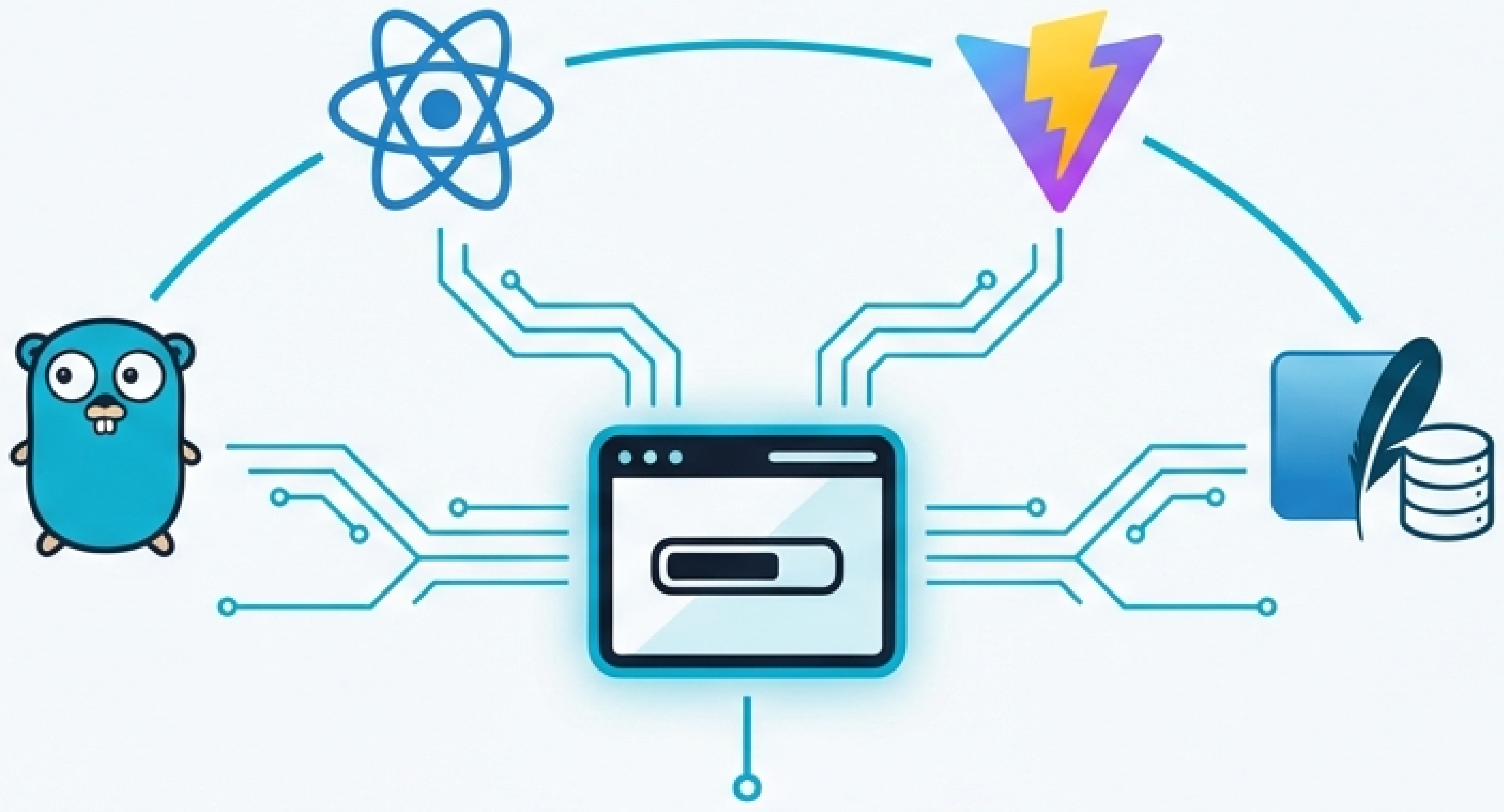
Building Native Desktop Apps with Grit

Go Backend. React Frontend. Single Binary.

The complete playbook for building, scaffolding, and distributing lightweight cross-platform applications.



