2025-10-17: Tokio Runtime Integration with eframe/egui

Problem

When attempting to perform async database operations in an egui app using `tokio::spawn`, the following error occurs:

```
there is no reactor running, must be called from the context of a Tokio
1.x runtime
```

Root Cause

- `eframe::run_native` is a blocking call that runs the GUI event loop on the main thread.
- `tokio::spawn` requires an active Tokio runtime to execute async tasks.
- Since the GUI loop blocks the thread, no Tokio runtime is available for spawning tasks from within the app's `update` method.

Solution

1. **Create Tokio Runtime in Main**: Initialize a `tokio::runtime::Runtime` in `main.rs` before calling `eframe::run_native`.

2. **Store Runtime Handle Globally**: Use a `OnceLock<tokio::runtime::Handle>` to store the runtime's handle, allowing access from anywhere in the app.

3. **Keep Runtime Alive**: Spawn a background thread that runs an async loop (e.g., waiting for signals) to prevent the runtime from shutting down.

4. **Use Handle for Spawning**: Instead of `tokio::spawn`, use `RT_HANDLE.get().unwrap().spawn(async move { ... })` to spawn tasks on the existing runtime.

Code Changes

- **lib.rs**: Add `pub static RT_HANDLE: OnceLock<tokio::runtime::Handle> = OnceLock::new();`
- **main.rs**:
  - Create runtime: `let rt = tokio::runtime::Runtime::new().unwrap();`
  - Set handle: `RT_HANDLE.set(rt.handle().clone()).unwrap();`
  - Spawn keeper thread: `std::thread::spawn(move || { rt.block_on(async { let _ = tokio::signal::ctrl_c().await; }); });`
- **app.rs**: Replace `tokio::spawn` with `crate::RT_HANDLE.get().unwrap().spawn`

Benefits

- Enables async operations (e.g., database queries) within GUI event handlers.
- Maintains thread safety with crossbeam channels for event communication.
- Allows decoupling of UI actions from async business logic.

Key Insights

- GUI frameworks like egui are typically single-threaded and blocking.
- Async runtimes must be managed separately and kept alive for the app's duration.
- Global handles provide a clean way to access runtime context from UI code.
- This pattern is essential for apps needing async I/O in immediate-mode GUIs.