**Overview**

**Application Type:** Desktop GUI application for pesticide Environmental Impact Quotient (EIQ) calculation and season planning

**Target Users:** Agronomists managing potato cultivation programs

**Core Purpose:** Evaluate, compare, and plan pesticide applications with a focus on environmental impact assessment using EIQ scores

**Directories Structure (see end of document for the full tree)**

PROJECT/ # Root

├── main.py # Application entry point

├── user\_preferences.json # User settings (country, region, seeding rate, row spacing)

│

├── common/ # Shared components, styles, constants

│ ├── calculations/ # EIQ calculation engine

│ └── widgets/ # Common UI components

│

├── data/ # Data layer

│

├── main\_page/ # Main window and home page

│

├── products\_page/ # Control product view and comparison features

│

├── eiq\_calculator\_page/ # EIQ calculation features

│

├── season\_planner\_page/ # Season planning features

│ ├── delegates/ # Table cell editors

│ ├── import\_export/ # Excel I/O functionality

│ ├── models/ # Data models for season planning

│ └── widgets/ # Season planner UI components

│

└── user\_manual/ # User manual

**Main directories and files (see attached html user manual for the screenshots):**

* **main.py:** Application entry point that initializes repositories, applies configurations, and launches the main window
* **user\_preferences.json:** Persistent storage for user settings including country, region, and default parameters (seeding rate, row spacing, and their units of measure)
* **common:** Shared components, styles, utilities, and calculations used across the entire application
* **data:** Data models, repositories, and CSV files containing product, active ingredient, and unit of measure information
* **main\_page:** Application entry point with home page, main window container, and user preferences management
* **products\_page:** Product listing, filtering, and side-by-side comparison functionality with fact sheets
* **eiq\_calculator\_page:** Environmental Impact Quotient calculator with single product and multi-product comparison tabs
* **season\_planner:** Season planning tools for creating, managing, and comparing pesticide application scenarios
* **user\_manual:** HTML-based user documentation with screenshots, features and “how-to-use” explanations

**Technical specifications**

* **Python Version:** 3.12.7
* **GUI Framework:** PySide6 v6.9.0
* **Target OS:** Windows 10/11
* **Architecture:** Fully offline, Single-threaded Qt application with event-driven UI
* **Primary Libraries:**

|  |  |
| --- | --- |
| **GUI Framework** | PySide6 (Qt GUI framework)  PySide6.QtCore  PySide6.QtWidgets  PySide6.QtGui  PySide6.QtPrintSupport |
| **Excel Files Handling** | openpyxl |
| **Standard Python** | json (JSON file handling)  csv (CSV file handling)  datetime (date/time operations)  typing (type hints)  dataclasses (data classes)  collections (specialized container datatypes)  traceback (error handling)  pathlib (file path operations)  os (operating system interface)  sys (system-specific parameters)  io (input/output operations)  itertools (iterator functions)  functools (functional programming tools)  operator (operator functions)  re (regular expressions)  webbrowser (web browser launching)  tempfile (temporary files)  shutil (file operations) |

**Current Functionality**

* **User workflow steps:**

1. Start on Home page - Set preferences that persist between sessions
2. Navigate using buttons - Click navigation buttons to access products page, EIQ calculator page, or season planner page
3. Products page - Filter/search products, select for comparison, view detailed fact sheets
4. EIQ Calculator - Choose single or multiple product tabs, search products, set application parameters, view calculated results
5. Season Planner - Create scenarios via tabs, add applications in table format, import/export scenarios, compare multiple scenarios

* **Excel import process:**

1. User clicks "Import Scenario" button in Season Planner
2. File dialog opens to select Excel files (.xlsx/.xls)
3. Parser detects format type (exported by app vs. external GXCore reports)
4. Shows preview with scenario metadata and application count
5. If unmatched products found, opens mapping tab to resolve (skip, import as-is, or map to existing products)
6. User reviews and approves mappings or other resolutions
7. Final scenario created and added as new tab

* **Excel export process:**

1. User clicks "Export" button in Season Planner scenarios page
2. File dialog opens to choose save location
3. Each scenario tab becomes a separate worksheet in the Excel file
4. Format includes: scenario metadata row, field information row, header row, then application data
5. File saved with user-chosen filename and location
6. Success message shows file path

* **Pesticide data source:**

Currently using a local version of GXCore obtained through a standard GXCore report, will need to be properly connected to GXCore/Azure to have the updated data once integrated in the systems.

