

Building a Modern Educational Platform

A Complete Guide from Zero to Production

Using Bun, SvelteKit, SQLite, and Modern Web Technologies

Author

SALAH AIT AMOKRANE

AFIDNA Project

December 12, 2025

Contents

1	Introduction	15
1.1	1.1 What We're Building	15
1.2	1.2 Why This Tech Stack?	15
1.2.1	The Traditional Approach (and Its Problems)	16
1.2.2	Our Modern Approach	16
1.2.3	The Result	17
1.3	1.3 Prerequisites	17
1.3.1	Technical Knowledge	17
1.3.2	Not Required (We'll Cover These)	17
1.3.3	Software Requirements	17
1.4	1.4 Project Overview	18
1.4.1	High-Level Architecture	18
1.4.2	Key Design Decisions	18
1.5	1.5 Book Structure	19
1.5.1	Part I: Foundation & Planning (Chapters 1-3)	19
1.5.2	Part II: Backend Fundamentals (Chapters 4-7)	19
1.5.3	Part III: Frontend Development (Chapters 8-11)	19
1.5.4	Part IV: Advanced Features (Chapters 12-15)	19
1.5.5	Part V: Admin Panel (Chapters 16-18)	19
1.5.6	Part VI: Content Pipeline (Chapters 19-20)	19
1.5.7	Part VII: Testing & Quality (Chapters 21-22)	20
1.5.8	Part VIII: Deployment (Chapters 23-26)	20
1.5.9	Part IX: Scaling & Optimization (Chapters 27-28)	20

1.5.10	Part X: Appendices	20
1.6	How to Use This Book	20
1.6.1	If You're Following Along	20
1.6.2	If You're Using This as Reference	20
1.6.3	Code Repository	21
1.7	Let's Begin	21
2	Architecture Design	22
2.1	2.1 Monolith vs Microservices	22
2.1.1	The Microservices Hype	22
2.1.2	The Reality Check	22
2.2	2.2 The “Compact Monolith” Philosophy	23
2.2.1	Key Principles	23
2.3	2.3 Dual Database Strategy	23
2.3.1	Why Two Databases?	24
2.3.2	The Implementation	24
2.3.3	Benefits	25
2.3.4	Connection Setup	25
2.4	2.4 Content vs User Data Separation	25
2.4.1	users.db — User-Generated Data	25
2.4.2	content.db — Educational Content	26
2.5	2.5 Designing for Scale	27
2.5.1	Scaling Strategies	27
2.5.2	Our Approach	27
2.6	2.6 Security Considerations from Day One	28
2.6.1	Authentication Security	28
2.6.2	Authorization Layers	28

2.6.3	Content Protection	28
2.6.4	Rate Limiting (Placeholder)	28
2.6.5	SQL Injection Prevention	28
2.7	Summary	29
3	Setting Up the Development Environment	30
3.1	3.1 Installing Bun	30
3.1.1	What is Bun?	30
3.1.2	Installation	30
3.1.3	Why Bun Over Node.js?	31
3.2	3.2 Editor Setup (VS Code / Cursor)	31
3.2.1	VS Code Installation	31
3.2.2	Cursor (AI-Powered Alternative)	31
3.2.3	Recommended Settings	31
3.3	3.3 Essential Extensions	32
3.3.1	Must-Have	32
3.3.2	Recommended	32
3.3.3	Installation via CLI	32
3.4	3.4 Git Configuration	33
3.4.1	Initial Configuration	33
3.4.2	SSH Key Setup (for GitHub)	33
3.4.3	Project .gitignore	33
3.5	3.5 Project Initialization	34
3.5.1	Create Project Directory	34
3.5.2	Initialize with Bun + SvelteKit	34
3.5.3	Install Dependencies	35
3.5.4	SvelteKit Configuration	35

3.5.5	Tailwind Setup	35
3.6	3.6 Directory Structure Best Practices	36
3.6.1	The Complete Structure	36
3.6.2	Directory Purposes	38
3.6.3	Creating the Structure	38
3.7	Verification Checklist	38
3.7.1	Expected Output	38
3.8	Summary	39
3.9	Quick Reference	39
3.9.1	Common Commands	39
3.9.2	Troubleshooting	40
4	Database Design with SQLite	41
4.1	4.1 Why SQLite for Production?	41
4.1.1	SQLite in Production	41
4.1.2	The Numbers	41
4.1.3	When NOT to Use SQLite	41
4.1.4	When SQLite Shines	42
4.2	4.2 Single File vs Multiple Databases	42
4.2.1	The Traditional Approach	42
4.2.2	Our Approach: Multiple Databases	42
4.2.3	Why Separate?	43
4.3	4.3 Schema Design Principles	43
4.3.1	Principle 1: Use TEXT for IDs (in content.db)	43
4.3.2	Principle 2: Use INTEGER for Auto-Increment (in users.db)	43
4.3.3	Principle 3: Store Timestamps as TEXT	43
4.3.4	Principle 4: Use INTEGER for Booleans	44