Single Binary Executable

NotebookLM

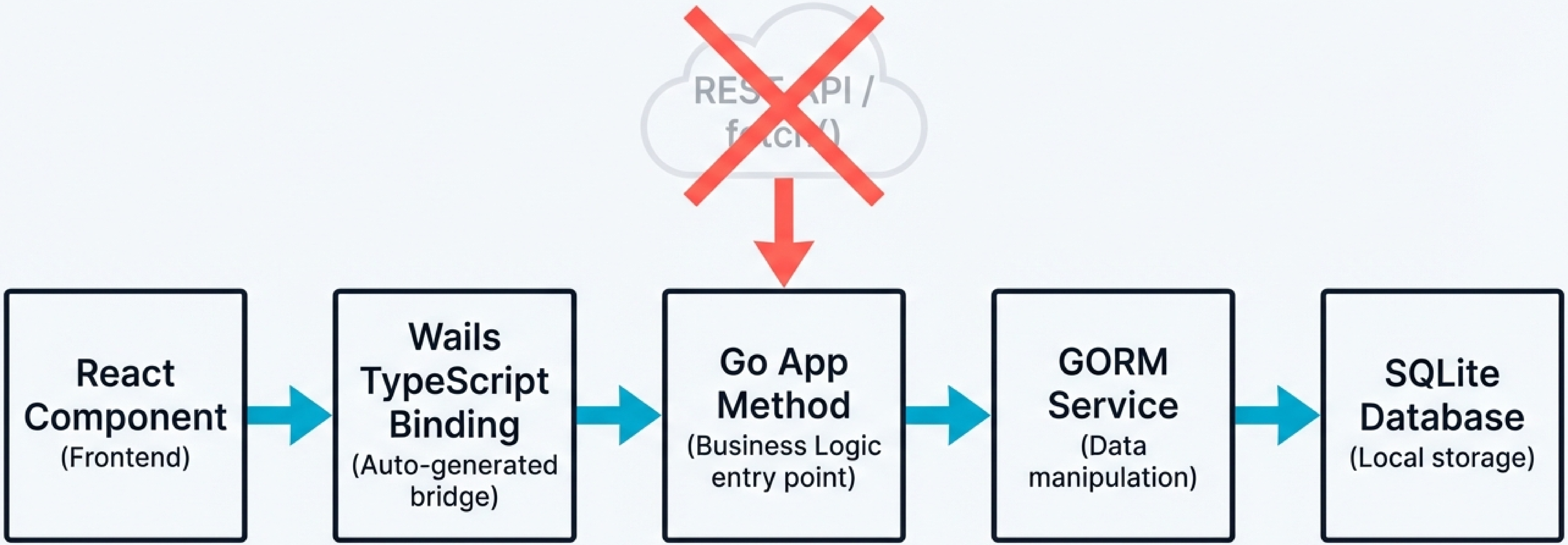
The Desktop Framework Dilemma

Grit (Wails)	Electron	Tauri
✅ ~10-15MB binary size	~150MB+ binary size	~5-10MB binary size
✅ ~30-50MB RAM	~200MB+ RAM	~30-50MB RAM
Go backend (no Rust learning curve)	JS/C++ backend	Rust backend (steep learning curve)
Direct Go-to-JS bindings	Heavy IPC serialization overhead	Rust commands

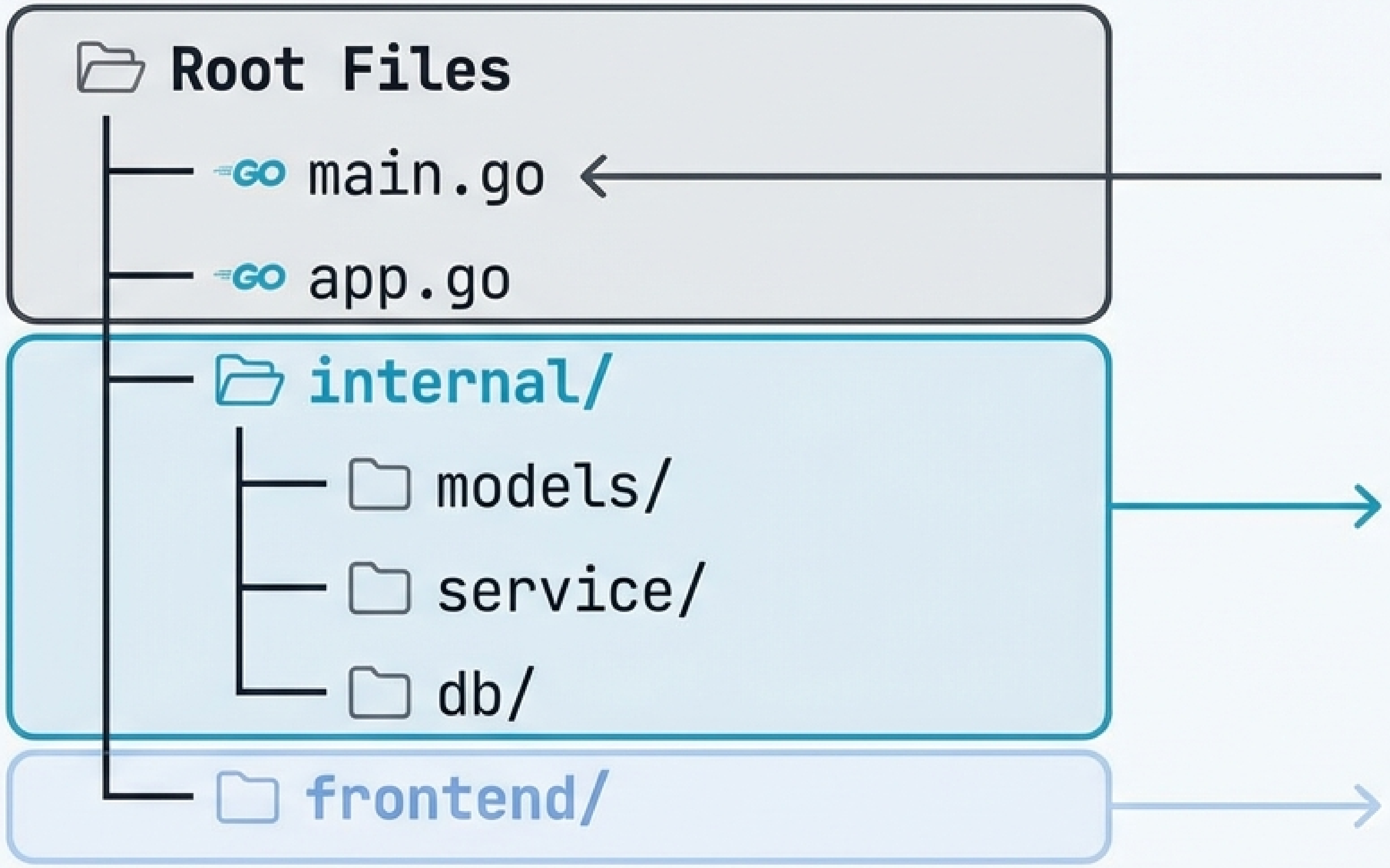
Go is the backend language. Wails is the natural choice for a unified developer experience without the bloat.

