    session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
    chosen_session = select_entity(session_list, id_key="ID", name_key="label")
    if not chosen_session:
        print("No session selected.")
        return
    session_id = chosen_session["ID"]
    container_entity_id = session_id
    scan_id = None
    all_files = get_input_file(
        xnat_host, session_id, 'session', xnat_user, xnat_password
    )

elif entity_type == "subject":
    subject_list = get_subjects_for_project(xnat_host, project_id, xnat_user,
xnat_password)
    chosen_subject = select_entity(subject_list, id_key="ID", name_key="label")
    if not chosen_subject:
        print("No subject selected.")
        return
    subject_id = chosen_subject["ID"]
    container_entity_id = subject_id
    scan_id = None
    all_files = get_input_file(
        xnat_host, subject_id, 'subject', xnat_user, xnat_password
    )

elif entity_type == "project":
    container_entity_id = project_id
    scan_id = None
    all_files = get_input_file(
```



```

        xnat_host, project_id, 'project', xnat_user, xnat_password
    )

else:
    print("This context is currently not supported.")
    return

# ----- FILE CHOOSER -----
selected_files = select_files(all_files)

if selected_files:
    for input_file_info in selected_files:
        launch_container_rest(
            xnat_host,
            project_id,
            command_id,
            wrapper_name,
            container_entity_id,
            xnat_user,
            xnat_password,
            input_file_info,
            entity_type=entity_type,
            scan_id=scan_id if entity_type == "scan" else None
        )
    else:
        print("No file selected. Container launch aborted.")

if __name__ == "__main__":
    main()
...

**What should you provide/adjust?**
- You should replace `"series_description"` in `select_entity(scan_list, id_key="ID",
name_key="series_description")` with the most informative attribute your XNAT
system provides for scan naming.
- Adapt entity retrieval (`get_sessions_for_project`, etc.) and selection functions to
your actual data structure.

```

**\*\*Scientific background:\*\***

Modular, guided selection ensures clarity, reduces user input errors, follows established usability heuristics (visibility of system status/guidance), and matches typical research data lifecycles.

If you need example implementations for the entity-fetching helper functions, let me know!

**Bousfiha, Tanae:**

so

```
def select_files(all_files):
    if not all_files:
        print("&quot;Keine verfügbaren Dateien gefunden.&quot;")
        return []
    print("&quot;Verfügbare Dateien:&quot;")
    for idx, f in enumerate(all_files, start=1):
        print(f"&quot;{idx}: {f['name']}&quot;")
    while True:
        choice = input("&quot;Nummern der Dateien auswählen (z.B. 1,3,5): &quot;")
        indices = [c.strip() for c in choice.split(',') if c.strip()]
        if all(i.isdigit() and 1 <= int(i) <= len(all_files) for i in indices):
            unique_indices = list(set(int(i) for i in indices))
            selected_files = [all_files[i-1] for i in unique_indices]
            print("&quot;Ausgewählte Dateien:&quot;")
            for s in selected_files:
                print(f"&quot;{s['name']}&quot;")
            return selected_files
        else:
            print("&quot;Ungültige Auswahl. Bitte wiederholen.&quot;")
def main():
    xnat_host = "&quot;https://xnat-dev.gwdg.de&quot;"
    docker_base_image = "&quot;python:3.10&quot;"

    xnat_user = get_input("&quot;XNAT Username: &quot;")
    xnat_password = getpass.getpass("&quot;XNAT Password: &quot;")
    project_id = get_input("&quot;Project ID: &quot;").strip()
    script_path = get_input("&quot;Path to the Python script: &quot;")
```

```

# 1. Gather modification/command data
mod_data = modification()
mod_data[&quot;contexts&quot;] = [mod_data[&quot;selected_context&quot;]]
mod_data[&quot;label_name&quot;] = mod_data[&quot;command_name&quot;]
mod_data[&quot;label_description&quot;] =
mod_data[&quot;command_description&quot;]
    wrapper_name = mod_data[&quot;command_name&quot;].replace(&quot; &quot;,
&quot;_&quot;).lower() + &quot;_wrapper&quot;

# 2. Prepare docker and command upload
dockerfile_path = write_dockerfile(&quot;.&quot;, os.path.basename(script_path),
docker_base_image)
local_image_name =
f&quot;{mod_data[&#039;command_name&#039;].lower().replace(&#039; &#039;,
&#039;_&#039;)}:latest&quot;
    full_image_name = build_and_push_docker_image(dockerfile_path,
local_image_name)

json_file_path = create_json_file(full_image_name, os.path.basename(script_path),
mod_data)
    send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)

    command_id = get_command_id_by_name(xnat_host, xnat_user, xnat_password,
mod_data[&quot;command_name&quot;])

try:
    wrapper_id = get_wrapper_id_by_command_name(
        xnat_host, xnat_user, xnat_password,
mod_data[&quot;command_name&quot;], wrapper_name
    )
    print(f&quot;Wrapper already exists: {wrapper_id}&quot;)
except SystemExit:
    print(&quot;Wrapper not found&quot;)
    outputs, external_inputs, derived_inputs = get_command_id(
        xnat_host, xnat_user, xnat_password, command_id
    )

```

```

enable_wrapper_sitewide(xnat_host, command_id, wrapper_name, xnat_user,
xnat_password)
enable_wrapper_for_project(xnat_host, project_id, command_id, wrapper_name,
xnat_user, xnat_password)

```

```

CONTEXT_ENTITY_MAPPING = {
    "xnat:projectData": {"entity_type": "project",
    "id_label": "Project ID"},
    "xnat:subjectData": {"entity_type": "subject",
    "id_label": "Subject ID"},
    "xnat:mrSessionData": {"entity_type":
    "session", "id_label": "Session ID"},
    "xnat:petSessionData": {"entity_type":
    "session", "id_label": "Session ID"},
    "xnat:ctSessionData": {"entity_type":
    "session", "id_label": "Session ID"},
    "xnat:sessionData": {"entity_type": "session",
    "id_label": "Session ID"},
    "xnat:imageScanData": {"entity_type": "scan",
    "id_label": "Session ID + Scan ID"},
    "xnat:resource": {"entity_type": "resource",
    "id_label": "Resource (not supported)"}
}

```

```

first_context = mod_data["contexts"][0]
context_info = CONTEXT_ENTITY_MAPPING.get(first_context)

```

```

if not context_info:
    print(f"Unknown context: {first_context}. Aborting.")
    return

```

```

entity_type = context_info["entity_type"]

```

```

# ----- ENTITY CHOOSER -----

```

```

if entity_type == "scan":
    # 1. Pick session first
    session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)

```

```

        chosen_session = select_entity(session_list, id_key='ID',
name_key='label')
        if not chosen_session:
            print('No session selected.')
            return
        session_id = chosen_session['ID']

# 2. Pick scan in session
        scan_list = get_scans_for_session(xnat_host, session_id, xnat_user,
xnat_password)
        chosen_scan = select_entity(scan_list, id_key='ID',
name_key='series_description')
        if not chosen_scan:
            print('No scan selected.')
            return
        scan_id = chosen_scan['ID']
        container_entity_id = session_id

# 3. File selection
        all_files = get_input_file(
            xnat_host, session_id, 'scan', xnat_user, xnat_password,
scan_id=scan_id
        )

elif entity_type == 'session':
        session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
        chosen_session = select_entity(session_list, id_key='ID',
name_key='label')
        if not chosen_session:
            print('No session selected.')
            return
        session_id = chosen_session['ID']
        container_entity_id = session_id
        scan_id = None
        all_files = get_input_file(
            xnat_host, session_id, 'session', xnat_user, xnat_password
        )

```