4.3.5	Principle 5: Store JSON as TEXT	44
4.4	4.4 Hierarchical Data Modeling	44
4.4.1	The Schema	44
4.4.2	Querying the Hierarchy	45
4.5	4.5 Audit Trail Implementation	46
4.5.1	The Fields	46
4.5.2	Complete Table Definition	46
4.5.3	Using the Audit Trail	46
4.6	4.6 Indexing Strategies	47
4.6.1	When to Index	47
4.6.2	Our Indexes	47
4.6.3	Composite Indexes	48
4.7	4.7 WAL Mode and Performance	48
4.7.1	Enabling WAL Mode	48
4.7.2	What WAL Does	48
4.7.3	Other Performance PRAGMAs	48
4.7.4	When to Use WAL	49
4.8	Summary	49
5	Drizzle ORM Setup	50
5.1	5.1 Introduction to Drizzle	50
5.1.1	Why Drizzle?	50
5.1.2	Core Philosophy	50
5.1.3	Installation	50
5.2	5.2 Schema Definition	51
5.2.1	users.db Schema	51
5.2.2	content.db Schema	52

5.3	5.3 Type-Safe Queries	53
5.3.1	Basic Select	53
5.3.2	Select Specific Columns	54
5.3.3	Insert	54
5.3.4	Update	54
5.3.5	Delete	54
5.4	5.4 Migrations Strategy	55
5.4.1	Why Not Traditional Migrations?	55
5.4.2	Our Approach: Setup Scripts	55
5.4.3	Schema Changes	56
5.5	5.5 Connecting Multiple Databases	56
5.5.1	Connection Module	56
5.5.2	Usage in SvelteKit	57
5.6	5.6 Query Optimization	57
5.6.1	Use Specific Selects	58
5.6.2	Use Limits	58
5.6.3	Use Joins Instead of N+1	58
5.6.4	Count Queries	58
5.6.5	Aggregations	59
5.7	Summary	59
6	Authentication System	60
6.1	6.1 Session-Based vs JWT	60
6.1.1	JWT (JSON Web Tokens)	60
6.1.2	Session-Based	60
6.1.3	Our Choice: Sessions	60
6.2	6.2 Password Hashing with Bun.password	62

6.2.1	Why Argon2id?	62
6.2.2	Implementation	62
6.2.3	Usage	62
6.3	6.3 Session Management	63
6.3.1	Session Flow	63
6.3.2	Session Schema	63
6.3.3	Session Functions	63
6.4	6.4 Cookie Security (HttpOnly, SameSite)	65
6.4.1	Cookie Options	65
6.4.2	Option Explanations	65
6.5	6.5 Remember Me Functionality	65
6.5.1	Implementation	66
6.5.2	Login Form	66
6.6	6.6 Role-Based Access Control (RBAC)	66
6.6.1	Roles	66
6.6.2	Schema	67
6.6.3	Authorization Helpers	67
6.6.4	Usage in Load Functions	67
6.7	6.7 Protected Routes with Hooks	68
6.7.1	The Hook	68
6.7.2	Type Definitions	68
6.7.3	Accessing User in Routes	69
6.7.4	Layout Data	69
6.8	Complete Auth Module	69
6.9	Summary	71
7	API Design	72

7.1	7.1 SvelteKit Server Routes	72
7.1.1	Route Types	72
7.1.2	API Routes Location	72
7.1.3	Basic Endpoint Structure	72
7.2	7.2 RESTful Endpoints	73
7.2.1	HTTP Methods	73
7.2.2	Progress API Example	73
7.3	7.3 Request Validation	75
7.3.1	Manual Validation	75
7.3.2	Validation Helper	76
7.4	7.4 Error Handling	76
7.4.1	Error Response Format	76
7.4.2	Error Responses	77
7.4.3	Try-Catch Wrapper	77
7.5	7.5 Rate Limiting	78
7.5.1	Simple In-Memory Rate Limiter	78
7.5.2	Usage in Hooks	79
7.6	7.6 CORS Configuration	79
7.6.1	When You Need CORS	79
7.6.2	SvelteKit CORS	80
7.6.3	For Our Platform	80
7.7	API Summary	80
7.8	Summary	81
8	SvelteKit Fundamentals	82
8.1	8.1 File-Based Routing	82
8.1.1	Basic Routing	82

8.1.2	Route Files	82
8.1.3	Dynamic Parameters	83
8.1.4	Multiple Parameters	83
8.1.5	Optional Parameters	83
8.1.6	Rest Parameters	83
8.2	8.2 Server vs Client Components	83
8.2.1	Server-Only Code	83
8.2.2	Client Code	84
8.2.3	Universal Code	84
8.2.4	When to Use Each	84
8.3	8.3 Load Functions	84
8.3.1	Server Load	85
8.3.2	Using Load Data	85
8.3.3	Accessing User from Locals	85
8.3.4	Dependent Data	85
8.4	8.4 Form Actions	86
8.4.1	Basic Form	86
8.4.2	Action Handler	87
8.4.3	Named Actions	87
8.4.4	Handling Action Results	87
8.4.5	Progressive Enhancement	88
8.5	8.5 Error Pages	88
8.5.1	Route-Level Error Page	88
8.5.2	Global Error Page	88
8.5.3	Throwing Errors in Load	89
8.6	8.6 Layout Nesting	89
8.6.1	Root Layout	89

