Icecat Learning Hub – End■to■End Handover

Generated: 2025-09-10 12:39

This document compresses our entire conversation into a single, practical guide. It includes setup, debugging on Windows, Prisma schema/seed, Next.js fixes, Docker, deployment, admin UX, and ready-to-paste Codex prompts. You can pass this PDF into a new chat as context.

1) Project Summary

A login■based Learning Hub for Icecat built with **Next.js 14 App Router**, **TypeScript**, **Tailwind**, **Prisma + PostgreSQL**, **NextAuth** (credentials), **MDX** content, and **Playwright/Vitest** tests. The app supports trainings, lessons, quizzes, a progress checklist, and an Admin area where content is managed.

Key folders:

```
app/                # App Router pages
actions/            # Server actions (progress, quiz, admin, presentations, approvals)
components/         # UI components (checklist, quiz runner, admin widgets)
content/            # MDX lessons (intro.mdx, tools.mdx, policies.mdx)
lib/                # db.ts, rbac.ts, auth.ts, mdx.ts, quiz.ts, i18n.ts
prisma/             # schema.prisma, migrations/, seed.ts
tests/, e2e/        # unit + e2e tests
docker-compose.yml  # postgres + app (optional)
```

2) Local Setup on Windows (PowerShell)

Install prerequisites: **Node.js 20+**, **Git for Windows**, **Docker Desktop** (optional for Postgres).

```
# Move to your project
cd "C:\Users\Rup\Desktop\Codes\icecat-learning-hub"

# Create .env from template
Copy-Item .env.example .env  # or create manually

# Edit .env in VS Code (do NOT paste env lines into terminal)
# Example values:
DATABASE_URL=postgresql://icecat:icecat@localhost:5432/icecat_hub?schema=public
NEXTAUTH_URL=http://localhost:3000
NEXTAUTH_SECRET=<paste 64-char hex from the command below>

# Generate a strong NEXTAUTH_SECRET (run in PowerShell):
node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"
```

Common mistake: pasting `DATABASE_URL=...` into PowerShell. Those are not commands. Put them in **.env** only.

3) Database Options

A. Dockerized Postgres (quick start):

```
# Run from the project root where docker-compose.yml exists
docker compose up -d postgres
```

B. Local Postgres: create a database named `icecat_hub` and keep the same .env connection string (host localhost:5432).

4) Prisma: format, generate, migrate, seed

```
npx prisma format
npx prisma generate
npx prisma migrate dev --name init
```

```
npx prisma db seed
```

If schema errors appear (P1012), check that enums/models are at the top level, no nested comments inside enums, and run `prisma format` again.

5) Start the App

```
npm run dev
```

Open http://localhost:3000 and sign in with demo accounts (seed): `admin@example.com`, `trainer@example.com`, `employee@example.com`, `viewer@example.com` — all with `password123`.

6) Troubleshooting Highlights

• **'.env is greyed out'**: VS Code shows it as ignored; that's normal. Ensure it's in the project root, not another folder.
• **docker compose: no configuration file provided**: You're not in the folder that contains `docker-compose.yml`.
• **Next.js complaining about next.config.ts**: rename to `next.config.mjs` (ESM) with a minimal default export.
• **NPM ERESOLVE (eslint 9 vs next 14)**: pin eslint to 8.57.0 and eslint-config-next to ^14.2.5, then `npm install`.
• **Quizzes orderBy createdAt**: if `createdAt` not in your `Quiz` model, order by `id`/`title` or add timestamps then migrate.
• **Missing file import** (`MarkDoneButton.tsx`): restore the component or update the import on dashboard.

Minimal next.config.mjs

```
/** @type {import('next').NextConfig} */
const nextConfig = {};
export default nextConfig;
```

7) VS Code Chat/Codex Agent Approvals (reduce popups)

File → Preferences → Settings (JSON) and add:

```
{
  // Turn on Agent mode features in VS Code chat
  "chat.agent.enabled": true,

  // Allow tools (including those from extensions) to run in agent mode
  "chat.extensionTools.enabled": true,

  // Auto-approve tool invocations in agent mode
  "chat.tools.autoApprove": true
}
```

Some extensions still request confirmation for destructive actions (scaffold, file writes). That's a safety choice by the extension and can't always be bypassed.