```

elif entity_type == "subject":
    subject_list = get_subjects_for_project(xnat_host, project_id, xnat_user,
xnat_password)
    chosen_subject = select_entity(subject_list, id_key="ID",
name_key="label")
    if not chosen_subject:
        print("No subject selected.")
        return
    subject_id = chosen_subject["ID"]
    container_entity_id = subject_id
    scan_id = None
    all_files = get_input_file(
        xnat_host, subject_id, "subject", xnat_user, xnat_password
    )

elif entity_type == "project":
    container_entity_id = project_id
    scan_id = None
    all_files = get_input_file(
        xnat_host, project_id, "project", xnat_user, xnat_password
    )

else:
    print("This context is currently not supported.")
    return

# ----- FILE CHOOSER -----
selected_files = select_files(all_files)

if selected_files:
    for input_file_info in selected_files:
        launch_container_rest(
            xnat_host,
            project_id,
            command_id,
            wrapper_name,
            container_entity_id,

```

```

        xnat_user,
        xnat_password,
        input_file_info,
        entity_type=entity_type,
        scan_id=scan_id if entity_type == "scan" else None
    )
else:
    print("No file selected. Container launch aborted.")

if __name__ == "__main__":
    main()

```

## **HAWKI (gpt-4.1):**

Yes, **this is the right direction!**

With this `main()` structure, users first select the appropriate entity (session, subject, scan, project) based on the context, and **then** select one or several files via the interactive menu.

### What to check/adjust before production

### 1. **Helper functions**

Make sure the following exist and return lists of dictionaries with the right keys (like `ID`, `label`, `series\_description`, etc.):

- `get\_sessions\_for\_project(xnat\_host, project\_id, xnat\_user, xnat\_password)`
- `get\_subjects\_for\_project(xnat\_host, project\_id, xnat\_user, xnat\_password)`
- `get\_scans\_for\_session(xnat\_host, session\_id, xnat\_user, xnat\_password)`
- `get\_input\_file(...)` (already checked; returns file list)
- `select\_entity(entities, id\_key, name\_key)` as discussed earlier

### 2. **Key names in select\_entity**

- Check that the keys you use (e.g. `ID`, `label`, `series\_description`) actually match the structure of your XNAT JSON/data. If they differ, change the values in `select\_entity` calls accordingly.

### 3. **Error handling**

- All paths ("no entity selected") lead to an early return—good!
- If no files exist, your `select\_files()` will tell users this and return an empty list—so no risk of a crash.

#### 4. **\*\*Readability / Structure\*\***

- Consider making each major block (project, subject, session, scan) a separate function in the future (Clean Code principle), but for now your structure is perfectly fine and readable.

#### ### Optional Improvements

- **\*\*Internationalization/UI:\*\*** You might want to unify the user info texts into one language (German/English).
- **\*\*Sorting entities and files:\*\*** You could sort by label or name for better UX.
- **\*\*Show more info:\*\*** When showing available scans etc., showing additional fields (modality, date, etc.) often helps users.

---

#### ### Scientific & Usability Perspective

- **\*\*Guided selection\*\*** like this minimizes user errors and exploratory frustration (in line with Jakob Nielsen's usability heuristics).
- **\*\*Early validation\*\*** (via menu) reduces the risk of invalid input and improves user satisfaction.
- **\*\*Separation of concerns\*\***: Clean handling of files vs. entities vs. execution step.

---

#### #### Summary

**\*\*Your logic is good.\*\*** Provided that your helper functions exist and the dictionary keys are correct, this menu-driven approach is best-practice for usability and robustness.

If you need a template for a missing helper, let me know!

If you want to show more info per entity (session date, scan modality, etc.), just say which extra fields you need.



## Bousfiha, Tanae:

he did not recognize the `get_sessions_for_project`

## HAWKI (gpt-4.1):

Das ist ein häufiger Punkt: `get_sessions_for_project` (und analoge Funktionen wie `get_subjects_for_project`, `get_scans_for_session`) sind Platzhalter für Hilfsfunktionen, die du selbst bereitstellen musst. Sie sind nicht im Standard-XNAT-Python-API-Modul enthalten, weil XNAT ein generisches Datenmodell hat.

Hier ist eine beispielhafte Implementierung für alle drei häufigen Fälle, angepasst auf die XNAT-REST-API und dein Nutzungsszenario:

```
python
import requests

def get_sessions_for_project(xnat_host, project_id, xnat_user, xnat_password):
    """
    Liste alle Sessions (Experimente) für ein Projekt.
    """
    url = f"{xnat_host.rstrip('/')}/data/projects/{project_id}/experiments?format=json"
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"Fehler beim Abrufen der Sessions für Projekt {project_id}: {resp.status_code}")
    return []
    return resp.json().get("ResultSet", {}).get("Result", [])

def get_subjects_for_project(xnat_host, project_id, xnat_user, xnat_password):
    """
    Liste alle Subjects für ein Projekt.
    """
    url = f"{xnat_host.rstrip('/')}/data/projects/{project_id}/subjects?format=json"
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"Fehler beim Abrufen der Subjects für Projekt {project_id}:
```

```

{resp.status_code}")
    return []
    return resp.json().get("ResultSet", {}).get("Result", [])

def get_scans_for_session(xnat_host, session_id, xnat_user, xnat_password):
    """
    Liste alle Scans für eine Session (Experiment).
    """
    url = f"{xnat_host.rstrip('/')}/data/experiments/{session_id}/scans?format=json"
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"Fehler beim Abrufen der Scans für Session {session_id}:
{resp.status_code}")
        return []
    return resp.json().get("ResultSet", {}).get("Result", [])
...

```

**\*\*Beachte:\*\***

- Rückgabewert ist jeweils eine Liste von Dictionaries, aus der du mit ``["ID"]``, ``["label"]``, ``["series_description"]``, etc. arbeiten kannst.
- Manche XNAT-Systeme liefern unterschiedliche Feldnamen – schau dir ggf. mit ``print()`` einmal die tatsächlichen JSON-Daten an!

---

**\*\*Tipp zum Testen\*\***

Du kannst das mal ausprobieren:

```

```python
sessions = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
print("Sessions:")
for s in sessions:
    print(s)
...

```

So siehst du alle Schlüssel und kannst gezielt die passenden Namen in deiner Auswahl verwenden.

---

**\*\*Wissenschaftlicher Hinweis:\*\***

Durch das explizite Aufführen und Auswählen der Entities wird das Prinzip der Sichtbarkeit und Fehlerprävention umgesetzt (vgl. Usability Heuristics von Jakob Nielsen). Nutzer\*innen erhalten Kontext, können explorativ arbeiten und geraten nicht in Dead-Ends oder Fehlerfälle.