Location: data directory

Files:

* csv\_products.csv - Control products database (control product standards from GXCore)
* csv\_AI.csv - Active ingredients with chemical MoA group and EIQ values
* csv\_UOM.csv - Units of measure for conversions

Format: CSV files loaded into memory via repository classes

Access: Data loaded at startup through ProductRepository and AIRepository

**Technical Details**

* **Excel files handling:**
  + Library: Uses openpyxl for both reading and writing Excel files
  + Format support: Handles both .xlsx and .xls files through file dialogs
  + Parser architecture: ExcelScenarioParser with dual format detection (exported vs external files)
  + Export functionality: ExcelScenarioExporter creates multi-worksheet Excel files with formatting
* **Where files get saved:**
  + User choice: File dialogs allow users to select save location and filename via QFileDialog.getSaveFileName
  + Default extension: Automatically appends .xlsx if not provided
  + Import location: Users browse and select Excel files from any location
  + Success feedback: Shows full file path in success message after export
* **Data storage:**
  + In-memory: All data stored in memory using Qt model/view architecture
  + No database: Uses CSV files for reference data (csv\_products.csv, csv\_AI.csv, csv\_UOM.csv)
  + Repository pattern: ProductRepository and AIRepository cache CSV data at startup
  + Session persistence: No automatic saving - data exists only during application session
* **Current structure:**
  + Feature-based directories, modular organization (season\_planner\_page, products\_page, etc.)
  + Shared components: common directory for styles, utilities, widgets
  + Data layer: data directory with models, repositories, and CSV files
  + Import/Export: Dedicated import\_export/ package with parser and exporter classes
* **Configuration files:**
  + Modular organization: Feature-based packages (season\_planner\_page, products\_page, etc.)
  + MVC pattern: Models (application\_table\_model.py), Views (Qt widgets), Controllers (delegates)
  + Shared components: common directory for styles, utilities, widgets
  + Data layer: data directory with models, repositories, and CSV files
  + Import/Export: Dedicated import\_export/ package with parser and exporter classes

**Current Limitations/Issues**

* **Known bugs or problems:**
  + When converting UOM of seed treatment and row length applications: physical compatibility check is not sound, conversion between normal application rates UOM (volume or weight/area) to special (volume or weight/length, volume or weight/weight) is always possible even when it shouldn't
  + Scenario comparison page still looks basic - UI polishing incomplete
  + Finding products by name using ProductSearchField widget: if you select a product and then delete the text, you have to press "space" once to see the full list of suggestions again
  + Excel import validation: Some application validation issues require manual correction in the applications table afterward (not really a bug, this is how it works: e.g. if an imported application has app.rate of 0.0, it needs to be set manually later)
* **Features wanted but not built yet:**
  + Products filtered by pest controlled
  + Scenario comparison to regional average 25% and 75% EIQ range
  + Suggestions for alternative similar products with lower EIQs
* **Important limitations:**
  + No auto-save functionality yet, users must export scenarios to preserve work between sessions
  + Static CSV-based data loaded at startup for now in the future this should become a cache-and-sync application once connected to GXCore

**Business Logic**

* **Pesticide filtering criteria:** by any characteristic the user wants, the filters work as “contains” not “begins with”
* **Spray plan structure:** A series of applications with metadata (crop year, grower, field, area, variety) and validation states
* **Data validation:**
  + Products must belong in the database
  + Physical compatibility checking: Prevents conversions between incompatible physical states (liquid↔dry without density data)
  + Required field validation: Product name, rate, and rate UOM must be provided
  + Range validation: Rates and areas cannot be negative; extremely high values trigger warnings
* **EIQ calculation logic:**
  + Estimation system: When active ingredient EIQ data is missing, uses average of valid applications or 50.0 default if errors occur (method agreed with Global)
  + Multi-AI products: Handles products with up to 4 active ingredients, calculating combined EIQ impact as the sum of each active ingredient
  + Unit standardization: All calculations performed in standardized units (kg/ha or l/ha) before EIQ computation
* **Output format:** scenario metadata row, field information row, header row, then application data

