Overview

Build a useful business dashboard

React for the interface. Node.js for the API. MySQL for data.
Use ChatGPT to assist each reviewed step.

CodeKits / Workboard • Original source + optional Phoenix integration guide

Define the product

Professional means usable and reliable

Start with a focused task workflow: create a task, view the list and update its status. Define how failures should behave.

```
MVP acceptance criteria:
• Create a task with a 1–120 character title
• Move it through To do → In progress → Done
• Search the loaded records
• Show loading, empty and error states
• Validate every mutation on the server
```

A good-looking template is a starting point, not a finished application.

Architecture

Keep the API independent

Nginx serves static React files and proxies /api to Node. The browser uses same-origin relative URLs.

```
Browser → Nginx HTTPS
             ├─ / → React build files
             └─ /api → Node.js loopback:3000
                          ↓
                   MySQL loopback:3306
```

A Swift or desktop client can reuse an appropriately authenticated API.

Choose a template

Phoenix is an optional starting point

Phoenix has separate HTML and React editions. Use a compatible React edition and follow its included documentation.

```
Choose the correct edition and licence
Run the unchanged purchased template first
Find its router, layout and component exports
Add one feature page inside the existing app
Keep purchased source in a private repository
```

This kit includes an original dashboard, not Phoenix files or an imitation of its source.

Choose a template

A template is UI, not security

Sign-in pages, role labels and hidden menu links do not protect API routes by themselves.

```
Template layer: cards, tables, forms, navigation
Application layer: state, requests, validation
Server layer: authentication and authorization
Database layer: schema, queries and constraints
```

The included internal demo uses Nginx Basic authentication over HTTPS. It has one shared permission level.

Use ChatGPT

Ask for a plan before code

Give the assistant your framework versions, real files, scope and acceptance criteria. Request one vertical slice at a time.

```
Prompt:
"Plan a React + Express + MySQL task dashboard.
Define the schema, routes, validation and access controls.
Implement create-task end to end first.
Use the files I supplied; do not invent template imports.
Explain how I can verify the result."
```

The starter includes a reusable prompt pack for planning, debugging, review and testing.

Use ChatGPT

Keep humans responsible for verification

Review generated changes and run the checks. Give redacted logs and reproducible steps when asking for debugging help.

```
Prompt:
"Review this diff for authorization, CSRF, SQL injection,
secret leakage and accessibility. Separate confirmed bugs
from hypotheses. Give a reproduction for each finding.
Do not claim tests passed unless they were run."
```

Never paste production secrets or private customer records into prompts.

Data model

Keep the first schema small

A task needs a stable ID, title and controlled status. Use a database constraint as well as API validation.

```
CREATE TABLE tasks (
 id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
 title VARCHAR(120) NOT NULL,
 status ENUM('todo','doing','done')
   NOT NULL DEFAULT 'todo'
) ENGINE=InnoDB;
```

Use reviewed migrations for later schema changes; never reset production tables to fix a bug.

API contract

Define the routes before wiring UI

Choose predictable JSON shapes and status codes so the frontend can handle success and failure.

```
GET   /api/tasks       → 200 { tasks: [...] }
POST  /api/tasks       → 201 { task: {...} }
PATCH /api/tasks/:id   → 200 { ok: true }

400 invalid input • 403 forbidden origin
404 missing row   • 503 backend unavailable
```

This starter shows the latest 200 tasks; counts and search apply to that loaded set.

API implementation

Validate and parameterise every write

Client validation improves usability; server validation protects the database. SQL placeholders keep input separate.

```
const title = req.body?.title;
if (typeof title !== 'string' || !title.trim() ||
    title.trim().length > 120) {
  return res.status(400).json({ error: 'Invalid title' });
}
await db.execute(
 'INSERT INTO tasks (title,status) VALUES (?,?)',
 [title.trim(), 'todo']
);
```

Use an allowlist for status values and verify numeric route IDs too.

API implementation

Protect state-changing requests

The demo checks the exact Origin for mutations and limits JSON body size. Authentication is provided by Nginx.

```
if (['POST','PATCH','DELETE','PUT'].includes(req.method) &&
    req.get('origin') !== process.env.APP_ORIGIN) {
 return res.status(403).json({ error: 'Origin not allowed' });
}
app.use(express.json({ limit: '8kb' }));
```

Origin checks address browser CSRF; they do not authenticate a user. Never expose the Node port publicly.

React interface

Build reusable pieces

Create a clear hierarchy: navigation, page heading, summary cards, filters, form and table.

```
App
 ├─ Sidebar and page heading
 ├─ Summary cards
 └─ Task panel
     ├─ Search and create form
     ├─ Loading / error / empty states
     └─ Table with status controls
```

Use visible labels, semantic buttons and keyboard focus. Do not rely on colour alone.

React interface

Use one API helper

Keep status and error handling in one place. Relative /api URLs avoid hardcoding a second browser origin.

```
async function api(path, options = {}) {
 const res = await fetch('/api' + path, options);
 const body = await res.json();
 if (!res.ok) throw new Error(body.error ?? 'Request failed');
 return body;
}
```

The full helper also sends JSON headers. The Vite dev proxy forwards /api to local Node.

React interface

Connect the form to the database

The form submits JSON to Node, which validates and writes to MySQL. Reload the authoritative list after success.

```
await api('/tasks', {
 method: 'POST',
 headers: { 'Content-Type': 'application/json' },
 body: JSON.stringify({ title })
});
setTitle('');
await load();
```

Disable repeated submissions while busy. Show a useful error instead of silently losing input.