---

**\*\*Falls du für weitere Entity-Typen (e.g. Ressourcen) oder zum Traversieren der Hierarchie Hilfe brauchst, gib kurz Bescheid!\*\***

### **Bousfiha, Tanae:**

this part needs correction

```
def main():
```

```
    xnat_host = &quot;https://xnat-dev.gwdg.de&quot;
```

```
    docker_base_image = &quot;python:3.10&quot;
```

```
    xnat_user = get_input(&quot;XNAT Username: &quot;)
```

```
    xnat_password = getpass.getpass(&quot;XNAT Password: &quot;)
```

```
    project_id = get_input(&quot;Project ID: &quot;).strip()
```

```
    script_path = get_input(&quot;Path to the Python script: &quot;)
```

```
    # 1. Gather modification/command data
```

```
    mod_data = modification()
```

```
    mod_data[&quot;contexts&quot;] = [mod_data[&quot;selected_context&quot;]]
```

```
    mod_data[&quot;label_name&quot;] = mod_data[&quot;command_name&quot;]
```

```
    mod_data[&quot;label_description&quot;] =
```

```
mod_data[&quot;command_description&quot;]
```

```
    wrapper_name = mod_data[&quot;command_name&quot;].replace(&quot; &quot;,
&quot;_&quot;).lower() + &quot;_wrapper&quot;
```

```
    # 2. Prepare docker and command upload
```

```
    dockerfile_path = write_dockerfile(&quot;.&quot;, os.path.basename(script_path),
docker_base_image)
```

```
    local_image_name =
```

```
f&quot;{mod_data[&quot;command_name&quot;].lower().replace(&quot; &quot;,
```

```

    '&#039;_&#039;)):latest&quot;
    full_image_name = build_and_push_docker_image(dockerfile_path,
    local_image_name)

    json_file_path = create_json_file(full_image_name, os.path.basename(script_path),
    mod_data)
    send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)

    command_id = get_command_id_by_name(xnat_host, xnat_user, xnat_password,
    mod_data[&quot;command_name&quot;])

    try:
        wrapper_id = get_wrapper_id_by_command_name(
            xnat_host, xnat_user, xnat_password,
            mod_data[&quot;command_name&quot;], wrapper_name
        )
        print(f&quot;Wrapper already exists: {wrapper_id}&quot;)
    except SystemExit:
        print(&quot;Wrapper not found&quot;)
        outputs, external_inputs, derived_inputs = get_command_id(
            xnat_host, xnat_user, xnat_password, command_id
        )

    enable_wrapper_sitewide(xnat_host, command_id, wrapper_name, xnat_user,
    xnat_password)
    enable_wrapper_for_project(xnat_host, project_id, command_id, wrapper_name,
    xnat_user, xnat_password)

    CONTEXT_ENTITY_MAPPING = {
        &quot;xnat:projectData&quot;: {&quot;entity_type&quot;: &quot;project&quot;,
        &quot;id_label&quot;: &quot;Project ID&quot;},
        &quot;xnat:subjectData&quot;: {&quot;entity_type&quot;: &quot;subject&quot;,
        &quot;id_label&quot;: &quot;Subject ID&quot;},
        &quot;xnat:mrSessionData&quot;: {&quot;entity_type&quot;:
        &quot;session&quot;, &quot;id_label&quot;: &quot;Session ID&quot;},
        &quot;xnat:petSessionData&quot;: {&quot;entity_type&quot;:
        &quot;session&quot;, &quot;id_label&quot;: &quot;Session ID&quot;},
        &quot;xnat:ctSessionData&quot;: {&quot;entity_type&quot;:

```

```

    &quot;session&quot;:, &quot;id_label&quot;: &quot;Session ID&quot;},
    &quot;xnat:sessionData&quot;: {&quot;entity_type&quot;: &quot;session&quot;,
    &quot;id_label&quot;: &quot;Session ID&quot;},
    &quot;xnat:imageScanData&quot;: {&quot;entity_type&quot;: &quot;scan&quot;,
    &quot;id_label&quot;: &quot;Session ID + Scan ID&quot;},
    &quot;xnat:resource&quot;: {&quot;entity_type&quot;: &quot;resource&quot;,
    &quot;id_label&quot;: &quot;Resource (not supported)&quot;},
    }

```

```

first_context = mod_data[&quot;contexts&quot;][0]
context_info = CONTEXT_ENTITY_MAPPING.get(first_context)

```

```

if not context_info:
    print(f&quot;Unknown context: {first_context}. Aborting.&quot;)
    return

```

```

entity_type = context_info[&quot;entity_type&quot;]

```

```

# ----- ENTITY CHOOSER -----

```

```

if entity_type == &quot;scan&quot;:
    # 1. Pick session first
    session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
    chosen_session = select_entity(session_list, id_key=&quot;ID&quot;,
name_key=&quot;label&quot;,)
    if not chosen_session:
        print(&quot;No session selected.&quot;)
        return
    session_id = chosen_session[&quot;ID&quot;]

```

```

    # 2. Pick scan in session
    scan_list = get_scans_for_session(xnat_host, session_id, xnat_user,
xnat_password)
    chosen_scan = select_entity(scan_list, id_key=&quot;ID&quot;,
name_key=&quot;series_description&quot;,)
    if not chosen_scan:
        print(&quot;No scan selected.&quot;)
        return

```

```
scan_id = chosen_scan['ID']
container_entity_id = session_id
```

# 3. File selection

```
all_files = get_input_files(
    xnat_host, session_id, 'scan', xnat_user, xnat_password,
    scan_id=scan_id
)
```

```
elif entity_type == 'session':
    session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
    xnat_password)
    chosen_session = select_entity(session_list, id_key='ID',
    name_key='label')
    if not chosen_session:
        print('No session selected.')
        return
    session_id = chosen_session['ID']
    container_entity_id = session_id
    scan_id = None
    all_files = get_input_files(
        xnat_host, session_id, 'session', xnat_user, xnat_password
    )
```

```
elif entity_type == 'subject':
    subject_list = get_subjects_for_project(xnat_host, project_id, xnat_user,
    xnat_password)
    chosen_subject = select_entity(subject_list, id_key='ID',
    name_key='label')
    if not chosen_subject:
        print('No subject selected.')
        return
    subject_id = chosen_subject['ID']
    container_entity_id = subject_id
    scan_id = None
    all_files = get_input_files(
        xnat_host, subject_id, 'subject', xnat_user, xnat_password
    )
```

```

elif entity_type == "project":
    container_entity_id = project_id
    scan_id = None
    all_files = get_input_files(
        xnat_host, project_id, "project", xnat_user, xnat_password
    )

else:
    print("This context is currently not supported.")
    return

# ----- FILE CHOOSER -----
selected_files = select_files(all_files)

if selected_files:
    for input_file_info in selected_files:
        launch_container_rest(
            xnat_host,
            project_id,
            command_id,
            wrapper_name,
            container_entity_id,
            xnat_user,
            xnat_password,
            input_file_info,
            entity_type=entity_type,
            scan_id=scan_id if entity_type == "scan" else None
        )
    else:
        print("No file selected. Container launch aborted.")

if __name__ == "__main__":
    main()

```

**HAWKI (gpt-4.1):**

Your main function is structurally almost correct, but here are a few **\*\*important**

corrections\*\* and didactic notes for improved clarity, error handling, modularity, and maintainability:

#### ### 1. \*\*Function Names Consistency\*\*

Ensure you are calling `get_input_files` everywhere (not `get_input_file`) and that this function actually `_returns` a list of files\_ without further user interaction.

#### ### 2. \*\*Correct Key Handling for Entities\*\*

Make sure the entity dictionary key names (like `"ID"`, `"label"`, `"series_description"`) match those returned by your helper functions (for XNAT, check what your server returns for JSON fields!).

#### ### 3. \*\*Helper Functions Existence\*\*

Ensure you have implemented (as in my previous post):

- `get_sessions_for_project`
- `get_subjects_for_project`
- `get_scans_for_session`
- `select_entity`

#### ### 4. \*\*Didactic and Modular Correction\*\*

Below is an improved version with some small but important tweaks and more robust checks, plus improved comments for educational clarity.

