Electron Prioritizer – Developer Workflow Guide

This document explains **exactly** how to add new database tables, queries, IPC handlers, routes, and renderer APIs **without breaking type safety**.

It is designed to be followed **step-by-step every time** you add or modify functionality.

Architecture Overview

The app is split into three strict layers:

1. **Database + Business Logic (Electron main process)**
2. **Shared IPC contract (types only)**
3. **Renderer (Next.js UI)**

```
Renderer (Next.js)
  ↓ window.api
Preload (typed bridge)
  ↓ ipcRenderer.invoke
Electron IPC handlers
  ↓ repositories
Drizzle / SQLite
```

Key folders

```
electron/
  db/schema/          # Drizzle tables + DB types
  repositories/       # DB queries only
  ipc/                # IPC handlers (main process)
  preload.ts          # window.api bridge

shared/ipc/
  routes.ts           # IPC contract (SOURCE OF TRUTH)
  channels.ts         # Route name constants
  result.ts           # IpcResult<T>
  schemas.ts          # Optional Zod validation

app/                  # Next.js UI (renderer)
```

Golden Rules (Do Not Break These)

- **Renderer never imports Electron or DB code**
- **Renderer only talks to `window.api`**
- **Every IPC route must exist in `shared/ipc/routes.ts`**

- **Every IPC response must return `IpcResult<T>`**
- **Repositories never know about IPC or Electron**
- **Preload must stay minimal**

If these rules are followed, type safety is guaranteed end-to-end.

---

1. Adding or Modifying a Database Table

1.1 Create or update a Drizzle schema

Add a new file or update an existing one:

```
electron/db/schema/<table>.ts
```

Example:

```
import { sqliteTable, text, integer } from "drizzle-orm/sqlite-core";

export const projects = sqliteTable("projects", {
  id: text("id").primaryKey(),
  name: text("name").notNull(),
  description: text("description"),
  archived: integer("archived", { mode: "boolean"
}).notNull().default(false),
});
```

1.2 Export the table

In:

```
electron/db/schema/index.ts
```

```
export * from "./projects";
```

1.3 Generate a migration

```
yarn drizzle-kit generate
```

1.4 Apply migrations locally

```
yarn db:migrate
```

---

2. Adding Database Queries (Repository Layer)

Repositories live in:

```
electron/repositories/
```

Rules for repositories

- DB only
- No IPC, no Electron imports
- Must return JSON-serializable data

Example

```
// electron/repositories/projects.repo.ts
import { db } from "../db";
import { projects } from "../db/schema";
import { eq } from "drizzle-orm";

export function list() {
  return db.select().from(projects);
}

export async function create(input: { id: string; name: string;
description?: string }) {
  await db.insert(projects).values(input);
  return input;
}

export function update(id: string, patch: Partial<typeof
projects.$inferInsert>) {
  return db.update(projects).set(patch).where(eq(projects.id, id));
}
```

---

3. Defining the IPC Contract (Most Important Step)

The **single source of truth** for IPC lives in:

```
shared/ipc/routes.ts
```

3.1 Add a route

```
export type IpcRoutes = {
  "projects:list": {
    input: void;
    output: Project[];
  };

  "projects:create": {
    input: Pick<NewProject, "name" | "description">;
    output: Project;
  };
};
```

3.2 Route result type

All IPC responses are wrapped:

```
export type RouteResult<K extends RouteKey> = IpcResult<IpcRoutes[K]
["output"]>;
```

This guarantees:

```
{ ok: true, data: T }
{ ok: false, error }
```

---

4. Implementing the IPC Handler (Electron Main)

Handlers live in:

```
electron/ipc/
```

4.1 Create the handler

```
import { ipcMain } from "electron";
import { createIpcHandler } from "./_shared";
import * as projectsRepo from "../repositories/projects.repo";

ipcMain.handle(
  "projects:create",
  createIpcHandler({
    handler: async (_event, input) => {
```

```
        return projectsRepo.create({
          id: crypto.randomUUID(),
          ...input,
        });
      },
    })
  );
```

4.2 Register the handler

In:

```
electron/ipc/index.ts
```

Ensure the file is imported or registration function is called.

---

5. (Optional but Recommended) Route Constants

```
shared/ipc/channels.ts
```

```
export const IPC = {
  projects: {
    list: "projects:list",
    create: "projects:create",
  },
} as const;
```

Avoids string typos.

---

6. Exposing IPC via Preload

Edit:

```
electron/preload.ts
```

6.1 Generic invoke helper

```
import { contextBridge, ipcRenderer } from "electron";
import type { RouteKey, RouteResult, IpcRoutes } from
"../shared/ipc/routes";
```

```
function invoke<K extends RouteKey>(
  channel: K,
  input: IpcRoutes[K]["input"]
): Promise<RouteResult<K>> {
  return ipcRenderer.invoke(channel, input);
}
```

6.2 Expose API

```
contextBridge.exposeInMainWorld("api", {
  projects: {
    list: () => invoke("projects:list", undefined),
    create: (input) => invoke("projects:create", input),
  },
});
```

---

7. Renderer Typings (`window.api`)

Edit:

```
types/electron.d.ts
```

```
import type { IpcRoutes, RouteResult } from "../shared/ipc/routes";

export {};

declare global {
  interface Window {
    api: {
      projects: {
        list: () => Promise<RouteResult<"projects:list">>;
        create: (
          input: IpcRoutes["projects:create"]["input"]
        ) => Promise<RouteResult<"projects:create">>;
      };
    };
  }
}
```

---

8. Calling from the Renderer

```
const res = await window.api.projects.create({ name: "Test" });

if (!res.ok) {
  console.error(res.error);
  return;
}

console.log(res.data);
```

---

9. Zod Validation (Optional)

If you want runtime validation:

```
createIpcHandler({
  schema: ProjectsCreateSchema,
  handler: async (_e, input) => projectsRepo.create(input),
});
```

Returns `VALIDATION_ERROR` automatically.

---

10. Debug Checklist

Preload fails to load

- `sandbox: false` must be set
- Ensure preload path points to `dist-electron/electron/preload.js`

IPC not firing

- Route exists in `shared/ipc/routes.ts`
- Handler registered in `electron/ipc/index.ts`
- Preload uses same route key

Types missing in UI

- `types/electron.d.ts` included
- Using `window.api`, not `ipcRenderer`

Final Checklist (Use This Every Time)