Polyglot Language Support for minard-loader

Overview

The minard-loader is a Rust application that loads PureScript project metadata into a DuckDB database for visualization. Currently, it tracks packages, modules, declarations, and dependencies - but treats all projects as if they target JavaScript.

The PSD3 ecosystem is genuinely polyglot:

- **purerl** (Erlang) - psd3-tilted-radio, tidal parser
- **purepy** (Python) - hypo-punter backends (ee-server, ge-server)
- **pslua** (Lua) - scuppered-ligature edge router
- **Standard JS** - most libraries and showcases

This plan extends the loader to track which backend each package uses, enabling visualizations that show the polyglot nature of the ecosystem.

Current State Analysis

Loader Architecture

The minard-loader follows a pipeline architecture:

```
ProjectDiscovery → SpagoLock Parsing → Package Creation → Module Parsing →
Database Insert
```

Key files:

- `src/loader/discovery.rs` - Finds projects via `spago.lock` + `output/` directory
- `src/parse/spago_lock.rs` - Parses spago.lock for package metadata
- `src/loader/pipeline.rs` - Main load pipeline, parses docs.json files
- `src/db/schema.rs` - DuckDB schema definition
- `src/model/entities.rs` - Rust data structures

Current Database Schema

The schema (v3.0) has these relevant tables:

```sql
-- Package versions table (needs extension)
CREATE TABLE package_versions (
    id                INTEGER PRIMARY KEY,
    name              VARCHAR NOT NULL,
    version           VARCHAR NOT NULL,
    description       TEXT,
    license           VARCHAR,
    repository        VARCHAR,
    source          VARCHAR DEFAULT 'registry',  -- "registry" | "local" |
"git"
    created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(name, version)
);

-- Modules table
CREATE TABLE modules (
    id                  INTEGER PRIMARY KEY,
    package_version_id  INTEGER NOT NULL REFERENCES package_versions(id),
    namespace_id        INTEGER REFERENCES module_namespaces(id),
    name                VARCHAR NOT NULL,
    path                VARCHAR,
    comments            TEXT,
    loc                 INTEGER,  -- PureScript LOC only
    UNIQUE(package_version_id, name)
);
```

LOC Calculation

Currently, LOC is computed from `docs.json` or `corefn.json` by finding the maximum end line across declarations. This gives PureScript LOC only - it doesn't count FFI files.

```
// From pipeline.rs
let corefn_path = docs_path.with_file_name("corefn.json");
let loc = if corefn_path.exists() {
    CoreFn::from_path(&corefn_path)
        .ok()
        .and_then(|cf| cf.compute_loc())
        .or_else(|| docs.compute_loc())
} else {
    docs.compute_loc()
};
```

Backend Detection Strategy

Detection Heuristics

Each backend has distinct markers. Detection should happen at the **project** level (not package level) since the spago.yaml determines the backend.

1. Purerl (Erlang)

Primary indicators:

- `spago.yaml` contains `backend.cmd` pointing to `purerl`
- `spago.yaml` references purerl package set URL: `https://raw.githubusercontent.com/purerl/package-sets/...`
- Dependencies include `erl-*` packages (erl-cowboy, erl-kernel, erl-process, etc.)

Secondary indicators:

- Presence of `.erl` files in `output/*/` directories (the `*@foreign.erl` pattern)
- Presence of `rebar.config` or `rebar3` structure (`_build/` directory)

Example from purerl-tidal/spago.yaml:

```
workspace:
  packageSet:
    url: https://raw.githubusercontent.com/purerl/package-sets/erl-0.15.3-
20220629/packages.json
  backend:
    cmd: "/Users/afc/bin/purerl"
```

2. PurePy (Python)

Primary indicators:

- Presence of `output-py/` directory (transpiled Python output)
- Python FFI files matching pattern `*_foreign.py` in `output-py/`
- Python source FFI files in `src/**/*.py` (e.g., `src/Data/UMAP.py`)

Secondary indicators:

- `requirements.txt` in project root
- `run.py` entry point
- Dockerfile with Python base image

Note: PurePy doesn't use spago.yaml backend configuration - it's a post-processing step that transforms `output/` to `output-py/`.

3. PsLua (Lua)

Primary indicators:

- `spago.yaml` comments mention `pslua` or `purescript-lua`
- Dependencies from `github.com/Unisay/purescript-lua-*` packages
- `lua-ffi/` directory with Lua FFI implementations

Secondary indicators:

- Presence of `.lua` files in output or FFI directories
- Reference to OpenResty/Nginx in README or Dockerfile

Example from scuppered-ligature/spago.yaml:

```
# Uses pslua backend
# See: https://github.com/Unisay/purescript-lua
```

```
workspace:
  extraPackages:
    effect:
      git: https://github.com/Unisay/purescript-lua-effect.git
      ref: v4.1.0
```

4. Standard JavaScript (default)

If none of the above indicators are present, assume JavaScript target.

