RiskLens AI - Selenium UI Testing Suite

This folder contains automated UI tests written with Selenium WebDriver (Python) to validate core user journeys in the RiskLens AI web application.

What these tests cover

- Authentication flows: user signup and login, admin login
- Finance Risk Assessment: complete and submit form, verify result displayed, save to DB
- Health Risk Assessment: complete and submit form, verify result displayed, save to DB

How Selenium works here (high level)

- Tests use Selenium WebDriver with a headless Chrome browser.
- Each test launches the browser, navigates to `http://localhost:3000`, and performs real user interactions (clicks, typing, form submissions).
- We use explicit waits to ensure UI elements are present before interacting.
- Selectors rely on accessible attributes (placeholder, label text, role) and stable CSS where possible.
- Assertions check page navigation, presence of success messages, and key UI text.

Prerequisites

- Python 3.10+
- Chrome browser installed

Install dependencies

```
cd testing/selenium
python3 -m venv .venv
source .venv/bin/activate  # Windows: .venv\\Scripts\\activate
pip install -r requirements.txt
```

Start the application under test

In a separate terminal from the project root:

```
npm run dev
# App should be at http://localhost:3000
```

Run tests

```
cd testing/selenium
source .venv/bin/activate
pytest -v
```

Or use the helper script (Unix/macOS):

```
bash run_tests.sh
```

Test files

- `tests/test_auth_flow.py` : Signup + login; admin login
- `tests/test_finance_assessment.py` : Fill & submit finance form; verify result

- `tests/test_health_assessment.py` : Fill & submit health form; verify result
- `tests/utils.py` : Browser setup, helpers (waits, navigation, login)

Notes

- These tests assume the app runs on `http://localhost:3000` .
- External ML/RAG endpoints are mocked by checking for UI success states rather than remote responses. If your remote services are live, the same tests will also validate end-to-end behavior.

Appendix: Why Selenium for demo/testing

- Demonstrates realistic end-user behavior
- Validates core happy-path flows quickly for presentations
- Produces repeatable results for CI/CD integration later