8.6.2	Layout Data	90
8.6.3	Nested Layouts	90
8.6.4	Breaking Out of Layout	91
8.7	Summary	91
9	UI with DaisyUI & Tailwind	92
9.1	9.1 Setting Up Tailwind CSS 4	92
9.1.1	Installation	92
9.1.2	Vite Configuration	92
9.1.3	CSS Entry Point	92
9.1.4	Import in Layout	92
9.1.5	Tailwind Basics	93
9.2	9.2 DaisyUI Components	93
9.2.1	Installation	93
9.2.2	Configuration	93
9.2.3	Core Components	93
9.3	9.3 Theme System (Light/Dark)	95
9.3.1	Available Themes	95
9.3.2	Setting Theme	95
9.3.3	Dynamic Theme Switching	95
9.3.4	Initialize Theme	95
9.4	9.4 RTL Support for Arabic	96
9.4.1	HTML Direction	96
9.4.2	Dynamic Direction	96
9.4.3	RTL Utilities	96
9.4.4	Logical Properties (CSS)	96
9.4.5	DaisyUI RTL	97

9.5	9.5 Responsive Design Patterns	97
9.5.1	Breakpoints	97
9.5.2	Common Patterns	97
9.5.3	Mobile Navigation Pattern	98
9.6	9.6 Custom Component Library	98
9.6.1	Button Component	98
9.6.2	Card Component	99
9.6.3	Input Component	100
9.6.4	Usage	101
9.7	Summary	101
10	Building Core Pages	103
10.1	10.1 Homepage with Dynamic Stats	103
10.1.1	Load Function	103
10.1.2	Homepage Component	104
10.2	10.2 Content Listing (Tracks, Lessons)	106
10.2.1	Tracks Listing	106
10.2.2	Lessons Listing with Search	107
10.3	10.3 Detail Pages	108
10.3.1	Single Track Page	108
10.3.2	Single Lesson Page	109
10.4	10.4 Search Functionality	109
10.5	10.5 User Dashboard	110
10.6	10.6 Error and 404 Pages	111
10.7	Summary	112
11	Reusable Components	113
11.1	11.1 Navbar with Auth State	113

11.2	11.2 Footer Component	115
11.3	11.3 Card Components	116
11.3.1	LessonCard	116
11.3.2	TrackCard	117
11.4	11.4 Modal System	118
11.4.1	Usage	118
11.5	11.5 Toast Notifications	119
11.5.1	Usage with Store	120
11.6	11.6 Loading States	120
11.6.1	Skeleton Loading	121
11.6.2	Page Loading	121
11.7	Summary	121
12	Video Player Integration	123
12.1	12.1 YouTube IFrame API	123
12.1.1	Loading the API	123
12.1.2	API Reference	123
12.2	12.2 Building VideoPlayer Component	124
12.3	12.3 Progress Tracking	126
12.3.1	The 10-Minute Rule	126
12.3.2	Reporting Progress	126
12.4	12.4 Handling Events	126
12.5	12.5-12.7 Advanced Features	127
12.5.1	Quality Selection (handled by YouTube)	127
12.5.2	Fullscreen Support	127
13	Interactive Quiz System	128
13.1	13.1 Quiz Data Structure	128

13.1.1	Storing in Database	128
13.2	13.2-13.7 Quiz Component	128
14	Progress Tracking	130
14.1	14.1-14.6 Progress System	130
14.1.1	Database Schema	130
14.1.2	UPSERT Logic	130
15	Global Search	131
15.1	15.1-15.6 Search Implementation	131
15.1.1	SQLite LIKE Search	131
15.1.2	Search Page	131
16	Part V: Admin Panel	132
16.1	Chapter 16: Admin Dashboard	132
16.1.1	16.1 Access Control	132
16.1.2	16.2-16.5 Dashboard Implementation	132
16.2	Chapter 17: Content Management	133
16.2.1	17.1-17.6 CRUD Operations	133
16.3	Chapter 18: Audit & Protection	134
16.3.1	18.1-18.5 Audit Trail	134
17	Part VI: Content Pipeline	135
17.1	Chapter 19: Content Seeding	135
17.1.1	19.1-19.6 Seed Script	135
17.2	Chapter 20: Media Management	135
17.2.1	20.1 Hybrid Video Strategy	136
17.2.2	20.2 Cloudflare R2 Setup	136
17.2.3	20.3 File Structure	136

17.2.4 20.4 Database Integration	136
17.2.5 20.5 No SDK Needed!	136
17.2.6 20.6 Upload via rclone	137
17.2.7 20.7 Future: Admin Upload (مُوجِل)	137
18 Part VII: Testing & Quality	138
18.1 Chapter 21: Testing Strategy	138
18.1.1 Unit Tests with Bun	138
18.1.2 Run Tests	138
18.2 Chapter 22: Code Quality	138
18.2.1 ESLint + Prettier	138
19 Part VIII: Deployment	140
19.1 Chapter 23-26: Production Deployment	140
19.1.1 Build	140
19.1.2 VPS Deployment	140
19.1.3 Caddy Config	140
20 Part IX: Scaling & Optimization	141
20.1 Chapter 27-28: Performance	141
20.1.1 Database Optimization	141
20.1.2 Caching	141
21 Part X: Appendices	142
21.1 Appendix A: Code Reference	142
21.2 Appendix B: Commands	142
21.3 Appendix C: Troubleshooting	142
21.4 Appendix D: Resources	142

Chapter 1

Introduction

1.1 1.1 What We're Building

Welcome to this comprehensive guide on building a modern educational platform from scratch. Throughout this book, we'll create **Afidna** — a fully-featured learning management system (LMS) designed for Islamic scholarly content.