Detection Algorithm

```
#[derive(Debug, Clone, Copy, PartialEq, Eq)]
pub enum Backend {
    JavaScript,  // Default
    Erlang,      // purerl
    Python,      // purepy
    Lua,         // pslua
}

pub struct BackendDetection {
    pub primary: Backend,
    pub has_ffi: HashMap<Backend, bool>,  // Which FFI types are present
}

impl BackendDetection {
    pub fn detect(project_path: &Path, spago_yaml: &str) -> Self {
        // 1. Check spago.yaml for explicit backend config
        // 2. Check for backend-specific package set URLs
        // 3. Check for erl-* dependencies
        // 4. Check for purescript-lua-* dependencies
        // 5. Check filesystem for output-py/, lua-ffi/, etc.
        // 6. Default to JavaScript
    }
}
```

Schema Design

Option A: Simple Column Addition (Recommended)

Add columns to `package_versions` table:

```
ALTER TABLE package_versions ADD COLUMN backend VARCHAR DEFAULT 'js';
-- Values: 'js' | 'erlang' | 'python' | 'lua'

ALTER TABLE package_versions ADD COLUMN loc_js INTEGER;     -- JavaScript
FFI LOC
ALTER TABLE package_versions ADD COLUMN loc_erlang INTEGER; -- Erlang FFI
LOC
```

```
ALTER TABLE package_versions ADD COLUMN loc_python INTEGER; -- Python FFI
LOC
ALTER TABLE package_versions ADD COLUMN loc_lua INTEGER;    -- Lua FFI LOC
ALTER TABLE package_versions ADD COLUMN ffi_file_count INTEGER DEFAULT 0;
```

Pros: Simple, backwards compatible, easy queries **Cons:** Some columns unused for most packages

Option B: Separate Language Statistics Table

```
CREATE TABLE package_language_stats (
    package_version_id  INTEGER PRIMARY KEY REFERENCES
package_versions(id),
    primary_backend     VARCHAR NOT NULL DEFAULT 'js',
    loc_purescript      INTEGER DEFAULT 0,
    loc_javascript      INTEGER DEFAULT 0,
    loc_erlang          INTEGER DEFAULT 0,
    loc_python          INTEGER DEFAULT 0,
    loc_lua             INTEGER DEFAULT 0,
    ffi_file_count      INTEGER DEFAULT 0,
    has_erlang_ffi      BOOLEAN DEFAULT FALSE,
    has_python_ffi      BOOLEAN DEFAULT FALSE,
    has_lua_ffi         BOOLEAN DEFAULT FALSE
);
```

Pros: Cleaner separation, extensible **Cons:** Requires joins for common queries

Option C: Project-Level Tracking (Hybrid)

Since backend is determined at the project/workspace level, not per-package:

```
-- Add to projects table
ALTER TABLE projects ADD COLUMN primary_backend VARCHAR DEFAULT 'js';
ALTER TABLE projects ADD COLUMN backends JSON;  -- ["js", "python"] for
hybrid projects

-- Package-level FFI stats (unchanged from Option A)
ALTER TABLE package_versions ADD COLUMN loc_ffi_js INTEGER;
ALTER TABLE package_versions ADD COLUMN loc_ffi_erlang INTEGER;
ALTER TABLE package_versions ADD COLUMN loc_ffi_python INTEGER;
ALTER TABLE package_versions ADD COLUMN loc_ffi_lua INTEGER;
```

Recommendation: Option C is most accurate to the actual structure. A project targets one primary backend, but packages can have FFI for multiple backends.