Bypassing the HTTP Layer

There is no REST API in desktop projects. The React frontend calls Go functions directly through generated bindings. No serialization overhead, no IPC, no HTTP server.


REST API / fetch()
React Component (Frontend)
Wails TypeScript Binding (Auto-generated bridge)
Go App Method (Business Logic entry point)
GORM Service (Data manipulation)
SQLite Database (Local storage)

Anatomy of a Grit App



The Wails entry point. The App struct here holds all methods bound to the frontend.

The Go backend engine. Houses GORM structs, CRUD logic, and SQLite setup.

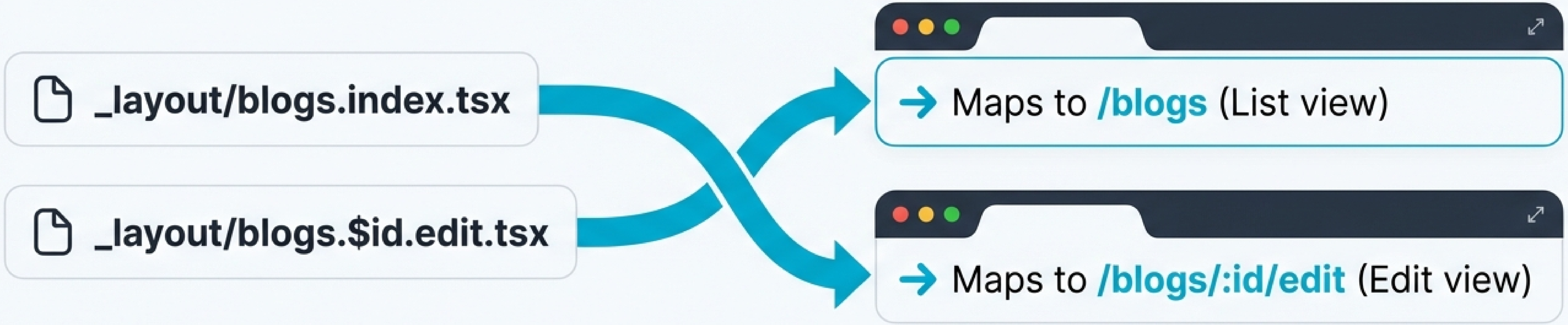
The Vite-powered React application featuring TanStack Router and Tailwind CSS.

Single directory architecture. This is not a monorepo.

Scaffolding and File-Based Routing

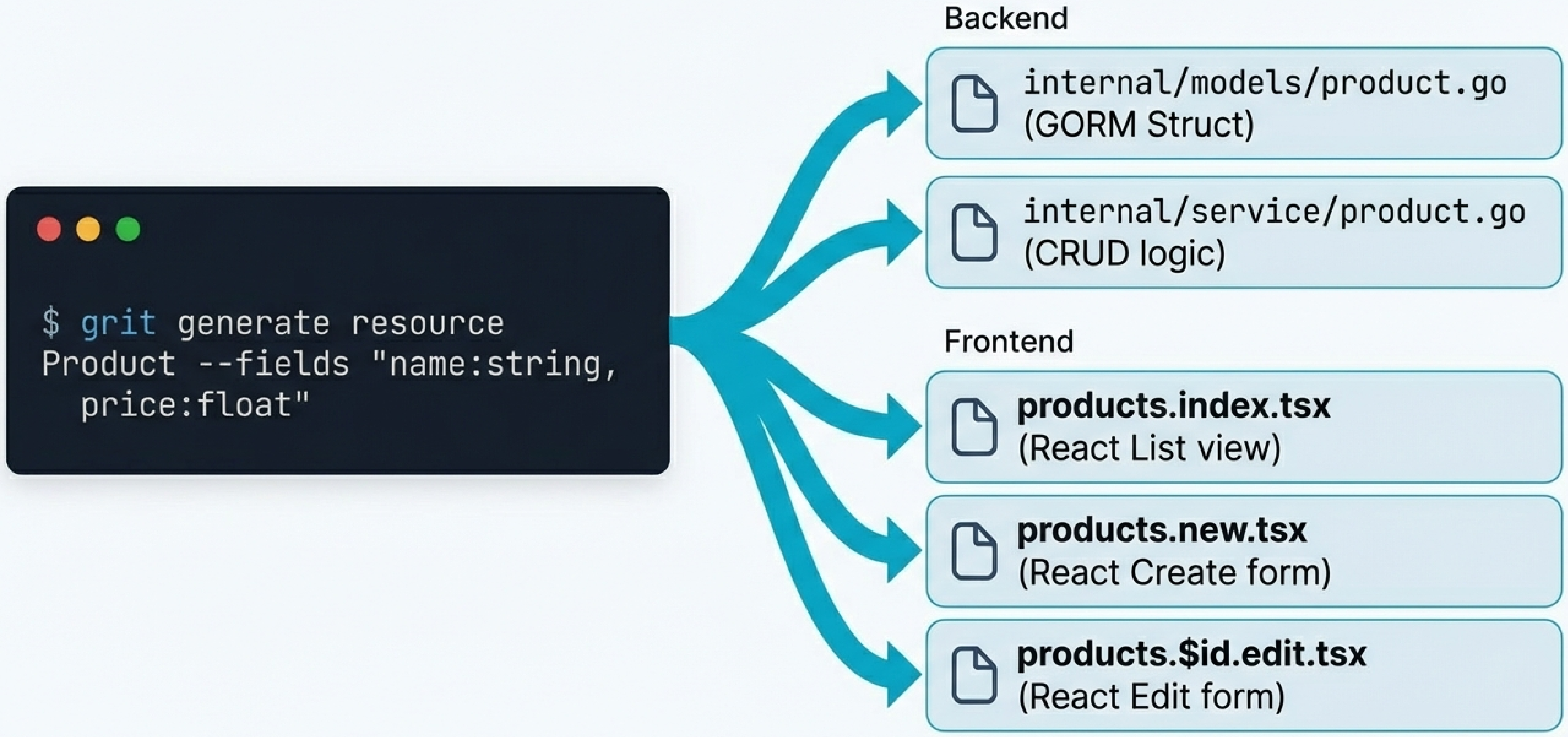
Powered by TanStack Router. Uses **createHashHistory()** to function perfectly offline from the local disk. No centralized route registry to maintain—just create or delete files.

```
$ grit new-desktop myapp
```