By the end of this journey, you'll have built:

- **A content delivery system** with hierarchical organization (Tracks → Series → Lessons → Videos)
- **User authentication** with secure session management
- **Progress tracking** that saves watch time and quiz scores
- **An interactive quiz system** with instant feedback
- **A video player** integrated with YouTube's API
- **An admin panel** for content management
- **A global search** across all content
- **A responsive UI** supporting both LTR and RTL languages

This isn't a toy project or a simple tutorial app. We're building production-ready software that can serve thousands of users.

1.2 1.2 Why This Tech Stack?

1.2.1 The Traditional Approach (and Its Problems)

Many educational platforms are built with:

- **Python/Django** or **Node/Express** for the backend
- **React** or **Vue** for the frontend
- **PostgreSQL** or **MySQL** for the database
- **Redis** for sessions
- **Nginx** for reverse proxy

This stack works, but it comes with complexity:

- Multiple services to manage
- High memory footprint
- Complex deployment
- Expensive hosting

1.2.2 Our Modern Approach

We're taking a different path — the “**Compact Monolith**” philosophy:

Component	Our Choice	Why
Runtime	Bun	4x faster than Node.js, built-in bundler, native TypeScript
Framework	SvelteKit	SSR + SPA in one, excellent DX, small bundle size
Database	SQLite	Zero configuration, single file, surprisingly capable
ORM	Drizzle	Type-safe, lightweight, great SQLite support
UI	DaisyUI + Tailwind	Beautiful components, rapid development
Auth	Session-based	Simple, secure, no JWT complexity

1.2.3 The Result

- **Single process** to deploy
 - **< 100MB RAM** in production
 - **Sub-millisecond** database queries
 - **\$5/month** VPS is more than enough
 - **Full-stack TypeScript** — same language everywhere
-

1.3 1.3 Prerequisites

Before diving in, make sure you have:

1.3.1 Technical Knowledge

- **JavaScript/TypeScript**: Comfortable with modern ES6+ syntax
- **HTML/CSS**: Basic understanding of web markup
- **Command Line**: Comfortable with terminal basics
- **Git**: Basic version control knowledge

1.3.2 Not Required (We'll Cover These)

- Svelte/SvelteKit experience
- SQLite knowledge
- Backend development experience
- DevOps skills