Proposed Final Schema Changes

```sql
-- Schema version bump: 3.0 → 3.1

-- Project-level backend tracking
ALTER TABLE projects ADD COLUMN primary_backend VARCHAR DEFAULT 'js';

-- Package-level FFI statistics
ALTER TABLE package_versions ADD COLUMN loc_purescript INTEGER;  -- Rename from loc
ALTER TABLE package_versions ADD COLUMN loc_ffi_js INTEGER DEFAULT 0;
ALTER TABLE package_versions ADD COLUMN loc_ffi_erlang INTEGER DEFAULT 0;
ALTER TABLE package_versions ADD COLUMN loc_ffi_python INTEGER DEFAULT 0;
ALTER TABLE package_versions ADD COLUMN loc_ffi_lua INTEGER DEFAULT 0;
ALTER TABLE package_versions ADD COLUMN ffi_file_count INTEGER DEFAULT 0;

-- New view for polyglot summary
CREATE OR REPLACE VIEW polyglot_summary AS
SELECT
    p.name AS project_name,
    p.primary_backend,
    COUNT(DISTINCT pv.id) AS package_count,
    SUM(pv.loc_purescript) AS total_loc_purescript,
    SUM(pv.loc_ffi_js) AS total_loc_js,
    SUM(pv.loc_ffi_erlang) AS total_loc_erlang,
    SUM(pv.loc_ffi_python) AS total_loc_python,
    SUM(pv.loc_ffi_lua) AS total_loc_lua,
    SUM(pv.ffi_file_count) AS total_ffi_files
FROM projects p
JOIN snapshots s ON s.project_id = p.id
JOIN snapshot_packages sp ON sp.snapshot_id = s.id
JOIN package_versions pv ON sp.package_version_id = pv.id
GROUP BY p.id, p.name, p.primary_backend;
```

Implementation Steps

Phase 1: Schema Updates (Rust changes)

1. **Update `src/db/schema.rs`**
    - Bump schema version to 3.1
    - Add new columns to CREATE TABLE statements
    - Add migration for existing databases
2. **Update `src/model/entities.rs`**
    - Add `Backend` enum
    - Add FFI LOC fields to `PackageVersion` struct
    - Add `primary_backend` to `Project` struct
3. **Update `src/db/insert.rs`**
    - Handle new columns in INSERT statements

Phase 2: Backend Detection (New module)

1. **Create `src/detect/mod.rs`**

   - `BackendDetector` struct
   - `detect_from_spago_yaml()` function
   - `detect_from_filesystem()` function

2. **Create `src/detect/spago_yaml.rs`**

   - Parse spago.yaml (not spago.lock)
   - Extract backend configuration
   - Check for backend-specific dependencies

3. **Create `src/detect/ffi_scan.rs`**

   - Scan for `*.erl`, `*@foreign.erl` files
   - Scan for `*_foreign.py` files
   - Scan for `*.lua` FFI files
   - Count files and compute LOC

Phase 3: LOC Calculation Enhancement

1. **Update `src/loader/pipeline.rs`**

   - Add FFI file scanning during load
   - Separate PureScript LOC from FFI LOC

2. **Create LOC counting functions**

```
fn count_loc_erlang(path: &Path) -> Option<i32> { ... }
fn count_loc_python(path: &Path) -> Option<i32> { ... }
fn count_loc_lua(path: &Path) -> Option<i32> { ... }
fn count_loc_javascript(path: &Path) -> Option<i32> { ... }
```

3. **FFI file patterns by backend:**

   - **Erlang:** `output/ModuleName/*@foreign.erl`
   - **Python:** `output-py/*_foreign.py` or `src/**/*.py`
   - **Lua:** `lua-ffi/**/*.lua`
   - **JavaScript:** `src/**/*.js` (standard FFI)

Phase 4: Discovery Enhancement

1. **Update `src/loader/discovery.rs`**

   - Add spago.yaml path to `ProjectDiscovery`
   - Store detected backend

2. **Update `ProjectDiscovery` struct:**

```
pub struct ProjectDiscovery {
    pub project_path: PathBuf,
    pub spago_yaml_path: Option<PathBuf>,  // NEW
    pub spago_lock_path: PathBuf,
    pub output_dir: PathBuf,
    pub docs_json_files: Vec<PathBuf>,
    pub relative_name: Option<String>,
    pub detected_backend: Backend,          // NEW
}
```

Phase 5: Views and Queries

Add analytical views to `src/db/schema.rs`:

```
-- Backend distribution across projects
CREATE OR REPLACE VIEW backend_distribution AS
SELECT
    primary_backend,
    COUNT(*) AS project_count,
    SUM(total_loc_purescript) AS total_purescript_loc
FROM polyglot_summary
GROUP BY primary_backend;

-- FFI usage by package
CREATE OR REPLACE VIEW ffi_usage AS
SELECT
    pv.name,
    pv.version,
    pv.ffi_file_count,
    pv.loc_ffi_js,
    pv.loc_ffi_erlang,
    pv.loc_ffi_python,
    pv.loc_ffi_lua,
    CASE
        WHEN pv.loc_ffi_erlang > 0 THEN 'erlang'
        WHEN pv.loc_ffi_python > 0 THEN 'python'
        WHEN pv.loc_ffi_lua > 0 THEN 'lua'
        WHEN pv.loc_ffi_js > 0 THEN 'js'
        ELSE 'none'
    END AS ffi_language
FROM package_versions pv
WHERE pv.ffi_file_count > 0;
```

