Subject: Re: Book idea: U++ Web development
Posted by BetoValle on Mon, 18 May 2026 15:18:52 GMT
View Forum Message <> Reply to Message

A practical real-world example that could be extremely valuable in the book is explaining why shared/global SQL session objects can become dangerous in multithreaded native web servers.

In my own Skylark production experience, I initially treated the database session almost like a global/shared resource:

```cpp
struct ConDB {
    MySqlSession session;
};

struct dbAss {
    ConDB db;
};

String dbAss::consultaDuplicados(String cpo, Value vini)
{
    Sql sql(db.session);
    sql.Execute(st);
}
```

Under concurrent web requests, this eventually caused instability and random crashes after days or weeks of uptime.

What finally solved the problem was redesigning the architecture so each request creates its own local database context and the object destruction becomes deterministic and thread-safe:

```cpp
if (v["oper"].ToString().IsEqual("checaCodigoDuplicado"))
{
    dbAss db;
    return db.baseCads(v, &dbAss::checaCodigoDuplicado);
} // db destructor executed here
```

This became a major stability improvement because:

* each request/thread gets its own isolated DB state
* no shared SQL session exists between concurrent requests
* destruction timing becomes predictable (RAII)
* resource cleanup becomes automatic
* thread interference is dramatically reduced

After this redesign, combined with a watchdog strategy, the server stopped crashing and has now been running stably for many months.

I think examples like this are incredibly valuable for developers coming from Java, PHP or Delphi backgrounds, where frameworks often hide object lifetime and threading complexity.

In native C++ web development, understanding:

* ownership
* lifetime
* stack scope
* RAII
* thread isolation
* shared state dangers

is essential for production stability.