_layout/blogs.index.tsx
_layout/blogs.$id.edit.tsx
→ Maps to /blogs (List view)
→ Maps to /blogs/:id/edit (Edit view)

Full-Stack Resource Generation



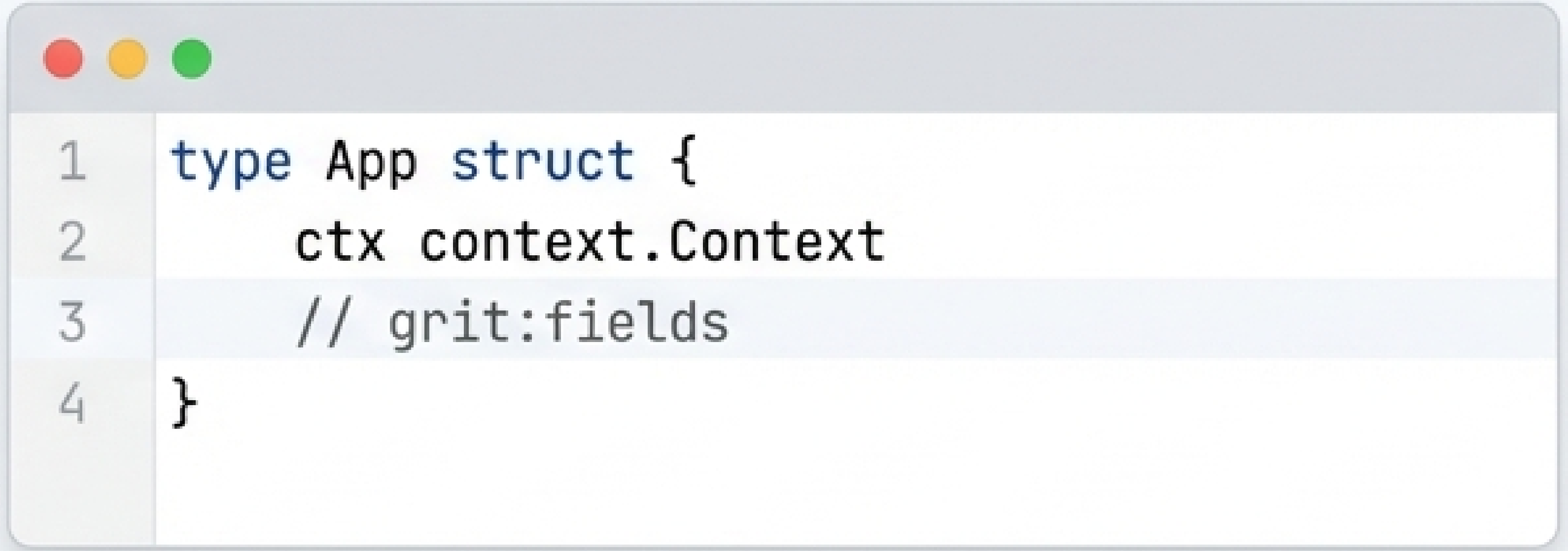
```
$ grit generate resource
Product --fields "name:string,
  price:float"
```

Backend
internal/models/product.go (GORM Struct)
internal/service/product.go (CRUD logic)
Frontend
products.index.tsx (React List view)
products.new.tsx (React Create form)
products.$id.edit.tsx (React Edit form)

Precision Code Injection

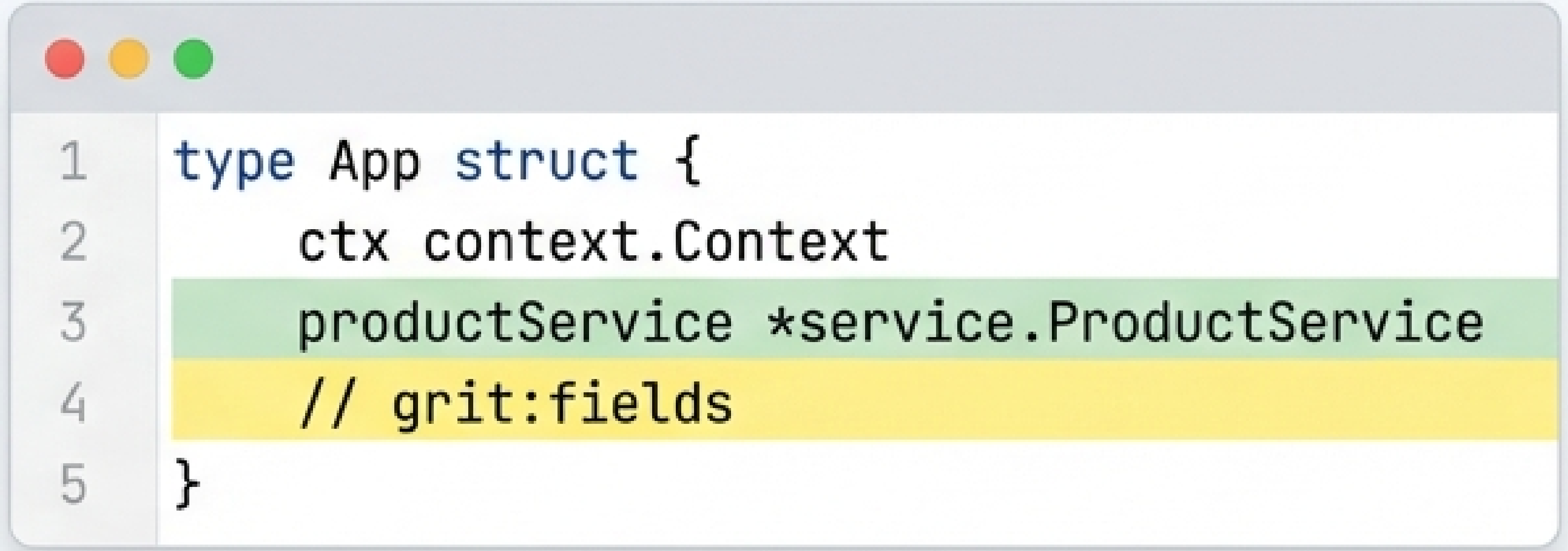
The Grit CLI executes 10 precise code injections across 6 files during resource generation.