---

```
python
def main():
    import os
    import getpass

    xnat_host = "https://xnat-dev.gwdg.de"
    docker_base_image = "python:3.10"

    xnat_user = get_input("XNAT Username: ")
    xnat_password = getpass.getpass("XNAT Password: ")
    project_id = get_input("Project ID: ").strip()
    script_path = get_input("Path to the Python script: ")
```



```

# Step 1: Gather Command/Wrapper Data
mod_data = modification()
mod_data["contexts"] = [mod_data["selected_context"]]
mod_data["label_name"] = mod_data["command_name"]
mod_data["label_description"] = mod_data["command_description"]
wrapper_name = mod_data["command_name"].replace(" ", "_").lower() +
"_wrapper"

# Step 2: Prepare and upload Docker image and command
dockerfile_path = write_dockerfile(".", os.path.basename(script_path),
docker_base_image)
local_image_name = f"{mod_data['command_name'].lower().replace(' ', '_')}:latest"
full_image_name = build_and_push_docker_image(dockerfile_path,
local_image_name)

json_file_path = create_json_file(full_image_name, os.path.basename(script_path),
mod_data)
send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)

command_id = get_command_id_by_name(xnat_host, xnat_user, xnat_password,
mod_data["command_name"])

try:
    wrapper_id = get_wrapper_id_by_command_name(
        xnat_host, xnat_user, xnat_password, mod_data["command_name"],
wrapper_name
    )
    print(f"Wrapper already exists: {wrapper_id}")
except SystemExit:
    print("Wrapper not found")
    outputs, external_inputs, derived_inputs = get_command_id(
        xnat_host, xnat_user, xnat_password, command_id
    )

enable_wrapper_sitewide(xnat_host, command_id, wrapper_name, xnat_user,
xnat_password)
enable_wrapper_for_project(xnat_host, project_id, command_id, wrapper_name,

```

```
xnat_user, xnat_password)
```

```
CONTEXT_ENTITY_MAPPING = {
    "xnat:projectData": {"entity_type": "project", "id_label": "Project ID"},
    "xnat:subjectData": {"entity_type": "subject", "id_label": "Subject ID"},
    "xnat:mrSessionData": {"entity_type": "session", "id_label": "Session ID"},
    "xnat:petSessionData": {"entity_type": "session", "id_label": "Session ID"},
    "xnat:ctSessionData": {"entity_type": "session", "id_label": "Session ID"},
    "xnat:sessionData": {"entity_type": "session", "id_label": "Session ID"},
    "xnat:imageScanData": {"entity_type": "scan", "id_label": "Session ID + Scan
ID"},
    "xnat:resource": {"entity_type": "resource", "id_label": "Resource (not
supported)"}
}
```

```
first_context = mod_data["contexts"][0]
```

```
context_info = CONTEXT_ENTITY_MAPPING.get(first_context)
```

```
if not context_info:
```

```
    print(f"Unknown context: {first_context}. Aborting.")
```

```
    return
```

```
entity_type = context_info["entity_type"]
```

```
# ----- ENTITY CHOOSER -----
```

```
all_files, container_entity_id, scan_id = [], None, None
```

```
if entity_type == "scan":
```

```
    session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
```

```
    if not session_list:
```

```
        print("No sessions found in project.")
```

```
        return
```

```
    chosen_session = select_entity(session_list, id_key="ID", name_key="label")
```

```
    if not chosen_session:
```

```
        print("No session selected.")
```

```
        return
```

```

session_id = chosen_session["ID"]

scan_list = get_scans_for_session(xnat_host, session_id, xnat_user,
xnat_password)
if not scan_list:
    print("No scans found in session.")
    return
# Find a good key for human-readable scan name:
scan_name_key = "series_description" if "series_description" in scan_list[0] else
"ID"
chosen_scan = select_entity(scan_list, id_key="ID",
name_key=scan_name_key)
if not chosen_scan:
    print("No scan selected.")
    return
scan_id = chosen_scan["ID"]
container_entity_id = session_id

all_files = get_input_files(
    xnat_host, session_id, 'scan', xnat_user, xnat_password, scan_id=scan_id
)

elif entity_type == "session":
    session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
    if not session_list:
        print("No sessions found in project.")
        return
    chosen_session = select_entity(session_list, id_key="ID", name_key="label")
    if not chosen_session:
        print("No session selected.")
        return
    session_id = chosen_session["ID"]
    container_entity_id = session_id
    all_files = get_input_files(
        xnat_host, session_id, 'session', xnat_user, xnat_password
    )

```

```

elif entity_type == "subject":
    subject_list = get_subjects_for_project(xnat_host, project_id, xnat_user,
xnat_password)
    if not subject_list:
        print("No subjects found in project.")
        return
    chosen_subject = select_entity(subject_list, id_key="ID", name_key="label")
    if not chosen_subject:
        print("No subject selected.")
        return
    subject_id = chosen_subject["ID"]
    container_entity_id = subject_id
    all_files = get_input_files(
        xnat_host, subject_id, 'subject', xnat_user, xnat_password
    )

```

```

elif entity_type == "project":
    container_entity_id = project_id
    all_files = get_input_files(
        xnat_host, project_id, 'project', xnat_user, xnat_password
    )

```

```

else:
    print("This context is currently not supported.")
    return

```

# ----- FILE CHOOSER -----

```

selected_files = select_files(all_files)

```

```

if selected_files:
    for input_file_info in selected_files:
        launch_container_rest(
            xnat_host,
            project_id,
            command_id,
            wrapper_name,
            container_entity_id,
            xnat_user,

```

```

        xnat_password,
        input_file_info,
        entity_type=entity_type,
        scan_id=scan_id if entity_type == "scan" else None
    )
else:
    print("No file selected. Container launch aborted.")

if __name__ == "__main__":
    main()
...

```

### **\*\*Wichtige Hinweise und wissenschaftliche Begründung:\*\***

- **\*\*Kritische Validierung:\*\*** Early return for failed steps (keine Sessions/Subjects gefunden, keine Auswahl).
- **\*\*Benutzungsfreundlich:\*\*** Geführte Auswahl auf jeder Entity-Ebene, kein "Rätselraten" mit IDs.
- **\*\*Nachvollziehbarkeit:\*\*** Durch ausführliche Hinweise im Fehlerfall, die die Ursachen offenlegen.
- **\*\*Datenstruktur-Navigation:\*\*** Modular und einfach zu testen.
- **\*\*Gute wissenschaftliche Praxis:\*\*** Robustheit, Transparenz, Modularität und Fehlerprävention nach bekannten Software Engineering und Usability Leitlinien.

Falls du Unsicherheiten zu den Hilfsfunktionen hast oder dir Rückgabefelder unklar sind, kannst du einfach deine print-Ausgaben von den REST-Resultaten/Dicts nutzen, um sie zu inspizieren!