**Project Structure (full tree)**

PROJECT/

├── main.py # Application entry point

├── icon.ico # Application icon

├── user\_preferences.json # User settings (country, region, defaults)

├── PesticidesApp.spec # PyInstaller configuration

│

├── common/ # Shared components

│ ├── constants.py # Colors, fonts, EIQ thresholds

│ ├── styles.py # CSS stylesheets

│ ├── utils.py # Configuration utilities

│ ├── calculations/ # EIQ calculation engine

│ │ ├── layer\_1\_interface.py # API interface

│ │ ├── layer\_2\_uom\_std.py # Unit standardization

│ │ └── layer\_3\_eiq\_math.py # EIQ mathematical calculations

│ └── widgets/ # Reusable UI components

│ ├── application\_params.py # Application parameter widget

│ ├── header\_frame\_buttons.py # Navigation buttons

│ ├── product\_selection.py # Product search widget

│ ├── scorebar.py # EIQ score visualization

│ ├── tracer.py # Calculation logging

│ └── UOM\_selector.py # Unit selection dialog

│

├── data/ # Data layer

│ ├── csv\_AI.csv # Active ingredients database

│ ├── csv\_products.csv # Product database

│ ├── csv\_UOM.csv # Unit conversion data

│ ├── converter\_UOM.py # Unit conversion functions

│ ├── model\_\*.py # Data models (AI, Application, Product, Scenario)

│ └── repository\_\*.py # Data access layer

│

├── main\_page/ # Home and main window

│ ├── window\_main.py # Main application window

│ ├── page\_home.py # Home page interface

│ ├── widget\_preferences\_row.py # User preferences widget

│ ├── logo\_McCain.png # Brand logos

│ └── logo\_NAAg.png

│

├── products\_page/ # Product management features

│ ├── page\_products.py # Products container page

│ ├── tab\_products\_list.py # Product listing tab

│ ├── tab\_products\_comparison.py # Product comparison tab

│ ├── widget\_products\_table.py # Main products table

│ ├── widget\_comparison\_table.py # Comparison table

│ └── widget\_filter\_row.py # Table filtering controls

│

├── eiq\_calculator\_page/ # EIQ calculation features

│ ├── page\_eiq\_calculator.py # Calculator container page

│ ├── tab\_single\_calculator.py # Single product calculator

│ ├── tab\_multi\_calculator.py # Multi-product calculator

│ ├── widget\_product\_card.py # Product representation widget

│ └── widgets\_results\_display.py # Results visualization

│

├── season\_planner\_page/ # Season planning features

│ ├── page\_scenarios\_manager.py # Scenario management

│ ├── page\_scenarios\_comparison.py # Scenario comparison

│ ├── tab\_scenario.py # Individual scenario tab

│ ├── delegates/ # Table cell editors

│ │ ├── date\_delegate.py # Date column editor

│ │ ├── method\_delegate.py # Application method editor

│ │ ├── numeric\_delegate.py # Numeric value editor

│ │ ├── product\_name\_delegate.py # Product selection editor

│ │ ├── product\_type\_delegate.py # Product type editor

│ │ ├── reorder\_delegate.py # Row reordering controls

│ │ └── uom\_delegate.py # Unit selection editor

│ ├── import\_export/ # Excel I/O functionality

│ │ ├── import\_dialog.py # Import interface

│ │ ├── excel\_parser.py # Excel file parser

│ │ └── exporter.py # Excel export engine

│ ├── models/ # Data models for season planning

│ │ ├── application\_table\_model.py # Qt table model

│ │ ├── application\_eiq\_calculator.py # EIQ calculation for apps

│ │ └── application\_validator.py # Data validation

│ └── widgets/ # Season planner UI components

│ ├── applications\_table.py # Main applications table

│ ├── eiq\_summary.py # EIQ summary display

│ ├── metadata\_widget.py # Scenario metadata

│ └── scenario\_comparison\_table.py # Scenario comparison

│

└── user\_manual/ # Documentation

├── user\_manual.html # HTML user guide

├── open\_user\_manual.py # Manual launcher

└── \*.png # Screenshot assets