Before



```go
type App struct {
    ctx context.Context
    // grit:fields
}
```

After



```go
type App struct {
    ctx context.Context
    productService *service.ProductService
    // grit:fields
}
```

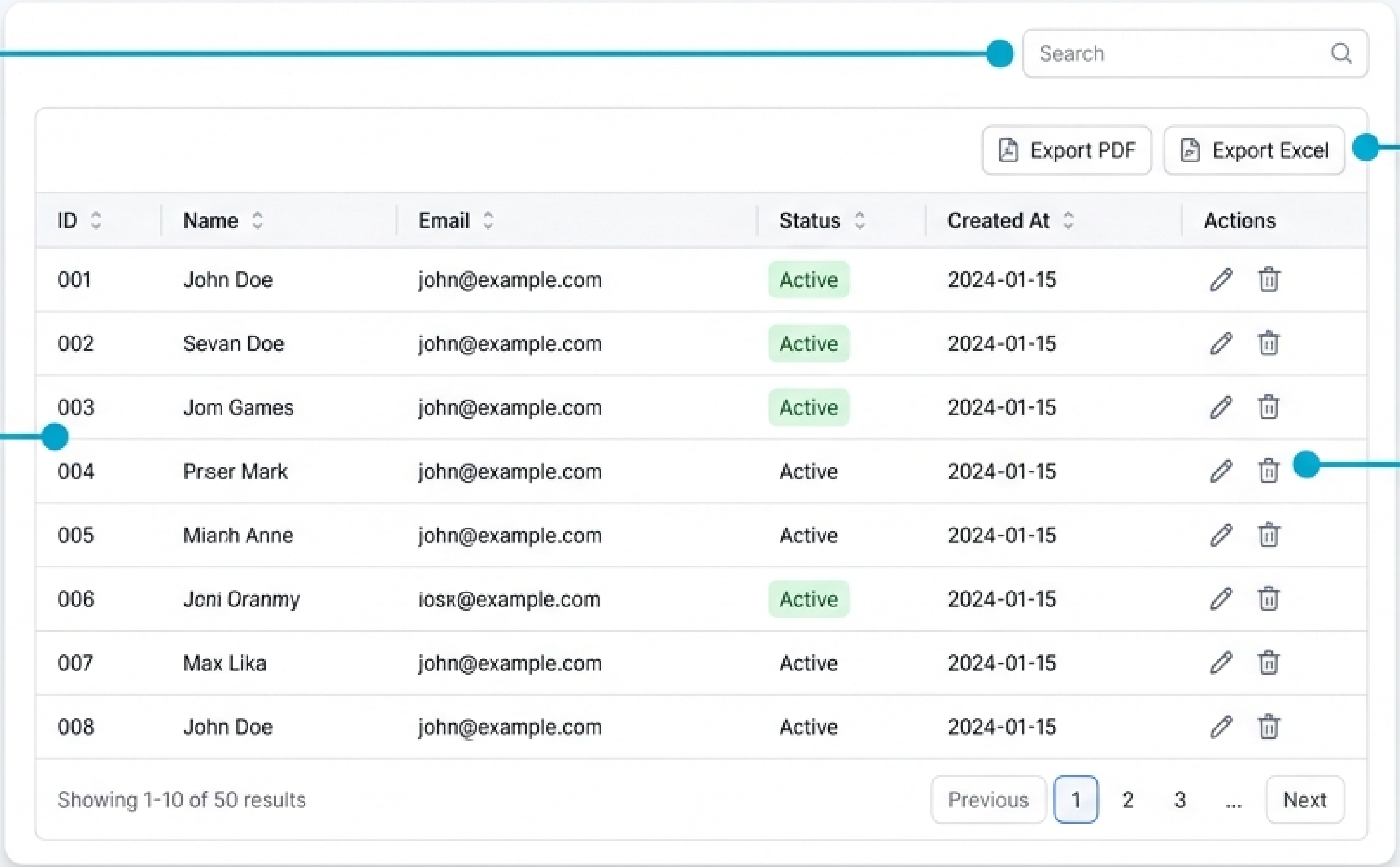
Never delete the `// grit:` markers. They are permanent injection points. Removing them breaks `grit generate` and `grit remove`.

Out-of-the-Box UI: The DataTable

Every generated resource includes a production-ready list page powered by TanStack Query.

1 **Search:** Debounced text filtering across primary fields.

2 **Data Grid:** Sortable columns and configurable pagination.