### **Bousfiha, Tanae:**

```

tanae@mi0082n:~/dockerfile/xnat-osa/Ziel2_osaAutomatisation$ python3 automat.py
XNAT Username: tanae
XNAT Password:
Project ID: tanae_osa_Predct
Path to the Python script: test.py

```

Wähle einen Kontext:

- 1 : xnat:subjectData,Subject-Ebene (Patientenebene)
- 2 : xnat:mrSessionData,MRI-Session-Ebene (häufigste Session-Ebene)
- 3 : xnat:petSessionData,PET-Session-Ebene
- 4 : xnat:ctSessionData,CT-Session-Ebene
- 5 : xnat:sessionData,Allgemeine Session (für alle Modalitäten)
- 6 : xnat:imageScanData,Scan-Ebene innerhalb einer Session
- 7 : xnat:resource,Resource-Ebene (z.B. für Datei-Uploads direkt)
- 8 : xnat:projectData,Projekt-Ebene

Nummer eingeben: 8

Name des Commands: test

Beschreibung des Commands: test

Dockerfile written to ./Dockerfile

Docker Hub username (to push the image): tanaebousfiha

Building Docker image &#039;test:latest&#039;...

Image &#039;test:latest&#039; built successfully.

Tagging image as &#039;tanaebousfiha/test:latest&#039;...

Pushing image to Docker Hub as &#039;tanaebousfiha/test:latest&#039;...

Image successfully pushed: tanaebousfiha/test:latest

Corrected command.json created.

Uploading command to <https://xnat-dev.gwdg.de/xapi/commands>

Failed to upload command: 401 - &lt;!doctype html&gt;&lt;html

lang=&quot;en&quot;&gt;&lt;head&gt;&lt;title&gt;HTTP Status 401 –

Unauthorized&lt;/title&gt;&lt;style type=&quot;text/css&quot;&gt;body

{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b

{color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3

{font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:

#525D76;border:none;}&lt;/style&gt;&lt;/head&gt;&lt;body&gt;&lt;h1&gt;HTTP Status

401 – Unauthorized&lt;/h1&gt;&lt;hr class=&quot;line&quot;

/&gt;&lt;p&gt;&lt;b&gt;Type&lt;/b&gt; Status

Report&lt;/p&gt;&lt;p&gt;&lt;b&gt;Message&lt;/b&gt; Your login attempt failed because

the username and password combination you provided was invalid or your user

already has the maximum number of user sessions open. After 20 failed login

attempts, your user account will be locked. If you believe your account is currently

locked, you can:&lt;ul&gt;&lt;li&gt;Unlock it by resetting your

password&lt;/li&gt;&lt;li&gt;Wait one hour for it to

unlock automatically&lt;/li&gt;&lt;/ul&gt;&lt;/p&gt;&lt;b&gt;Description&lt;/b&gt; The request has not been applied because

it lacks valid authentication credentials for the target resource.&lt;/p&gt;&lt;hr

class="line" /><h3>Apache Tomcat/9.0.43  
(Debian)</h3></body></html>  
Error fetching commands: 401

## **HAWKI (gpt-4.1):**

Dein Ablauf läuft technisch korrekt, **aber** das Problem ist ein reiner **Authentifizierungsfehler auf XNAT-Seite (HTTP 401 Unauthorized)**. Das hat nichts mit deiner main()-Struktur oder Python-Logik zu tun.

### Was bedeutet das konkret?

- **Dein Username** oder **dein Passwort** für XNAT sind falsch **oder**
- Das Konto ist evtl. gesperrt (siehe die XNAT-Meldung: „your user account will be locked“) **oder**
- Du hast die maximale Zahl paralleler Sessions erreicht **oder**
- Die Authentifizierung ist anders konfiguriert (z./B. via Token nur Username/Passwort)

---

#### Wie kannst du vorgehen?

### 1. **Prüfe Username/Passwort**

- Funktioniert der Login mit denselben Zugangsdaten direkt im Browser auf `https://xnat-dev.gwdg.de/`?`
- Tippfehler? Falls im Skript ein Copy&Paste-Problem mit Leerzeichen oder Zeichenkodierung ist?

### 2. **Maximale User Sessions?**

- Falls du schon viele parallele Logins/Sessions offen hast, schließe die Browser-Tabs/Fenster und warte kurz.
- Oder: Setze gegebenenfalls dein XNAT-Passwort zurück (falls der Account gesperrt wurde!)

### 3. **Authentifizierungsweg**

- Nutzt eure Instanz wirklich „klassischen“ Basic Auth Username/Passwort? Oder brauchst du ggf. ein Session-Ticket, API-Token oder Ähnliches?
- Falls eure XNAT-Instanz SSO/LDAP/OAuth benutzt, rate ich zu Kontakt mit dem Administrator.

#### 4. **\*\*Falls alles nicht hilft\*\***

- Kontaktiere den/die XNAT-Admin/: credentials prüfen lassen  
Account entsperren oder Session-Limit anpassen.

---

### Technisch alles richtig!

Die Python-Seite (dein main()), das Programm- und Entitätenauswahl-Design sind korrekt und können so bleiben.

---

### Wissenschaftliche Begründung:

**\*\*Fehlermeldungen\*\*** und deren klare Interpretation sind zentral für alle wissenschaftlichen Systeme. Ein 401 ist ein standardisiertes Webprotokoll-Signal für Authentifizierungsfehler und kein Programmierbug. Gute Programme geben diese Statuscodes immer direkt aus, genau wie dein Skript.

---

**\*\*Zusammenfassend:\*\***

- Korrigiere/prüfe dein Login.
- Code ist korrekt!
- Systematisch nach obigen Punkten vorgehen.

Gib gern Bescheid, wenn du nach Username/Passwort-Korrektur weitere (programmatische) Probleme siehst!

**Bousfiha, Tanae:**

tanae@mi0082n:~/dockerfile/xnat-osa/Ziel2\_osaAutomatisation\$ python3 automat.py

XNAT Username: tanae

XNAT Password:

Project ID: tanae\_osa\_Predct

Path to the Python script: test.py

Wähle einen Kontext:

1 : xnat:subjectData,Subject-Ebene (Patientenebene)



2 : xnat:mrSessionData,MRI-Session-Ebene (häufigste Session-Ebene)

3 : xnat:petSessionData,PET-Session-Ebene

4 : xnat:ctSessionData,CT-Session-Ebene

5 : xnat:sessionData,Allgemeine Session (für alle Modalitäten)

6 : xnat:imageScanData,Scan-Ebene innerhalb einer Session

7 : xnat:resource,Resource-Ebene (z.B. für Datei-Uploads direkt)

8 : xnat:projectData,Projekt-Ebene

Nummer eingeben: 6

Name des Commands: test

Beschreibung des Commands: test

Dockerfile written to ./Dockerfile

Docker Hub username (to push the image): tanaebousfiha

Building Docker image 'test:latest'...

Image 'test:latest' built successfully.

Tagging image as 'tanaebousfiha/test:latest'...

Pushing image to Docker Hub as 'tanaebousfiha/test:latest'...

Image successfully pushed: tanaebousfiha/test:latest

Corrected command.json created.

Uploading command to https://xnat-dev.gwdg.de/xapi/commands

Failed to upload command: 400 - Bad request:

org.nrg.framework.exceptions.NrgServiceRuntimeException: Error [Unknown]: Error [Unknown]: This command duplicates a command already in the database.

Wrapper already exists: 240

Wrapper 'test\_wrapper' wurde global aktiviert.

Wrapper 'test\_wrapper' wurde im Projekt

'tanae\_osa\_Predct' aktiviert.