8) Restoring `MarkDoneButton` (server action form)

```
// components/MarkDoneButton.tsx
import { toggleProgress } from "@/actions/progress";

export default function MarkDoneButton({ lessonId, done }: { lessonId: string; done?: boolean; }) {
  async function run() {
    "use server";
    await toggleProgress(lessonId);
  }
  return (
    <form action={run}>
      <button
```

```
        type="submit"
        className={["px-3 py-1.5 rounded-full text-sm border transition",
          done ? "bg-green-50 border-green-200 text-green-700 hover:bg-green-100"
               : "bg-white border-slate-300 text-slate-800 hover:bg-slate-50"].join(" ")}
        aria-pressed={!!done}
      >
        {done ? "Done" : "Mark as done"}
      </button>
    </form>
  );
}
```

9) Feature: Presentations (Public Read■only, Admin CRUD)

Public `/presentations` shows a minimalist grid, filtered by audience (For Retailers / For Brands).

Admin `/admin/presentations` manages items. Final schema:

```
enum Audience { RETAILERS BRANDS }

model Presentation {
  id          String    @id @default(cuid())
  title       String
  description String?
  path        String
  audience    Audience
  tags        String[]  @default([])
  createdById String?
  createdBy   User?     @relation(fields: [createdById], references: [id])
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt
  @@unique([title, audience])
  @@index([audience, updatedAt])
}
```

Actions: create/update/delete (sketch)

```
export async function createPresentation(formData: FormData) { /* upsert by (title, audience) */ }
export async function updatePresentation(formData: FormData) { /* update by id */ }
export async function deletePresentation(id: string) { /* delete by id */ }
```

Admin polish: modal, search, audience pills

Admin toolbar provides quick search and audience filter. New/Edit open as modals; table shows Edit/Delete. Public page remains read■only with the audience switch pills.

10) Admin Shell & Sections

A dedicated Admin layout with a left sidebar: Overview, Manuals, Presentations, Documents, Quizzes, Training program, User approvals. Access is enforced server■side (role ADMIN/TRAINER) and optionally via middleware.

Top■right user menu links to Admin for ADMIN/TRAINER.

11) Lightweight User Approvals

```
enum ApprovalStatus { PENDING APPROVED REJECTED }
enum ApprovalType   { ACCESS ROLE_CHANGE }
```

```
model UserApproval {
  id           String   @id @default(cuid())
  userId       String
  user         User     @relation(fields: [userId], references: [id])
  type         ApprovalType
  requestedRole Role?
  reason       String?
  status       ApprovalStatus @default(PENDING)
  decidedById  String?
  decidedBy    User?    @relation("Decider", fields: [decidedById], references: [id])
  createdAt    DateTime @default(now())
  decidedAt    DateTime?
}
```

Server actions: `approveRequest(id)` (optionally apply role change) and `rejectRequest(id)`; both revalidate the admin approvals page.

12) Seed Script (example)

Adjust if your schema differs. This seeds roles, users, trainings, modules, lessons, quizzes, and a few presentations.

```
// prisma/seed.ts
import { PrismaClient } from "@prisma/client";
import bcrypt from "bcryptjs";

const prisma = new PrismaClient();

async function upsertUser(email: string, role: "ADMIN"|"TRAINER"|"EMPLOYEE"|"VIEWER", name?: string)
  const passwordHash = await bcrypt.hash("password123", 10);
  return prisma.user.upsert({
    where: { email },
    update: { role },
    create: { email, name, role, password: passwordHash },
  });
}

async function main() {
  const admin    = await upsertUser("admin@example.com", "ADMIN", "Admin");
  const trainer  = await upsertUser("trainer@example.com", "TRAINER", "Trainer");
  const employee = await upsertUser("employee@example.com", "EMPLOYEE", "Employee");
  const viewer   = await upsertUser("viewer@example.com", "VIEWER", "Viewer");

  // Training skeleton
  const training = await prisma.training.upsert({
    where: { slug: "onboarding" },
    update: {},
    create: {
      slug: "onboarding",
      title: "Icecat Onboarding",
      summary: "Welcome to Icecat. Learn the basics and policies.",
      durationMinutes: 30,
      tags: ["onboarding","policies"],
      modules: {
        create: [
          {
            title: "Getting Started",
            order: 1,
            lessons: { create: [
              { title: "Company Introduction", slug: "intro",   order: 1 },
              { title: "Tools Overview",       slug: "tools",   order: 2 },
            ]}
          },
```

```
          {
            title: "Policies",
            order: 2,
            lessons: { create: [
              { title: "Security & Compliance", slug: "policies", order: 1 },
            ]}
          }
        ]
      },
      quizzes: {
        create: [{
          title: "Onboarding Quiz",
          description: "Check your understanding",
          timeLimitSeconds: 600,
          passThreshold: 70,
          questions: {
            create: [
              { type: "TRUEFALSE", prompt: "Icecat values security and compliance.", options: ["true
              { type: "SINGLE",    prompt: "Which tool is used for code hosting?", options: ["GitHub
              { type: "MULTI",     prompt: "Select collaboration tools we use.", options: ["Slack","
              { type: "SHORT",     prompt: "Type our company name.", options: [], correct: ["icecat"
              { type: "SINGLE",    prompt: "Minimum passing score for mandatory quizzes?", options:
            ]
          }
        }]
      }
    }
  });

  // Presentations examples
  await prisma.presentation.createMany({
    data: [
      { title: "Q4 Roadmap",        description: "Upcoming initiatives & milestones", path: "/files/
      { title: "Data Ops 101",      description: "Intro for new team members",       path: "/files/p
      { title: "Platform Overview", description: "Architecture & services",           path: "/files/
    ],
    skipDuplicates: true,
  });
}

main().catch((e)=>{ console.error(e); process.exit(1); })
  .finally(async ()=>{ await prisma.$disconnect(); });
```

