**Introduction**

**🎯 Objective of Automation Testing**

* Ensure stable, scalable, and reliable validation of Trial Library’s core functionalities (UI, API, and DB).
* Reduce manual testing effort and improve test coverage across clinical trial workflows.
* Support continuous integration (CI) by integrating automated tests into the deployment pipeline.

**📌 Scope**

**Will be Automated:**

* Sponsored trial search via UI and API
* Trial detail validation
* Principal Investigator data
* Pre-screening eligibility checks
* DB assertions (if applicable, e.g., trial or user data stored internally)

**Will NOT be Automated (Initially):**

* Third-party integrations without API access
* One-time setup/config tools
* Visual layout testing (pixel-perfect validations)

**🛠️ Tools & Frameworks**

* **UI**: Playwright (TypeScript)
* **API**: Pytest + Requests (Python)
* **DB**: MySQL Connector + Pytest
* **CI**: Jenkins / GitHub Actions
* **Reporting**: HTML reports (pytest), Allure (optional)

**✅ 2. Test Scope**

**🧪 Functional Areas to Be Automated**

* **UI**: Trial search, dropdown filters (ZIP, radius), trial detail page
* **API**: /mvp/trial-search, validation of response, schema, edge cases
* **DB (if access granted)**: PI contact info, trial UUID validation

**🔁 Types of Tests**

* **Smoke**: Verify basic site and endpoint functionality (status codes, page load, key flows)
* **Regression**: Run before each release to catch feature breakages
* **End-to-End (E2E)**: From trial search > selection > investigator contact
* **Negative Testing**: Invalid ZIPs, missing parameters, CORS headers, etc.
* **Data Validation**: Inclusion/exclusion criteria match between API/UI

**✅ 3. Test Strategy**

**🧱 Framework Design**

* **UI**: Playwright BDD with Cucumber-style Gherkin (optional)
* **API**: Pytest, modular structure with reusable fixtures
* **DB**: Pytest-driven DB validations
* **Reusable utilities** for config, auth headers, schema validation, etc.

**🗃️ Test Data Management**

* Test data stored in .json / .ini / .env or database
* Data-driven approach using parametrize (Pytest) and test scenarios (UI)

**🌐 Environment Setup**

* **Staging**: https://app.triallibrary.com
* **API Endpoint**: https://api.triallibrary.com/mvp/trial-search
* **DB**: TBD (depends on access)

**🔄 CI/CD Integration**

* Automated triggers via Jenkins on:
  + Pull requests
  + Staging deployment
  + Nightly builds
* Test summary reports to Slack or Email

**✅ 4. Tools and Technologies**

| **Category** | **Tool/Library** |
| --- | --- |
| Language | TypeScript (UI), Python (API/DB) |
| UI Automation | Playwright |
| API Automation | Requests, Pytest |
| DB Validation | mysql-connector-python |
| Test Runner | Pytest, npx playwright |
| Reporting | HTML reports, Allure |
| CI/CD | Jenkins, GitHub Actions |
| Version Control | Git (Bitbucket/GitHub) |

**✅ 5. Entry and Exit Criteria**

**🚪 Entry Criteria**

* Feature is stable and available in staging
* UI locators and API contracts are finalized
* Test data is ready

**✅ Exit Criteria**

* All critical test cases have passed
* Bugs in scope are closed or deferred
* Reports are generated and reviewed

**✅ 6. Roles and Responsibilities**

| **Role** | **Responsibility** |
| --- | --- |
| QA Automation Engineer | Writes test cases, scripts, debug failures |
| Manual QA | Provides test data, exploratory scenarios |
| DevOps / QA Lead | CI/CD setup, job triggers, test environments |
| Reviewer | Code reviews for test automation repos |

**✅ 7. Schedule and Milestones**

| **Sprint Week** | **Milestone** |
| --- | --- |
| Week 1 | Framework setup (Playwright, Pytest) |
| Week 2 | Basic trial search API + UI flow automation |
| Week 3 | Add DB checks and test data pipelines |
| Week 4+ | Integrate with Jenkins and nightly builds |