Traceback (most recent call last):

File

'/home/tanae/dockerfile/xnat-osa/Ziel2\_osaAutomatisation/automat.py',

line 672, in <module>

main()

File

'/home/tanae/dockerfile/xnat-osa/Ziel2\_osaAutomatisation/automat.py',

line 584, in main

session\_list = get\_sessions\_for\_project(xnat\_host, project\_id, xnat\_user,  
xnat\_password)

NameError: name 'get\_sessions\_for\_project' is not defined

**HAWKI** (gpt-4.1):

Die Fehlermeldung `NameError: name 'get\_sessions\_for\_project' is not defined` bedeutet, dass du die Funktion `**get_sessions_for_project**` nicht (oder nicht im richtigen Scope) definiert hast oder sie von deinem Script nicht gefunden wird.

### So löst du das Problem:

#### 1. `**Definiere die Funktion (oder importiere sie), bevor sie in `main()` verwendet wird.**`

Füge diese Funktion `_vor_` deinem ``main()`` direkt ein:

```
```python
import requests

def get_sessions_for_project(xnat_host, project_id, xnat_user, xnat_password):
    """
    Listet alle Sessions (Experimente) für ein Projekt auf.
    """
    url = f"{xnat_host.rstrip('/')}/data/projects/{project_id}/experiments?format=json"
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"Fehler beim Abrufen der Sessions für Projekt {project_id}:
{resp.status_code}")
    return []
    return resp.json().get("ResultSet", {}).get("Result", [])
```
```

#### 2. `**Genauso für die anderen Helper-Funktionen**`

Je nach Auswahl in deiner `main()` brauchst du wahrscheinlich noch:

```
```python
def get_subjects_for_project(xnat_host, project_id, xnat_user, xnat_password):
    url = f"{xnat_host.rstrip('/')}/data/projects/{project_id}/subjects?format=json"
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"Fehler beim Abrufen der Subjects für Projekt {project_id}:
{resp.status_code}")
    return []
```

```

return resp.json().get("ResultSet", {}).get("Result", [])

def get_scans_for_session(xnat_host, session_id, xnat_user, xnat_password):
    url = f"{xnat_host.rstrip('/')}/data/experiments/{session_id}/scans?format=json"
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"Fehler beim Abrufen der Scans für Session {session_id}:
{resp.status_code}")
        return []
    return resp.json().get("ResultSet", {}).get("Result", [])
...

```

#### 3. **\*\*Speicher und teste dein Skript erneut\*\***

---

**\*\*Wissenschaftliche Begründung:\*\***

Funktionen müssen im Python-Scope bekannt und geladen sein, bevor sie aufgerufen werden können (vgl. grundlegende Prinzipien der funktionalen Programmierung und des Software Engineerings). Modularisierung und Wiederverwendung von Code („helper functions“) erhöhen zudem die Wartbarkeit und Lesbarkeit, was wissenschaftlich und praktisch zentral ist.

---

**\*\*Zusammengefasst:\*\***

Lege die Funktionen wie oben an den Anfang deines Skripts (nach den Imports), damit sie verfügbar sind, wenn deine main() diese aufruft.

> **\*\*Tipp:\*\***

Falls du mit mehreren Modulen arbeitest, stelle sicher, dass du die Funktionen entweder definierst oder aus dem passenden Modul importierst (`from modulname import get\_sessions\_for\_project`).

---

Falls du eine Vorlage für alle benötigten XNAT-Helper-Funktionen möchtest, sag Bescheid!

## Bousfiha, Tanae:

meine Container hat bis jetzt immer eine Datei result erstellt obwohl ich mehrere container gelanched habe

wie kann das skript erweitern dass er mehrere result erstellt und dass die datei result\_nummer oder name der input heißt

## HAWKI (gpt-4.1):

Sehr gute Frage, die sowohl praxisbezogen als auch wissenschaftlich relevant ist.

Dein Ziel:

**\*\*Jeder Containerlauf soll eine eigene Ergebnisdatei erzeugen – zum Beispiel `result\_1.txt`, `result\_2.txt`, `result\_name.txt` oder ähnliches, statt immer nur `result`.\*\*** Das ist essenziell für Nachvollziehbarkeit, Reproduzierbarkeit und parallele Analysen.

### ### Grundproblem

Wahrscheinlich erzeugt dein aktueller Container oder das im Container laufende Skript immer eine Datei mit festem Namen (`result`), unabhängig vom Input. Das überschreibt bei mehreren Durchläufen alte Ergebnisse.

Damit jede Instanz eindeutige Ergebnisse schreibt, kannst du das Script/die Ausführung so modifizieren, dass der **\*\*Zieldateiname dynamisch\*\*** aus z.B. Namen, IDs oder Nummern des Inputs gebildet wird.

---

### ## Lösungsmöglichkeiten

#### ### 1. Namen aus dem Input in den Ausgabedateinamen übernehmen

**\*\*a) Script-Anpassung im Container:\*\***

Passe dein Python-Skript (z.B. `test.py`) im Container an – die Ausgabe soll, je nach übergebenem Input, z.B. `result\_<dateiname>` erstellen:

```
```python
import sys
import os
```

```
# Angenommen, Skript wird so aufgerufen: python test.py inputfile.txt
inputfile = sys.argv[1]
basename = os.path.splitext(os.path.basename(inputfile))[0]
outfile = f"result_{basename}.txt"
```

# Beispielhafte Verarbeitung:

```
with open(inputfile, 'r') as fin, open(outfile, 'w') as fout:
    data = fin.read()
    # Verarbeitung ...
    fout.write(data) # oder das Ergebnis der Verarbeitung
...
```

**\*\*b) Alternativ – Automatisch nummerieren:\*\***

Falls du eine einfache Nummerierung willst (z./B. `result\_1.txt`),  
könntest du beim Starten jedes Containers eine Umgebungsvariable oder ein  
Argument übergeben und dann benutzen:

```
```python
import sys
import os
```

```
idx = sys.argv[2] # z.B. übergib "1", "2", ...
outfile = f"result_{idx}.txt"
...
```

Dazu musst du beim `launch\_container\_rest`-Aufruf das Argument mitgeben.

---

**### 2. Anpassung in `launch\_container\_rest` (Automatisieren am Aufrufer)**

Starte jeden Container so, dass der gewünschte Ausgabedateiname im Argument  
gegeben wird:

**#### Beispiel:**

Du rufst den Container z./B. mit

```
```bash
```

```
docker run ... python test.py input1.txt result_input1.txt
```

```
docker run ... python test.py input2.txt result_input2.txt
```

```
---
```

Das kann über deine Python-Steuerlogik automatisiert werden, z.B. indem du im  
`launch\_container\_rest` für jede Datei den jeweiligen Namen berechnest und mitgibst.

```
---
```

### ### 3. Python: Automatische Namensgebung im Script beim Multi-Start

Wenn du viele Inputs hast, kannst du in Python eine Schleife bauen:

```
```python
inputs = [...]
for input_file in inputs:
    basename = os.path.splitext(os.path.basename(input_file))[0]
    output_file = f"result_{basename}.txt"
    # Container-Start mit input_file und output_file als Argument
```
```

```
---
```

### ## Wissenschaftlicher Hintergrund

- **Reproduzierbarkeit & Nachvollziehbarkeit**: Nach Open-Science-Prinzipien sollen Ergebnisse eindeutig Input und Parametersätzen zuzuordnen sein.
- **Datenintegrität**: Resultate dürfen sich bei Parallelverarbeitung nicht überschreiben.
- **Provenienz** (Datenherkunft): Der Dateiname selbst kodiert bereits einen Teil des Workflows und macht spätere Analysen einfacher.

```
---
```

### ### **Kompakte Zusammenfassung**

1. **Passe dein Skript/test.py im Container an**, damit der Ausgabenname **durch einen Argumentwert** (z.B. `result\_<inputname>.txt`) gesteuert wird.
2. **Übergib beim Starten jedes Containers** (automatisch aus deiner Steuerlogik) das Argument an das Skript.
3. **Optional**: Lege deine Ergebnisse separat nach Inputnummer oder -name ab,

um wissenschaftlich valide Workflows und Resultate sicherzustellen.