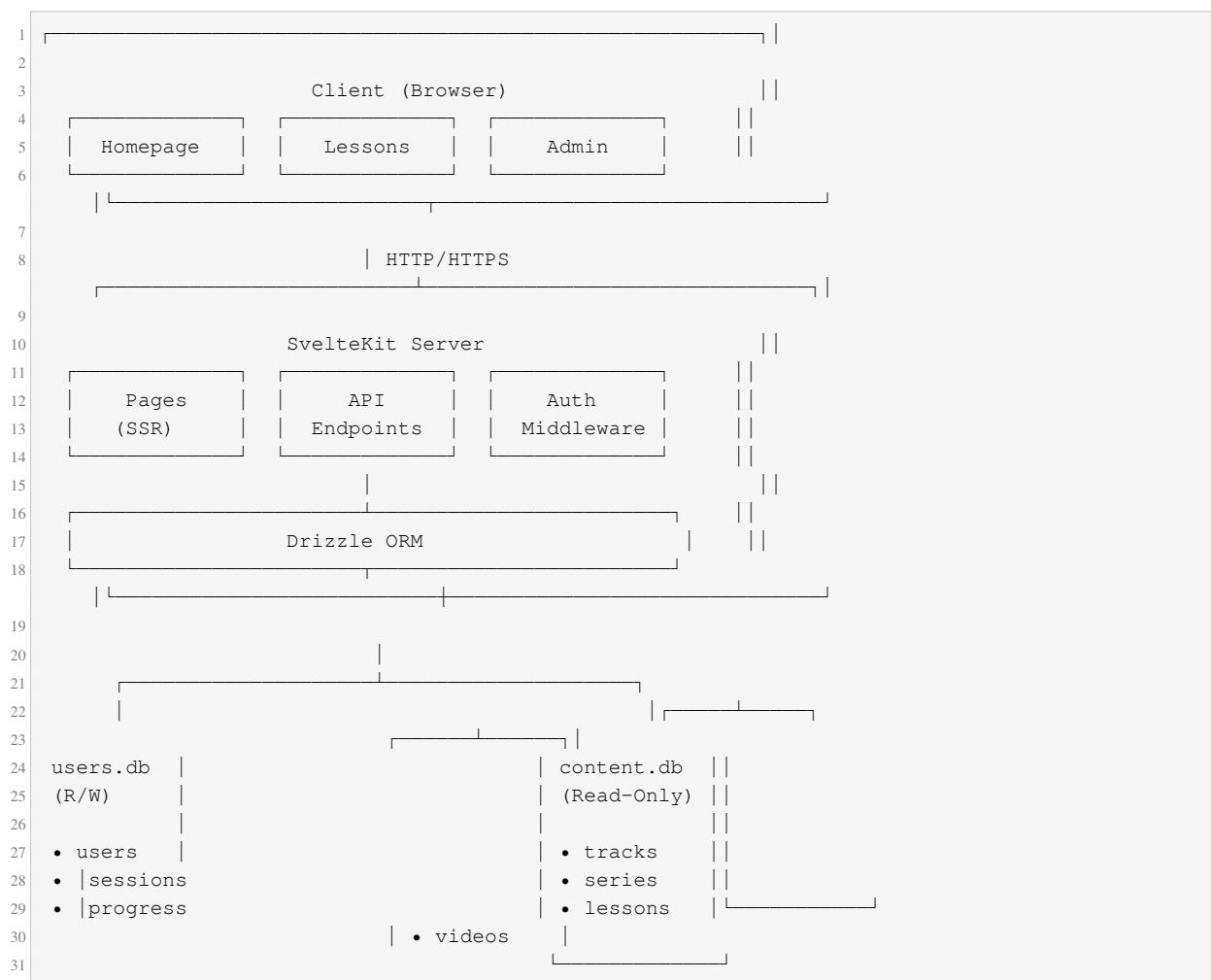
1.3.3 Software Requirements

- **Operating System**: Linux, macOS, or Windows (WSL recommended)
- **Node.js**: Version 18+ (for some tooling)
- **Bun**: We'll install this together
- **Code Editor**: VS Code or Cursor recommended

- **Git:** For version control
-

1.4 1.4 Project Overview

1.4.1 High-Level Architecture



1.4.2 Key Design Decisions

1. **Dual Database Strategy:** Separate user data from content
2. **Hierarchical Content:** Tracks → Series → Lessons → Videos
3. **Human-First Editing:** Protect manual edits from automated imports
4. **Progressive Enhancement:** Works without JavaScript, better with it

5. **Mobile-First Design:** Responsive from the start
-

1.5 1.5 Book Structure

This book is organized into **10 parts**:

1.5.1 Part I: Foundation & Planning (Chapters 1-3)

Setting up your environment and understanding the architecture.

1.5.2 Part II: Backend Fundamentals (Chapters 4-7)

Database design, ORM setup, authentication, and API development.

1.5.3 Part III: Frontend Development (Chapters 8-11)

SvelteKit basics, UI components, and building core pages.

1.5.4 Part IV: Advanced Features (Chapters 12-15)

Video player, quiz system, progress tracking, and search.

1.5.5 Part V: Admin Panel (Chapters 16-18)

Building a complete content management interface.

1.5.6 Part VI: Content Pipeline (Chapters 19-20)

Automated content import and media management.

1.5.7 Part VII: Testing & Quality (Chapters 21-22)

Testing strategies and code quality tools.

1.5.8 Part VIII: Deployment (Chapters 23-26)

Production builds, server setup, and monitoring.

1.5.9 Part IX: Scaling & Optimization (Chapters 27-28)

Performance tuning and growth strategies.

1.5.10 Part X: Appendices

Reference materials and troubleshooting guides.

1.6 How to Use This Book

1.6.1 If You're Following Along

1. **Type the code yourself** — don't just copy-paste
2. **Run the code after each section** — see it work
3. **Experiment** — break things, fix them
4. **Take notes** — document your learnings

1.6.2 If You're Using This as Reference

- Each chapter is self-contained
- Code examples are complete and runnable
- The appendices have quick reference guides

1.6.3 Code Repository

All code from this book is available at:

1 <https://github.com/salahAit/afidna-Sveltekit>

You can clone it to compare with your work or use it as a starting point.

1.7 Let's Begin

In the next chapter, we'll dive deep into architecture design — understanding why we make certain decisions and how they impact our application's future.

Get your development environment ready. We're about to build something amazing.

Next Chapter: [Chapter 2: Architecture Design](#)

Chapter 2

Architecture Design

2.1 2.1 Monolith vs Microservices

Before writing any code, we need to make a fundamental decision about our application's structure.

2.1.1 The Microservices Hype

You've probably heard that "microservices are the future" and "monoliths are dead." Let's examine this claim.

Microservices architecture means splitting your application into many small, independent services:

Advantages: - Independent scaling - Technology flexibility - Team independence - Fault isolation

Disadvantages: - Network latency between services - Distributed system complexity - Operational overhead - Debugging nightmares - Higher infrastructure costs

2.1.2 The Reality Check

Netflix, Amazon, and Google use microservices because they have: - Thousands of developers - Millions of users - Billions in revenue to pay for infrastructure

For our educational platform, we have: - 1-5 developers - Hundreds to thousands of users - A budget that prefers efficiency

The truth: Most applications don't need microservices. They add complexity that doesn't pay off until you're operating at massive scale.