**✅ 8. Risks and Mitigation**

| **Risk** | **Mitigation** |
| --- | --- |
| Dynamic UI locators | Use robust Playwright selectors |
| Test data inconsistency | Use mock/staged data or fixtures |
| CORS/API header restrictions | Add headers via automation |
| Environment downtime | Retry mechanism, run retries in CI |

**✅ 9. Deliverables**

* ✅ Automated test scripts (UI + API + DB)
* ✅ Test reports (HTML/Allure)
* ✅ CI/CD job logs and test results
* ✅ Documentation:
  + Test plan (this document)
  + Test case mapping
  + Setup instructions

C:\Users\Deepika\OneDrive\Desktop\Deepika\trial\_library\.venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2024.3.2/plugins/python-ce/helpers/pycharm/\_jb\_pytest\_runner.py" --path C:\Users\Deepika\OneDrive\Desktop\Deepika\trial\_library\tests\api\test\_trial\_search.py

Testing started at 1:32 PM ...

Launching pytest with arguments C:\Users\Deepika\OneDrive\Desktop\Deepika\trial\_library\tests\api\test\_trial\_search.py --no-header --no-summary -q in C:\Users\Deepika\OneDrive\Desktop\Deepika\trial\_library\tests\api

============================= test session starts =============================

collecting ... collected 15 items

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[60616-50]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[60616-100]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[60616-150]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[60616-250]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[60616-6000]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[10001-50]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[10001-100]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[10001-150]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[10001-250]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[10001-6000]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[94105-50]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[94105-100]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[94105-150]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[94105-250]

test\_trial\_search.py::test\_trial\_search\_by\_zip\_and\_radius[94105-6000]

============================= 15 failed in 4.20s ==============================

FAILED [ 6%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[60616-50])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '60616', radius = 50

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 60616 radius 50

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 13%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[60616-100])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '60616', radius = 100

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 60616 radius 100

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 20%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[60616-150])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '60616', radius = 150

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 60616 radius 150

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 26%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[60616-250])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '60616', radius = 250

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 60616 radius 250

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 33%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[60616-6000])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '60616', radius = 6000

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 60616 radius 6000

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 40%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[10001-50])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '10001', radius = 50

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 10001 radius 50

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 46%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[10001-100])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '10001', radius = 100

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 10001 radius 100

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 53%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[10001-150])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '10001', radius = 150

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 10001 radius 150

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 60%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[10001-250])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '10001', radius = 250

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 10001 radius 250

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 66%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[10001-6000])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '10001', radius = 6000

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 10001 radius 6000

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 73%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[94105-50])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '94105', radius = 50

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 94105 radius 50

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 80%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[94105-100])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '94105', radius = 100

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 94105 radius 100

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 86%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[94105-150])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '94105', radius = 150

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 94105 radius 150

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [ 93%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[94105-250])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '94105', radius = 250

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 94105 radius 250

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

FAILED [100%]

test\_trial\_search.py:30 (test\_trial\_search\_by\_zip\_and\_radius[94105-6000])

403 != 200

Expected :200

Actual :403

<Click to see difference>

zip\_code = '94105', radius = 6000

@pytest.mark.parametrize("zip\_code,radius", test\_cases)

def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):

"""Validate API response for multiple zip codes and radius values"""

params = {

"zip5\_code": zip\_code,

"radius\_in\_miles": radius

}

response = requests.get(BASE\_URL, params=params)

> assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"

E AssertionError: Status code 403 for zip 94105 radius 6000

E assert 403 == 200

E + where 403 = <Response [403]>.status\_code

test\_trial\_search.py:40: AssertionError

Process finished with exit code 1

This test ensures that the structure of the API response matches what the frontend expects — both in terms of required fields and correct data types. This type of schema validation helps catch breaking changes early in the pipeline.

import requests  
import pytest  
from itertools import product  
from utilities.configuration import get\_trial\_search\_base\_url  
from utilities.zipcode import get\_realistic\_zip\_codes  
from utilities.schema\_validator import REQUIRED\_FIELDS  
  
ZIP\_CODES = get\_realistic\_zip\_codes(count=2)  
RADIUS\_VALUES = [50, 100, 150, 250, 6000]  
BASE\_URL = get\_trial\_search\_base\_url()  
  