---

#### Musteraufruf in Python zur Containersteuerung (Pseudo-Code):

```
```python
for idx, input_file in enumerate(selected_files, 1):
    basename = os.path.splitext(os.path.basename(input_file['name']))[0]
    output_file = f"result_{basename}.txt"
    launch_container_rest(..., input_file, output_arg=output_file)
```
```

und dann im Container:

```
```python
# test.py
import sys
inputfile = sys.argv[1]
outputfile = sys.argv[2]
# ... wie oben
```
```

---

**\*\*Tipp:\*\***

Das Vorgehen ist bestens wissenschaftlich legitimiert (vgl. FAIR Principles; Reproducible Science, Stodden et al., 2016).

Wenn du ein Beispiel für den notwendigen Umbau von `launch\_container\_rest` oder deines Skriptes brauchst, poste ggf. die Schnittstelle/Signatur – ich helfe gern beim Anpassen!

**Bousfiha, Tanae:**

what should i have in my main part ?

**HAWKI (gpt-4.1):**

Excellent question!

**\*\*You want your main() to ensure that each launched container writes its output into a unique file, e.g. `result\_<inputname>.txt` (or similar), instead of overwriting `result`.\*\***  
This is best practice for reproducibility and data traceability.

### Step-by-step for your main()

#### 1. **\*\*Pass the output filename individually for each container launch.\*\***

You loop through `selected\_files`, and for each file:

- Derive a unique output filename (e.g. from the input file name).
- Start the container (via `launch\_container\_rest`), passing the correct input *and* output filename (either as command-line argument or as container variable, whichever your workflow uses).

#### 2. **\*\*What you need in your Python main():\*\***

First, **\*\*your containerized script\*\*** (e.g. `test.py`) must accept the output filename as an argument. If not, you need to modify it as discussed earlier.

**\*\*Example for your main():\*\***

Assume:

- Each entry in `selected\_files` is a dict with `"name"` or similar.
- `launch\_container\_rest` allows passing an output filename (adjust interface if needed).

```
```python
```

```
import os
```

```
def main():
```

```
    # ... your previous code up to selected_files = select_files(all_files)
```

```
    selected_files = select_files(all_files)
```

```
    if selected_files:
```

```
        for input_file_info in selected_files:
```

```
            input_filename = input_file_info["name"] # Adjust if your key is called
differently
```



```
base = os.path.splitext(os.path.basename(input_filename))[0]
output_filename = f"result_{base}.txt"
```

# Launch the container with both the input and output file names as arguments

```
    launch_container_rest(
        xnat_host,
        project_id,
        command_id,
        wrapper_name,
        container_entity_id,
        xnat_user,
        xnat_password,
        input_file_info,
        entity_type=entity_type,
        scan_id=scan_id if entity_type == "scan" else None,
        container_args=[input_filename, output_filename] # This is the key line!
    )
else:
    print("No file selected. Container launch aborted.")
...

```

#### 3. **\*\*Your launch\_container\_rest should look like this:\*\***

It should use the supplied `container\_args` to run something like:

```
```python
docker run ... python test.py input_file output_file
...

```

If `container\_args` does not exist in your function, **\*\*add it\*\***.

---

### **\*\*Checklist for Success\*\***

- Your **\*\*test.py\*\*** in the container must have:

```
```python
import sys
inputfile = sys.argv[1]
outputfile = sys.argv[2]

```

...

and use ``outputfile`` to write its results.

- In your `**main**`, create a unique output filename for each run and pass as argument to the container.
- Adapt `**launch_container_rest**` to accept and use both input and output filenames.

---

### ### `**Scientific Context**`

- `**Data provenance**` and `**reproducibility**` require unique, traceable results for each computational run (see [Stodden et al. 2016, "Enhancing reproducibility for computational methods"](<https://www.nature.com/articles/520437a>)).
- Overwriting outputs masks errors and prevents scientific validation.

---

### `**Summary:**`

Your `main()` must:

- Loop over input files,
- Construct a unique result filename for each,
- Pass both to the container for processing.

`**If you need a worked-out version of `launch_container_rest`, just post it or say so!**`

---

Let me know if you want the code for handling a missing second argument in ``test.py`` (e.g., fallback/default output file for backward compatibility)!

## Bousfiha, Tanae:

```
tanae@mi0082n:~/dockerfile/xnat-osa/Ziel2_osaAutomatisation$ python3 automat.py
```

Traceback (most recent call last):

File

"/home/tanae/dockerfile/xnat-osa/Ziel2\_osaAutomatisation/automat.py",,

line 546, in <module>

inputfile = sys.argv[1]

IndexError: list index out of range

## HAWKI (gpt-4.1):

Der Fehler

...

IndexError: list index out of range

...

bedeutet, dass dein Python-Skript (`test.py`) im Container so aufgerufen wurde, dass  
\*\*kein Argument\*\* übergeben wurde (also `sys.argv[1]` gibt es nicht).

**\*\*Was ist passiert?\*\***

Deine `main()` (besser: die Funktion, die den Container startet) muss sicherstellen,  
dass der Container \_mit\_ mindestens einem Argument (`inputfile.txt`) gestartet wird.  
Wenn du auch ein Ausgabedateinamen-Argument erwartest, müssen **zwei**  
Argumente übergeben werden.

### Was du tun musst

#### 1. Prüfe, wie der Container gestartet wird

Schau im Code von `launch\_container\_rest` (oder wie du den Container startest).

Du solltest da so etwas wie das folgende sehen:

```
python
```

```
docker run ... python test.py inputfile.txt result_inputfile.txt
```

...

Stattdessen ist es offenbar so aufgerufen worden:

```
python
```

```
docker run ... python test.py
```

...

#### #### 2. Korrigiere `launch\_container\_rest`

Du musst die Argumente passend übergeben.

Beispiel-Pseudocode für die Stelle, wo der Container-Run gebaut wird:

```
```python
docker_command = [
    "docker", "run", "--rm", "myimage:latest",
    "python", "test.py", input_filename, output_filename
]
subprocess.run(docker_command, ...)
```
```

Wenn in deinem Aufruf die `container\_args` fehlen oder leer sind, werden Argumente nicht korrekt übergeben.

#### #### 3. Optional: Fehlerabfrage im Skript (`test.py`) verbessern

Du kannst am Anfang deines Containerschritts Folgendes einbauen:

```
```python
import sys
if len(sys.argv) < 3:
    print("Usage: test.py inputfile outputfile")
    sys.exit(1)
inputfile = sys.argv[1]
outputfile = sys.argv[2]
```
```

So bekommst du eine klarere Fehlermeldung.

---

**\*\*Zusammenfassung der empfohlenen Korrekturen:\*\***

- **\*\*In deinem Python-Workflow:\*\*** Stelle sicher, dass beim Containerstart immer alle notwendigen Argumente übergeben werden.
- **\*\*Im Containertestskript (`test.py`):\*\*** Baue obenstehenden Check ein, um Fehlerfälle abzufangen.
- **\*\*Wissenschaftlich:\*\*** Das saubere Übergeben von Ein-/Ausgabedateinamen garantiert Nachvollziehbarkeit und Reproduzierbarkeit, beides grundlegende Anforderungen guter wissenschaftlicher Praxis.

---

**\*\*Gern helfe ich dir, falls du dein `launch\_container\_rest` posten möchtest, wie man die Argumente korrekt durchschleift!\*\***

**Bousfiha, Tanae:**

i think it th eexternal skript is responsible

**HAWKI (gpt-4.1):**

**\*\*You are correct!\*\***

The error comes from your external script (probably `test.py`), which is being run inside the container and expects (for example)