React interface

Wire status changes and filters

A controlled select sends a PATCH request. Search filters the loaded list without changing the server data.

```
await api('/tasks/' + id, {
 method: 'PATCH',
 body: JSON.stringify({ status })
});
await load();

const shown = tasks.filter(task =>
 task.title.toLowerCase().includes(query.toLowerCase()));
```

For large datasets, move pagination, filtering and global aggregates to the API.

Local development

Run frontend and backend separately

Configure a local database and API environment. Use two terminals for development.

```
# API environment: DB_HOST, DB_USER, DB_PASSWORD, DB_NAME
# Set APP_ORIGIN=http://localhost:5173
cd api && npm ci && npm start

# Second terminal, from project/:
cd frontend && npm ci && npm run dev
```

The local dev API has no login layer. Keep it bound to your own machine.

Phoenix integration

Move the feature into the real template

Use the purchased template’s actual exports and router. Transfer feature behavior rather than copying a second app root.

1. Add a Workboard page in the existing router
2. Use the documented layout wrapper
3. Map cards, form and table to actual components
4. Replace sample arrays with API data
5. Preserve validation, errors and accessibility
6. Build and test before removing demo pages

Exact import paths vary by Phoenix version. PHOENIX-INTEGRATION.md explains the adaptation.

Testing

Test behavior, not just rendering

The included tests exercise API validation, forbidden origins, parameter binding, missing rows and safe failures.

```
# From project/api:
npm test
# From project/frontend:
npm run build

# Then verify against a real MySQL instance:
create task → refresh → change status → refresh
```

Database doubles test HTTP behavior; they do not prove real SQL or infrastructure works.

Ubuntu setup

Prepare a fresh VPS and DNS

Create a sudo deployment user with tested SSH key access. Point admin.example.com to the VPS.

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y nginx mysql-server ufw curl \
 ca-certificates xz-utils openssl rsync snapd apache2-utils
sudo ufw allow OpenSSH
sudo ufw allow "Nginx Full"
sudo ufw enable
```

Allow your actual SSH port before enabling UFW. Do not open 3000 or 3306.

Ubuntu setup

Upload and initialise the app

Copy the project without dependencies or secrets. The setup script creates a least-privilege database account.

```
# Local, inside project/:
rsync -av --exclude=node_modules --exclude=dist \
 --exclude=".env*" ./ deploy@VPS_IP:~/workboard/
# VPS:
cd ~/workboard
bash scripts/install-node24.sh
bash scripts/init-database.sh
sudo nano /etc/workboard.env
```

Set APP_ORIGIN=https://admin.example.com to your exact real origin, without a trailing slash.

Deploy and protect

Install services and add a login gate

Deploy the API and static frontend. Create the Nginx password file before enabling the protected site.

```
bash scripts/deploy.sh
sudo htpasswd -c /etc/nginx/workboard.htpasswd admin
sudo chown root:www-data /etc/nginx/workboard.htpasswd
sudo chmod 640 /etc/nginx/workboard.htpasswd
nano deploy/nginx.conf
```

Use -c only to create the file initially; omit it when adding another account. Enter passwords at the prompt.

Deploy and protect

Protect both pages and API routes

Server-level Basic authentication covers the frontend and /api. Nginx serves React and forwards API traffic.

```
auth_basic "Workboard internal demo";
auth_basic_user_file /etc/nginx/workboard.htpasswd;
location /api/ { proxy_pass http://127.0.0.1:3000; }
location / { try_files $uri $uri/ /index.html; }
```

Basic auth is an internal-demo gate, not a full account system. All users have equal rights.

Deploy and protect

Enable HTTPS before logging in

Install the site and use Certbot with an HTTPS redirect. Do not submit your login password over plain HTTP.

```
sudo cp deploy/nginx.conf /etc/nginx/sites-available/workboard
sudo ln -s /etc/nginx/sites-available/workboard \
 /etc/nginx/sites-enabled/workboard
sudo nginx -t && sudo systemctl reload nginx
sudo snap install --classic certbot
sudo /snap/bin/certbot --nginx --redirect -d admin.example.com
sudo /snap/bin/certbot renew --dry-run
```

Before TLS, an HTTP 401 shows the gate is active. Use the HTTPS URL for the first login.

Verify and maintain

Check the full workflow and backups

Confirm unauthenticated requests are rejected, valid users can edit, and forbidden origins cannot change data.

```
curl -I https://admin.example.com/api/tasks
# Expect 401 without credentials.
curl -u admin https://admin.example.com/api/tasks
# curl prompts for the password.
bash scripts/backup.sh
sudo journalctl -u workboard -n 50 --no-pager
```

Copy backups off-server and test restoration. Review logs without exposing credentials.

Production roadmap

Know what is still missing

A public customer product needs more than this internal demo. Add features according to real requirements.

```
User sessions and dependable logout
Per-record and role-based authorization
Audit events and account recovery
Pagination, migrations and monitoring
Restore tests and deployment rollback
Accessibility and device testing
```

Use ChatGPT to review the plan and code, then verify every important claim yourself.

Finish

Ship a workflow, not just a theme

Build one feature end to end, review it and test it.
Explore custom business apps at CodeKits.

codekits.shop • Original starter, Phoenix guide and ChatGPT prompt pack included

References and validation

Silent illustrated walkthrough using an original dashboard. Optional Phoenix React integration is explained; paid Phoenix source is not included. Frontend build and API tests with database doubles pass. Live MySQL, Nginx authentication and VPS/TLS still require verification.

Phoenix React official product

React documentation

Express production security

MySQL2 prepared statements

Nginx Basic authentication

Certbot