# Create combinations of zip and radius  
test\_cases = list(product(ZIP\_CODES, RADIUS\_VALUES))  
  
@pytest.mark.positive  
@pytest.mark.parametrize("zip\_code,radius", test\_cases)  
def test\_trial\_search\_by\_zip\_and\_radius(zip\_code, radius):  
 *"""Validate API response for multiple zip codes and radius values"""* params = {  
 "zip5\_code": zip\_code,  
 "radius\_in\_miles": radius  
 }  
 response = requests.get(BASE\_URL, params=params)  
  
 assert response.status\_code == 200, f"Status code {response.status\_code} for zip {zip\_code} radius {radius}"  
  
 data = response.json()  
 assert isinstance(data, list), f"Response is not a list for zip {zip\_code} radius {radius}"  
  
  
@pytest.mark.negative  
@pytest.mark.parametrize("radius", [0, -1, -100, 55, 105, 99999])  
def test\_trial\_search\_invalid\_radius(radius):  
 zip\_code = "60616"  
 response = requests.get(BASE\_URL, params={"zip5\_code": zip\_code, "radius\_in\_miles": radius})  
 assert response.status\_code == 400  
  
@pytest.mark.schema  
@pytest.mark.parametrize("zip\_code,radius", test\_cases)  
def test\_trial\_search\_response\_schema(zip\_code, radius):  
 *"""Test if the response schema matches the expected structure for zip and radius"""* params = {"zip5\_code": zip\_code, "radius\_in\_miles": radius}  
 response = requests.get(BASE\_URL, params=params)  
  
 assert response.status\_code == 200, f"Expected 200, got {response.status\_code} for {zip\_code=} {radius=}"  
  
 data = response.json()  
 assert isinstance(data, list), f"Response should be a list for {zip\_code=} {radius=}"  
  
 for trial in data:  
 assert isinstance(trial, dict), "Each trial must be a dictionary"  
 for field, expected\_type in REQUIRED\_FIELDS.items():  
 assert field in trial, f"Missing field: {field}"  
 assert isinstance(trial[field], expected\_type), (  
 f"Field '{field}' has type {type(trial[field])}, expected {expected\_type}"  
 )  
  
@pytest.mark.schema  
@pytest.mark.positive  
@pytest.mark.parametrize("zip\_code,radius", test\_cases)  
def test\_required\_field\_values\_are\_valid(zip\_code, radius):  
 params = {  
 "zip5\_code": zip\_code,  
 "radius\_in\_miles": radius  
 }  
 response = requests.get(BASE\_URL, params=params)  
 assert response.status\_code == 200, f"Status {response.status\_code} for {zip\_code=} {radius=}"  
  
 data = response.json()  
 assert isinstance(data, list), f"Expected list, got {type(data)}"  
  
 for trial in data:  
 # Trial name must not be empty  
 assert trial.get("sponsored\_trial\_name"), f"Missing name for {zip\_code=} {radius=}"  
  
 # URL must start with http/https  
 url = trial.get("sponsored\_trial\_study\_url", "")  
 assert url.startswith("http"), f"Invalid URL '{url}' for {zip\_code=} {radius=}"  
  
 # Phase must be a valid value  
 assert trial.get("sponsored\_trial\_phase") in ["1", "1/2", "2", "2b", "3", "N/A"], \  
 f"Unexpected trial phase for {zip\_code=} {radius=}"  
  
 # Distance should not exceed radius  
 distance = trial.get("distance\_to\_closest\_location\_in\_miles")  
 assert distance is not None and distance <= radius, \  
 f"Distance {distance} > radius {radius} for {zip\_code=}"  
  
 # Investigator name checks  
 assert trial.get("closest\_principal\_investigator\_first\_name"), "Missing PI first name"  
 assert trial.get("closest\_principal\_investigator\_last\_name"), "Missing PI last name"  
  
 # Location name must exist  
 assert trial.get("closest\_sponsored\_trial\_location\_name"), "Missing trial location name"

run test

pytest -m negative

|  |
| --- |
|  |

|  |
| --- |
| pytest -m schema |

pytest -v

pytest --html=reports/trial\_report.html --self-contained-html

pytest tests/api/test\_trial\_search.py