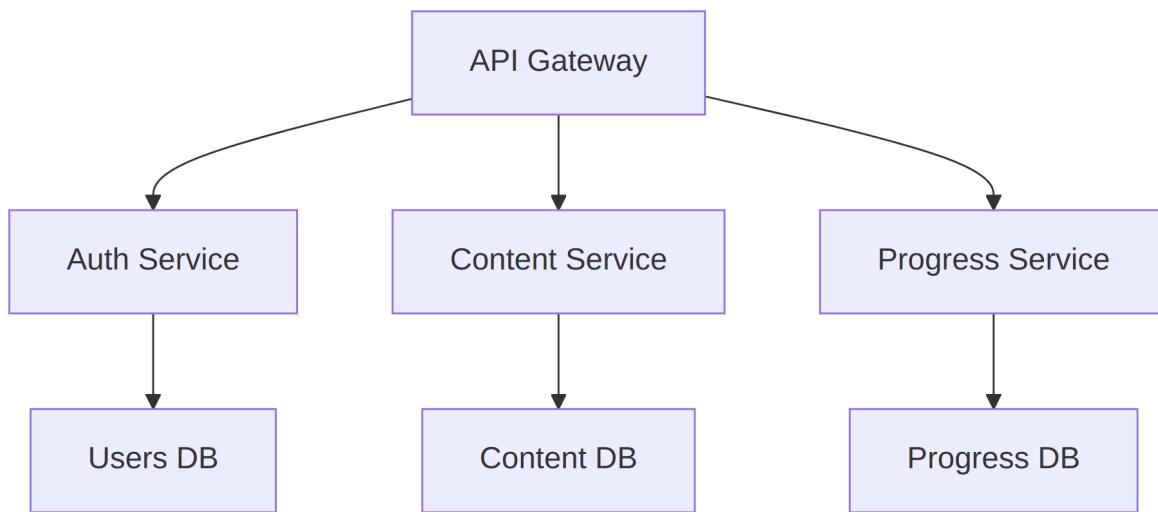


Figure 2.1: Microservices Architecture

2.2 2.2 The “Compact Monolith” Philosophy

We’re going to build what I call a “**Compact Monolith**” — a single, well-structured application that’s:

- **Simple to develop:** One codebase, one language
- **Simple to deploy:** One process to run
- **Simple to scale:** Optimize what matters
- **Easy to change:** Refactor to microservices later if needed

2.2.1 Key Principles

1. **Modular by Design:** Clear separation of concerns
2. **Vertical Slices:** Each feature is self-contained
3. **Shared Infrastructure:** One database connection, one auth system
4. **No Premature Optimization:** Build simple, optimize when needed

2.3 2.3 Dual Database Strategy

One unique aspect of our architecture is using **two separate SQLite databases**.

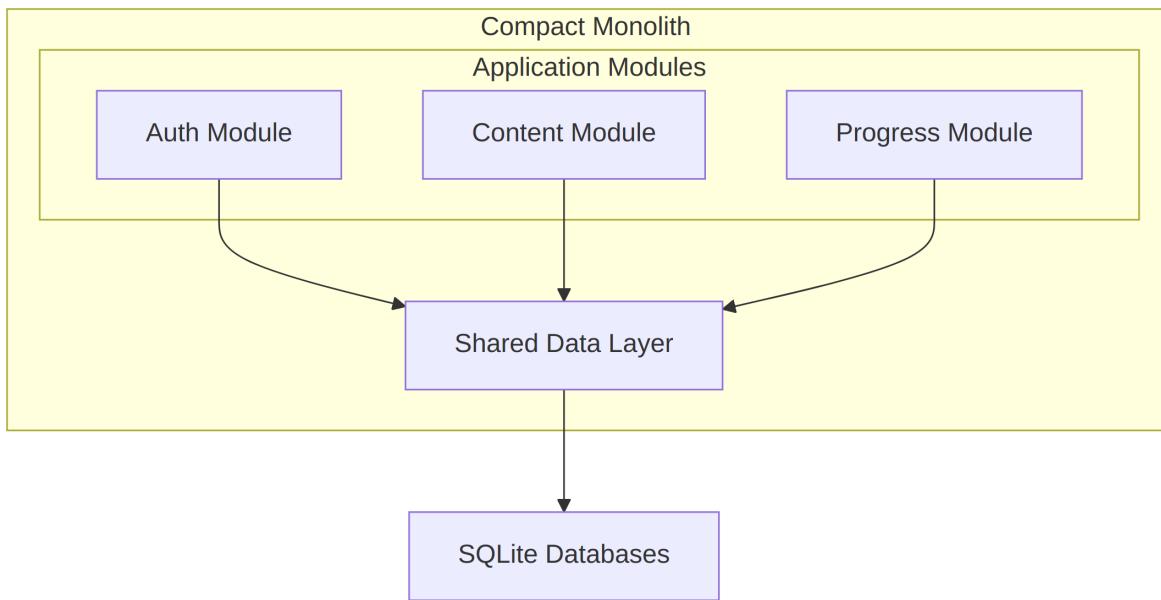


Figure 2.2: Compact Monolith Architecture

2.3.1 Why Two Databases?

Consider the nature of our data:

Aspect	User Data	Content Data
Changes	Frequently (every user action)	Rarely (admin updates)
Size	Grows with users	Fixed (course catalog)
Backup	Critical (user progress)	Less critical (can regenerate)
Access	Read/Write	Mostly Read

2.3.2 The Implementation

```

1 data/|__  

2   users.db      # User-generated data|  

3     |__ users      # Account information|  

4     |__ sessions    # Login sessions|  

5     |__ lesson_progress # Watch history, quiz scores|__  

6  

7   content.db    # Educational content  

8     |__ tracks      # Learning paths  

9     |__ series       # Course series  

10    |__ lessons      # Individual lessons  

11    |__ videos       # Video segments