ID	Name	Email	Status	Created At	Actions
001	John Doe	john@example.com	Active	2024-01-15	
002	Sevan Doe	john@example.com	Active	2024-01-15	
003	Jom Games	john@example.com	Active	2024-01-15	
004	Prser Mark	john@example.com	Active	2024-01-15	
005	Mianh Anne	john@example.com	Active	2024-01-15	
006	Jeni Oranmy	iosк@example.com	Active	2024-01-15	
007	Max Lika	john@example.com	Active	2024-01-15	
008	John Doe	john@example.com	Active	2024-01-15	

Showing 1-10 of 50 results

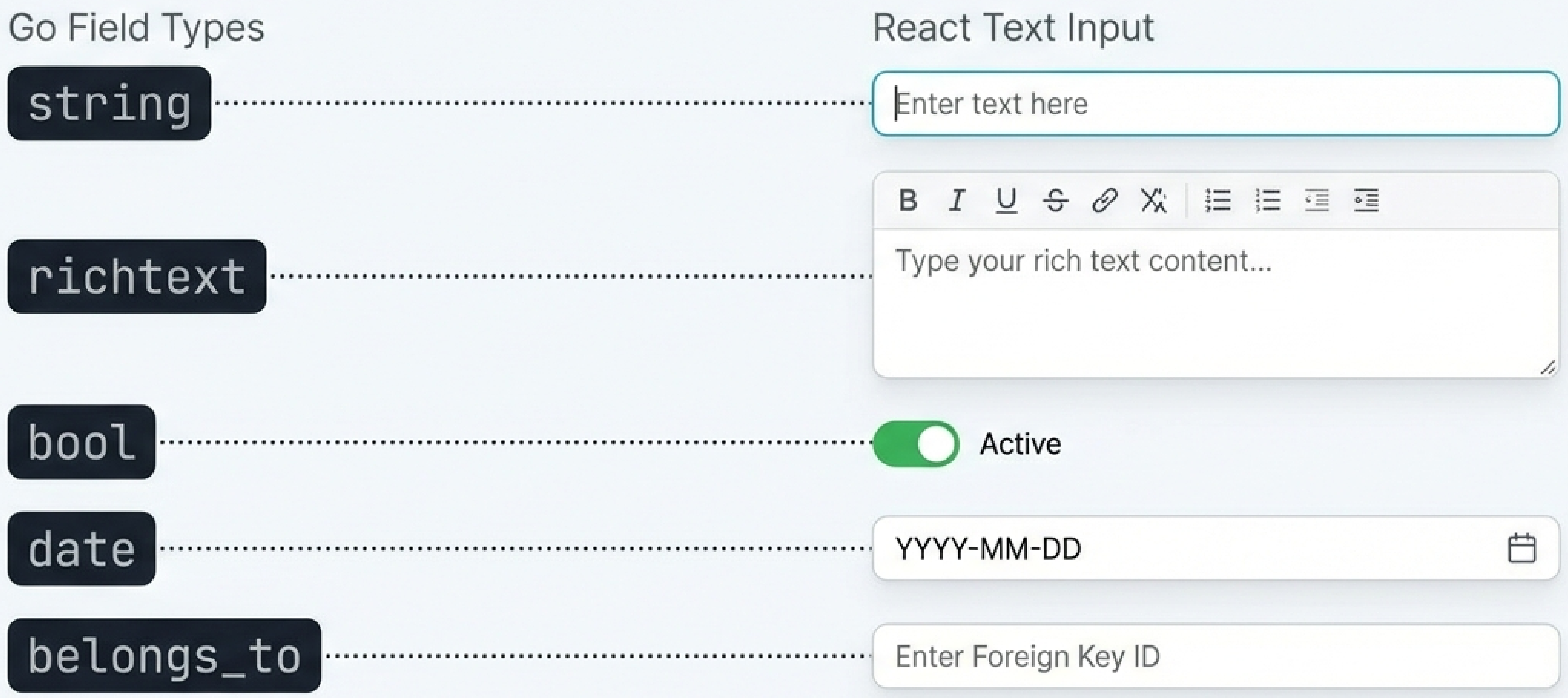
4 **Export Bar:** One-click native PDF and Excel (.xlsx) data exports.

3 **Action Menu:** Edit and Delete operations with confirmation dialogs.

NotebookLM

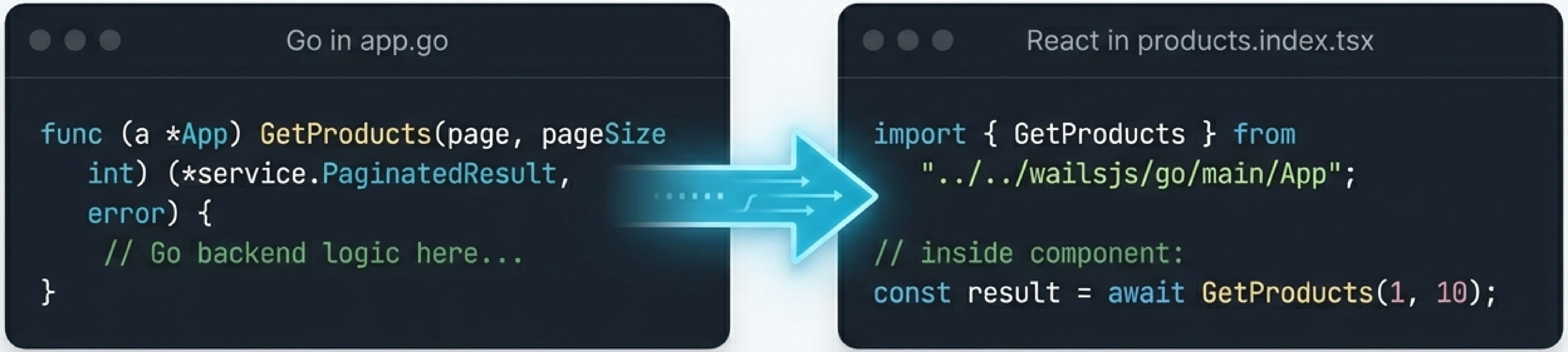
Out-of-the-Box UI: The FormBuilder

Forms handle both Create and Edit modes automatically. Validation constraints map directly from GORM to toast notifications in the UI.