Example Queries for Visualization

1. Polyglot Ecosystem Overview (Sunburst/Treemap)

```sql
-- For a sunburst showing Backend → Project → Package
SELECT
    p.primary_backend AS backend,
    p.name AS project_name,
    pv.name AS package_name,
    pv.loc_purescript
FROM projects p
JOIN snapshots s ON s.project_id = p.id
JOIN snapshot_packages sp ON sp.snapshot_id = s.id
JOIN package_versions pv ON sp.package_version_id = pv.id
WHERE sp.source = 'workspace'
ORDER BY p.primary_backend, p.name, pv.name;
```

2. LOC by Language (Stacked Bar)

```sql
-- Total LOC breakdown by language across all projects
SELECT
    primary_backend,
    SUM(total_loc_purescript) AS purescript,
    SUM(total_loc_js) AS javascript_ffi,
    SUM(total_loc_erlang) AS erlang_ffi,
    SUM(total_loc_python) AS python_ffi,
    SUM(total_loc_lua) AS lua_ffi
FROM polyglot_summary
GROUP BY primary_backend;
```

3. FFI Density (Scatter Plot)

```sql
-- FFI LOC vs PureScript LOC per package
SELECT
    pv.name,
    pv.loc_purescript,
    (COALESCE(pv.loc_ffi_js, 0) +
     COALESCE(pv.loc_ffi_erlang, 0) +
     COALESCE(pv.loc_ffi_python, 0) +
     COALESCE(pv.loc_ffi_lua, 0)) AS total_ffi_loc,
    pv.ffi_file_count
FROM package_versions pv
WHERE pv.loc_purescript > 0;
```

4. Backend-Specific Dependency Graph

```sql
-- Packages in Erlang projects and their dependencies
WITH erlang_packages AS (
    SELECT DISTINCT pv.id, pv.name
    FROM projects p
    JOIN snapshots s ON s.project_id = p.id
    JOIN snapshot_packages sp ON sp.snapshot_id = s.id
    JOIN package_versions pv ON sp.package_version_id = pv.id
    WHERE p.primary_backend = 'erlang'
)
SELECT
    ep.name AS package,
    pd.dependency_name AS depends_on
FROM erlang_packages ep
JOIN package_dependencies pd ON pd.dependent_id = ep.id;
```

5. Cross-Language Type Mapping Analysis

```
-- Find declarations that have FFI (externalValue kind indicates FFI)
SELECT
    m.name AS module,
    d.name AS declaration,
    d.type_signature,
    pv.name AS package,
    p.primary_backend
FROM declarations d
JOIN modules m ON d.module_id = m.id
JOIN package_versions pv ON m.package_version_id = pv.id
JOIN snapshot_packages sp ON sp.package_version_id = pv.id
JOIN snapshots s ON sp.snapshot_id = s.id
JOIN projects p ON s.project_id = p.id
WHERE d.kind = 'foreign'
ORDER BY p.primary_backend, pv.name, m.name;
```

Testing Plan

1. **Unit tests for detection logic**
   - Test each backend detection heuristic
   - Test edge cases (missing files, mixed signals)
2. **Integration tests with real projects**
   - Run loader on psd3-tilted-radio (expects: erlang)
   - Run loader on hypo-punter/ee-server (expects: python)
   - Run loader on scuppered-ligature (expects: lua)
   - Run loader on corrode-expel (expects: js)
3. **Verify FFI LOC counting**
   - Manually count LOC in sample FFI files
   - Compare to loader output

Risks and Mitigations

Risk	Mitigation
Detection false positives	Use multiple signals, require 2+ indicators
spago.yaml format variations	Parse with serde_yaml, handle missing fields
Missing FFI files	Make FFI stats optional, don't fail on missing
Schema migration issues	Version check, provide migration SQL
Performance (large repos)	Parallelize FFI scanning, cache results

Success Criteria

1. All PSD3 polyglot projects correctly identified by backend
2. FFI LOC accurately counted for each language
3. New views enable meaningful visualizations
4. No regression in existing loader functionality
5. Stats command shows polyglot summary

Future Enhancements

1. **Multi-backend projects** - Some projects might compile to multiple backends (same source, different FFI)
2. **FFI type extraction** - Parse FFI files to extract type mappings
3. **Backend compatibility matrix** - Which packages work with which backends
4. **Visualization integration** - Add polyglot view to minard-frontend

Status / Next Steps

Status: Plan complete, ready for implementation

Next steps:

1. Review plan with stakeholders
2. Create feature branch
3. Implement Phase 1 (schema)
4. Implement Phase 2 (detection)
5. Test with all polyglot showcases
6. Merge and document