```

2.3.3 Benefits

1. **Independent Backups:** Backup user data more frequently
2. **Easy Content Updates:** Replace content.db without affecting users
3. **Performance Isolation:** Heavy reads don't affect writes
4. **Development Simplicity:** Clear ownership of data

2.3.4 Connection Setup

```

1 // src/lib/server/db/index.ts
2
3 import { drizzle } from 'drizzle-orm/bun-sqlite';
4 import { Database } from 'bun:sqlite';
5
6 // User database (read-write)
7 const usersSqlite = new Database('data/users.db');
8 usersSqlite.exec('PRAGMA journal_mode = WAL');
9 usersSqlite.exec('PRAGMA foreign_keys = ON');
10
11 export const db = drizzle(usersSqlite);
12
13 // Content database (read-mostly)
14 const contentSqlite = new Database('data/content.db');
15 contentSqlite.exec('PRAGMA foreign_keys = ON');
16
17 export const contentDatabase = drizzle(contentSqlite);

```

2.4 2.4 Content vs User Data Separation

Let's define exactly what goes where.

2.4.1 users.db — User-Generated Data

Everything that comes from user actions:

```

1 -- Who are our users?
2 CREATE TABLE users (
3     id INTEGER PRIMARY KEY,
4     email TEXT UNIQUE NOT NULL,
5     password_hash TEXT NOT NULL,
6     name TEXT NOT NULL,
7     role TEXT DEFAULT 'user', -- user, editor, admin
8     created_at TEXT DEFAULT CURRENT_TIMESTAMP
9 );

```

```

11 -- Active login sessions
12 CREATE TABLE sessions (
13     id TEXT PRIMARY KEY,    -- UUID
14     user_id INTEGER REFERENCES users(id),
15     expires_at TEXT NOT NULL
16 );
17
18 -- Learning progress
19 CREATE TABLE lesson_progress (
20     id INTEGER PRIMARY KEY,
21     user_id INTEGER REFERENCES users(id),
22     lesson_id TEXT NOT NULL,    -- References content.db
23     watched_seconds INTEGER DEFAULT 0,
24     video_completed INTEGER DEFAULT 0,
25     quiz_passed INTEGER DEFAULT 0,
26     quiz_score INTEGER DEFAULT 0
27 );

```

2.4.2 content.db — Educational Content

All the learning material:

```

1 -- Learning tracks (e.g., "Islamic Creed", "Hadith Studies")
2 CREATE TABLE tracks (
3     id TEXT PRIMARY KEY,    -- e.g., "aqeedah", "hadith"
4     title TEXT NOT NULL,
5     description TEXT,
6     icon TEXT,
7     "order" INTEGER DEFAULT 0
8 );
9
10 -- Series within tracks
11 CREATE TABLE series (
12     id TEXT PRIMARY KEY,
13     track_id TEXT REFERENCES tracks(id),
14     title TEXT NOT NULL,
15     instructor TEXT,
16     is_locked INTEGER DEFAULT 0
17 );
18
19 -- Individual lessons
20 CREATE TABLE lessons (
21     id TEXT PRIMARY KEY,
22     track_id TEXT REFERENCES tracks(id),
23     series_id TEXT REFERENCES series(id),
24     title TEXT NOT NULL,
25     slug TEXT UNIQUE NOT NULL,
26     is_published INTEGER DEFAULT 0
27 );
28
29 -- Video segments within lessons
30 CREATE TABLE videos (
31     id TEXT PRIMARY KEY,
32     lesson_id TEXT REFERENCES lessons(id),
33     title TEXT,
34     youtube_id TEXT,
35     duration INTEGER,
36     quiz TEXT    -- JSON array of questions
37 );

```

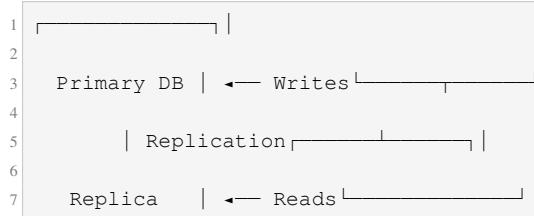
2.5 2.5 Designing for Scale

Even though we're building a monolith, we should design for growth.

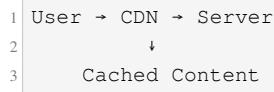
2.5.1 Scaling Strategies

1. Vertical Scaling (Scale Up) - Add more CPU/RAM to the server - Works until ~10,000 concurrent users - Simple and effective

2. Read Replicas



3. Edge Caching



4. Eventually, Microservices If we grow to millions of users, we can extract services: - Auth → Separate service - Video processing → Separate service - Search → ElasticSearch cluster

2.5.2 Our Approach

We'll build with these principles:

1. **Stateless requests:** Any server can handle any request
 2. **Database as source of truth:** No in-memory state
 3. **Cacheable responses:** Proper HTTP headers
 4. **Modular code:** Easy to extract later
-

2.6 2.6 Security Considerations from Day One

Security isn't an afterthought — it's baked into our architecture.