The Wails Bridge in Action

Wails auto-generates TypeScript bindings for any exported method on the App struct. React calls Go exactly like a local async JavaScript function.



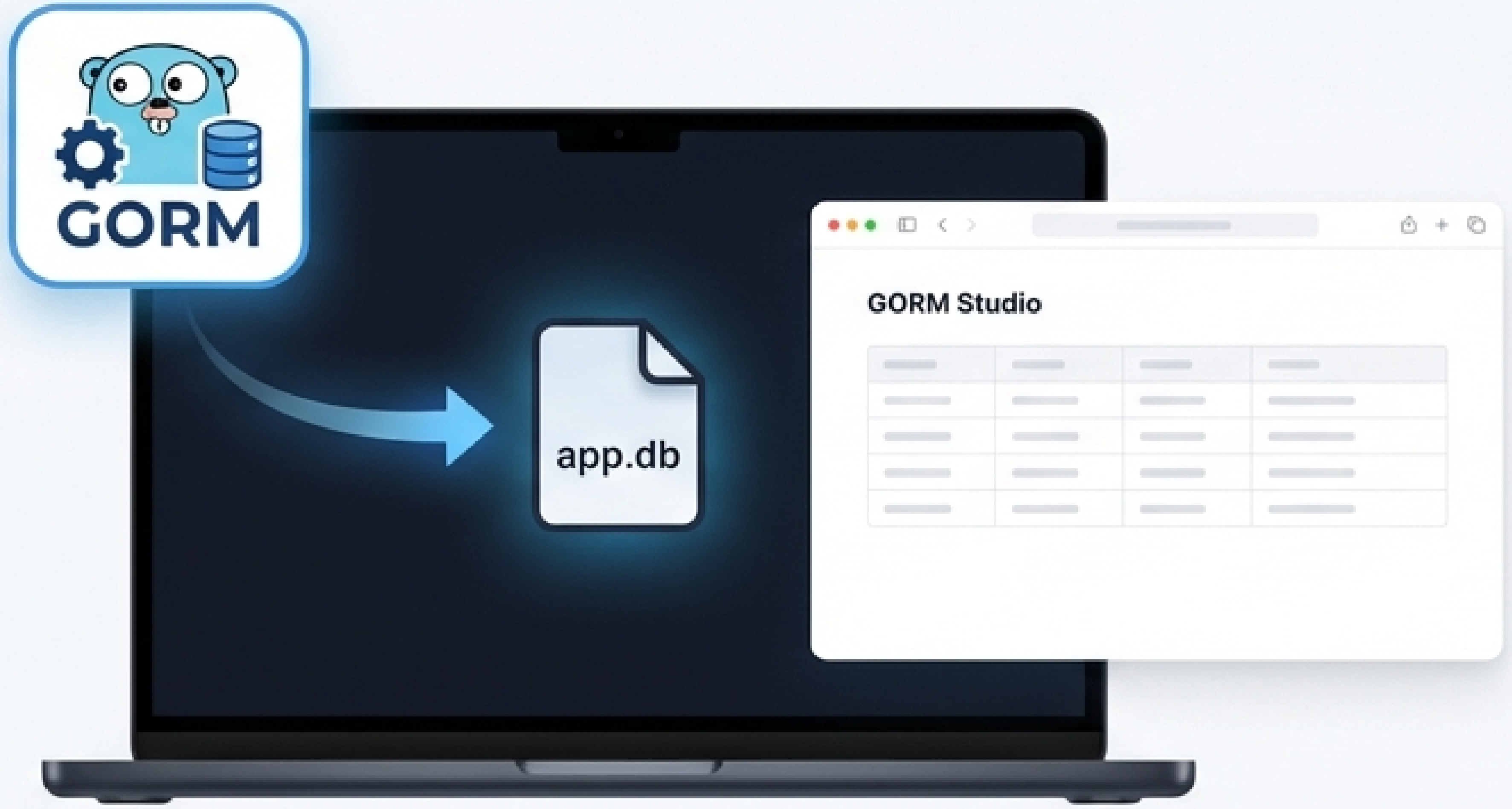
```go
func (a *App) GetProducts(page, pageSize
   int) (*service.PaginatedResult,
   error) {
    // Go backend logic here...
}
```

```tsx
import { GetProducts } from
   "../../wailsjs/go/main/App";

// inside component:
const result = await GetProducts(1, 10);
```

Offline-First Data Portability

Grit Desktop uses **local SQLite by default.** All data lives on the user's machine, requiring zero network connection. GORM handles automatic database migrations on startup.