13) One■Shot Codex Prompts You Can Paste

A) Minimal Scandinavian UI sweep (cream background, rounded cards, clean nav):

```
You are in a Next.js 14 App Router + Tailwind project.
Goal: apply a Scandinavian, minimal theme site■wide:
- Cream background (subtle), plenty of whitespace, rounded■2xl cards with soft shadow.
- Consistent typography: tight tracking on titles, readable body text.
- Sidebar nav with clear icons/labels; active item with subtle color.
- Replace any inline 'Add' forms with modal dialogs.
- Use pill filters where we switch categories/audiences.
Make changes to globals.css, layout.tsx, and shared components only; do not break routes.
```

B) Presentations: public read■only + Admin CRUD (audience Retailers/Brands):

```
Implement /presentations with audience switch pills (RETAILERS/BRANDS). Read■only grid of cards.
Create /admin/presentations with: toolbar (search + audience pills), table list, 'New presentation' m
Use Prisma model Presentation (id, title, description?, path, audience, createdAt/updatedAt). Actions
```

C) Admin shell with sidebar + user approvals:

```
Create app/admin/layout.tsx with a left sidebar (Overview, Manuals, Presentations, Documents, Quizze
Protect with role check (ADMIN/TRAINER).
Add models ApprovalStatus, ApprovalType, UserApproval and actions approveRequest/rejectRequest.
Implement /admin/approvals page showing pending requests with Approve/Reject buttons.
```

14) Deployment: Vercel + Neon (Step■by■Step)

```
# Push code to GitHub
cd "C:\Users\Rup\Desktop\Codes\icecat-learning-hub"
git init && git add -A && git commit -m "initial"
git branch -M main
git remote add origin https://github.com/<your-user>/icecat-learning-hub.git
git push -u origin main

# Create Neon DB; copy the connection string (sslmode=require).

# Vercel → New Project → Import repo
# Add envs:
DATABASE_URL=postgresql://...sslmode=require
NEXTAUTH_URL=https://<your-vercel-url>  (later change to your domain)
NEXTAUTH_SECRET=<same as local>
OPENAI_API_KEY=<if chatbot>

# After first deploy, from local:
$env:DATABASE_URL="postgresql://...sslmode=require"
npx prisma migrate deploy
npx prisma db seed

# Custom domain in Vercel; update NEXTAUTH_URL; redeploy.
```

15) Cost Pointers (rough, check provider pages)

• Vercel: starter tiers are free for hobby projects; Pro adds seats/builds. • Neon/Supabase Postgres: free tiers exist with sleep/row limits; paid plans scale with storage/connections. • Domain: typically ~€10–€20/year depending on TLD. • Optional S3 storage for files: object storage starts at a few €/month plus bandwidth.

16) Suggested Next Steps

• Add **pinning** and **badges** (Updated, Mandatory) on cards.
• Implement **search** across Manuals/Presentations/Docs with server-side filtering.
• Add **S3 upload** for documents and presentations with signed URLs.
• Internationalize UI with next-intl (messages/en.json, messages/nl.json already stubbed).
• Expand Admin sections to full CRUD using the same modal/table patterns.
• Add analytics events (progress, quiz attempts).