```
```python
inputfile = sys.argv[1]
```
```

but **\*\*no argument is provided\*\***—so the script fails with `IndexError`.

### Why does this happen?

- Your Python (main) script launches containers—each runs your `test.py` (or whatever script is in the container).
- If the container is started WITHOUT command-line arguments (or the arguments are empty), then inside the container, `sys.argv[1]` does not exist.
- This causes `IndexError`.

---

## How to fix this

### **\*\*Check your launcher code\*\***

Wherever you call the container (often in your `launch\_container\_rest` function), you must pass both **\*\*the input and output file name\*\*** as arguments.

**\*\*For example, your launcher should trigger:\*\***

```
```bash
docker run ... python test.py inputfile.txt result_inputfile.txt
```
```

Instead of:

```
```bash
docker run ... python test.py
```
```

```
#### **In Python (using subprocess):**
```python
subprocess.run([
    "docker", "run", "--rm", "your_image",
    "python", "test.py", inputfilename, outputfilename
])
```
```

Make sure that both `inputfilename` and `outputfilename` are real, concrete strings.

---

```
### **Add checks in your external script**
Make your script robust against missing arguments.
```

```
**At the start of `test.py`:**
```python
import sys

if len(sys.argv) < 3:
    print("Usage: test.py inputfile outputfile")
    sys.exit(1)

inputfile = sys.argv[1]
outputfile = sys.argv[2]
```
```

This will make errors user-friendly.

---

```
### **How is this used in your main workflow?**
```

```
**In your main loop:**
```python
for input_file_info in selected_files:
```

```

inputfilename = input_file_info["name"]
basename = os.path.splitext(os.path.basename(inputfilename))[0]
outputfilename = f"result_{basename}.txt"
# Assuming your launch_container_rest can take these as arguments:
launch_container_rest(
    ...,
    container_args=[inputfilename, outputfilename]
)
...

```

---

## ## Summary

- **\*\*You must ensure that when your container is run, both filenames\*\*** (input/output) **\*\*are passed as arguments\*\***.
- **\*\*Your external script should check for missing arguments and print an error.\*\***
- This avoids `IndexError` and guarantees every result is named uniquely!

---

## ### Scientific context

This approach enforces *\*data provenance\** and *\*reproducibility\**, critical elements of scientific workflows (see e.g., FAIR and Open Science principles). It avoids overwriting outputs and ensures traceability of every result to its specific input, which is key for valid scientific data management.

---

**\*\*If you want, post your `launch\_container\_rest` or how you start the containers, and I'll quickly show you the exact correction!\*\***

## Bousfiha, Tanae:

my skript can launch multiples container but uploaded one result file

## HAWKI (gpt-4.1):

This is a classic issue when running multiple containers on different inputs, but seeing

only one result file ("result.txt" or "result"). The problem indicates:

- Either **\*\*your launched containers all write to the same output filename internally\*\*** ("result"),
- Or you are **\*\*not differentiating output file names per launched container\*\*** in your workflow,
- **\*\*OR\*\*** you don't collect/download/output the results under unique names, and so each new result overwrites the previous one.

**\*\*Solution:\*\***

You must (1) give each container instance a unique output filename \_ (e.g. "result\_foo.txt")\_ and (2) make sure your workflow downloads/saves under unique names.

---

### Step 1: Your script inside the container

**\*\*It must accept an output filename as argument\*\***

\_example (test.py in Container)\_

```
```python
import sys
if len(sys.argv) < 3:
    print("Usage: test.py inputfile outputfile")
    sys.exit(1)
inputfile = sys.argv[1]
outputfile = sys.argv[2]
# do stuff and write results to outputfile
with open(inputfile) as fin, open(outputfile, "w") as fout:
    # process...
    fout.write(fin.read())
```
```

**\*This ensures the result is not always called "result".\***

---

### Step 2: Your main workflow (container launcher)



**\*\*Loop over each input, derive a unique result name, and pass it as argument:\*\***

```
```python
import os

for input_file_info in selected_files:
    inputfilename = input_file_info["name"]
    base = os.path.splitext(os.path.basename(inputfilename))[0]
    outputfilename = f"result_{base}.txt"
    launch_container_rest(
        # ... other arguments ...,
        container_args=[inputfilename, outputfilename]
    )
...

```

**\*\*Make sure `launch\_container\_rest` is adapted to pass these args to the container:\*\***

```
```python
docker run ... python test.py inputfilename outputfilename
...

```

or with `subprocess`:

```
```python
subprocess.run([
    "docker", "run", "--rm", "your_image",
    "python", "test.py", inputfilename, outputfilename
])
...

```

---

### ### Step 3: After execution

When you *download* or collect files from your containers or server, save each as the unique output file name you set in step 2. Otherwise, retrieval into a single directory with the same name will again overwrite results.

---

### ### Didactic/Scientific Note

This approach guarantees:

- **Reproducibility**: Each result clearly traced to its input.
- **Integrity**: No overwriting, no lost data.
- **Clarity**: Simpler analyses post hoc; matches good research data management practice (vgl. FAIR & Reproducible Science guidelines).

---

#### **Summary/Checklist:**

- [x] Your script (`test.py` etc.) gets output filename as `*argument*`, writes there.
- [x] Main Python driver generates a unique result filename per container launch.
- [x] Each container gets BOTH input and output filename as argument.
- [x] Download/copy results ALSO keeps those unique result filenames.

If you share your `launch_container_rest` code or describe how results are downloaded, I can show you the exact change there as well!

**Bousfiha, Tanae:**

ok i understnad

**HAWKI (gpt-4.1):**

Das freut mich sehr!

Du hast jetzt die Grundprinzipien für ein robustes, reproduzierbares und wissenschaftlich gutes Container-Workflow-Design verstanden:

1. **Jeder Container bekommt eine individuelle Eingabe & einen individuellen Ausgabedateinamen** (übergib beide explizit als Argumente).
2. **Das Ausführungsskript im Container nimmt den Ausgabedateinamen als Argument und schreibt die Ergebnisse dorthin.**
3. **Die Ergebnisse werden nicht überschrieben, sondern einzigartig benannt und gespeichert.** Das sichert Nachvollziehbarkeit und Datenintegrität.

Das ist „Best Practice“ wissenschaftlicher Datenanalyse (vgl. FAIR, Open Science, Reproduzierbarkeit) und hält auch Software-Engineering-Prinzipien wie Modularität, Separation of Concerns und Transparenz ein.

Wenn du noch Details (z./B. zu `launch\_container\_rest`, Skriptsammlungen etc.) brauchst: Sag gern Bescheid!

Auch zu Usability, automatischer Ergebnis-Sammlung, Erstellen von Logdateien oder Integration mit XNAT unterstütze ich dich gerne weiter.

**\*\*Good scientific practice = traceable, reproducible, robust workflows!\*\***