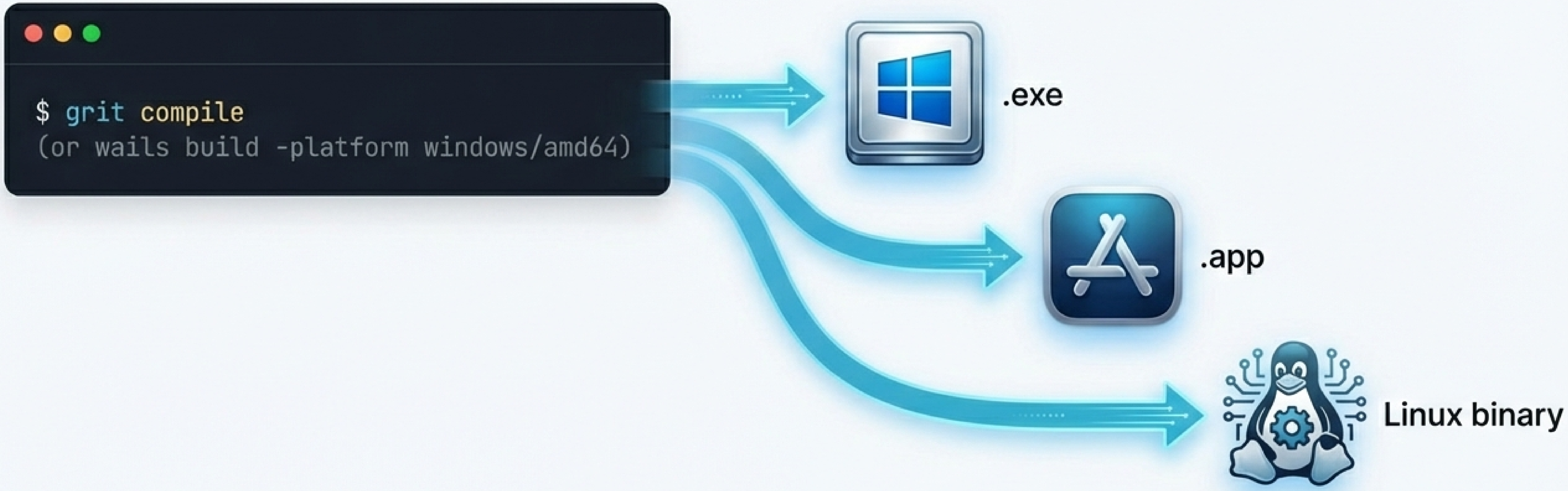


💡 **Pro Tip:** Run `$ grit studio` to launch a visual database browser on localhost:4000 to inspect tables and run queries during development.

Compiling the Single Binary

The entire React frontend is embedded directly into the Go binary via `//go:embed all:frontend/dist`. Zero external assets to distribute.

Use the `-nsis` flag to automatically generate a Windows installer with Start Menu shortcuts and uninstallation support.



```
$ grit compile
(or wails build -platform windows/amd64)
```

Golden Rules

Architectural Constraints

1. **SQLite Only**
   Do not use PostgreSQL, Docker, or networked databases for standard desktop builds.

2. **No HTTP/REST**
   Never use `fetch()` or build `Gin` handlers. Use Wails bindings exclusively.

3. **The App Struct Rule**
   Only exported methods attached to the `App` struct (in `app.go`) are exposed to the React frontend.

Workflow Constraints

1. **Strict CLI Usage**

   Always use `grit generate` and `grit remove`. Never manually create or delete resource files, as this orphans injected code.

2. **Relational Ordering**

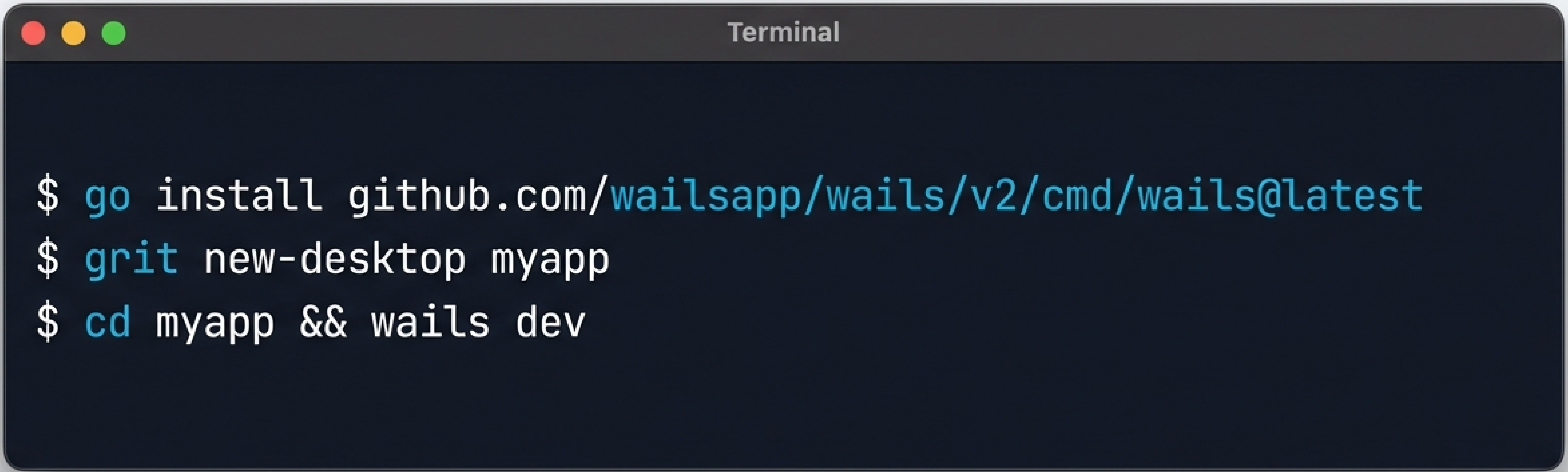
   Generate parent models before child models (e.g., generate Category before generating a Product with a category_id:belongs_to field).

3. **Binding Regeneration**

   Always restart `wails dev` after adding new Go methods so Wails can regenerate the TypeScript bindings.

Ready to Build

Native performance. Uncompromising developer experience. Zero Electron bloat.



```
$ go install github.com/wailsapp/wails/v2/cmd/wails@latest
$ grit new-desktop myapp
$ cd myapp && wails dev
```

Documentation: docs.grit.com | GitHub: @grit-framework