2.6.1 Authentication Security

```

1 Password Flow:
2 User Input → bcrypt hash → Store hash
3 Login:      User Input → bcrypt verify → Session token

```

Key decisions: - **bcrypt** for password hashing (not MD5, not SHA1) - **Session tokens** stored in HttpOnly cookies - **SameSite=Lax** to prevent CSRF - **Secure flag** in production

2.6.2 Authorization Layers

```

1 Request Flow:
2 1. hooks.server.ts → Validate session
3 2. +page.server.ts → Check role
4 3. Database query → Filter by user

```

2.6.3 Content Protection

```

1 // Audit trail on every content table
2 {
3     created_at: TEXT,           // When created
4     updated_at: TEXT,          // When modified
5     last_modified_by: TEXT,   // Who modified
6     is_locked: INTEGER        // Prevent changes
7 }

```

2.6.4 Rate Limiting (Placeholder)

```

1 // hooks.server.ts
2 if (url.pathname.startsWith('/api/auth')) {
3     // TODO: Implement rate limiting
4     // 5 requests per minute per IP
5 }

```

2.6.5 SQL Injection Prevention

Drizzle ORM uses parameterized queries automatically:

```
1 // Safe - Drizzle handles escaping
2 db.select()
3   .from(users)
4   .where(eq(users.email, userInput));
5
6 // NEVER do this
7 db.run(`SELECT * FROM users WHERE email = '${userInput}'`);
```

2.7 Summary

In this chapter, we established our architectural foundation:

Decision	Choice	Reason
Architecture	Compact Monolith	Simplicity, low overhead
Database	Dual SQLite	Separation of concerns
Auth	Session-based	Simple, secure
Scaling	Start small, grow smart	Optimize when needed
Security	Defense in depth	Built-in from day one

These decisions will guide every choice we make throughout the book. They're not set in stone — if your needs differ, adapt them. But for most educational platforms, this architecture will serve you well.

Next Chapter: [Chapter 3: Setting Up the Development Environment](#)

Chapter 3

Setting Up the Development Environment

3.1 3.1 Installing Bun

Bun is our JavaScript runtime — think of it as a faster, more modern alternative to Node.js.

3.1.1 What is Bun?

Bun is an all-in-one toolkit that includes:

- **Runtime:** Executes JavaScript/TypeScript
- **Package Manager:** Faster than npm/yarn
- **Bundler:** Built-in bundling for production
- **Test Runner:** Native testing support

3.1.2 Installation

macOS / Linux:

```
1 curl -fsSL https://bun.sh/install | bash
```

Windows (via WSL):

```
1 # First, install WSL if you haven't
2 wsl --install
3
4 # Then install Bun in WSL
5 curl -fsSL https://bun.sh/install | bash
```

Verify installation:

```
1 bun --version
2 # Should output: 1.x.x
```

3.1.3 Why Bun Over Node.js?

Feature	Node.js	Bun
Startup time	~40ms	~5ms
Package install	~10s	~2s
TypeScript	Needs transpiler	Native
SQLite	Needs npm package	Built-in
Test runner	Needs Jest/Vitest	Built-in

3.2 3.2 Editor Setup (VS Code / Cursor)

A good editor setup will save you hours of debugging.

3.2.1 VS Code Installation

Download from: <https://code.visualstudio.com/>

3.2.2 Cursor (AI-Powered Alternative)

Cursor is VS Code with AI built-in: <https://cursor.sh/>

3.2.3 Recommended Settings

Create `.vscode/settings.json` in your project:

```

1 {
2     // Editor basics
3     "editor.tabSize": 4,
4     "editor.formatOnSave": true,
5     "editor.defaultFormatter": "esbenp.prettier-vscode",
6
7     // TypeScript
8     "typescript.preferences.importModuleSpecifier": "relative",
9     "typescript.suggest.autoImports": true,
10
11    // Svelte
12    "svelte.enable-ts-plugin": true,
13
14    // File associations
15    "files.associations": {

```

```
16     ".*.svelte": "svelte"
17 },
18
19 // Exclude from search
20 "search.exclude": {
21     "**/node_modules": true,
22     "**/dist": true,
23     "**/.svelte-kit": true
24 }
25 }
```

3.3 Essential Extensions

Install these VS Code extensions:

3.3.1 Must-Have

Extension	Purpose
Svelte for VS Code	Svelte language support
Tailwind CSS IntelliSense	Tailwind autocomplete
Prettier	Code formatting
ESLint	Code linting
Error Lens	Inline error display

3.3.2 Recommended

Extension	Purpose
GitLens	Git blame and history
SQLite Viewer	View .db files
Thunder Client	API testing
Todo Tree	Track TODOs

3.3.3 Installation via CLI

```
1 # Install all at once
2 code --install-extension svelte.svelte-vscode
3 code --install-extension bradlc.vscode-tailwindcss
4 code --install-extension esbenp.prettier-vscode
5 code --install-extension dbaeumer.vscode-eslint
6 code --install-extension usernamehw.errorlens
7 code --install-extension eamodio.gitlens
8 code --install-extension qwtel.sqlite-viewer
```

3.4 3.4 Git Configuration

Version control is essential. Let's set it up properly.

3.4.1 Initial Configuration

```
1 # Set your identity
2 git config --global user.name "Your Name"
3 git config --global user.email "your@email.com"
4
5 # Better defaults
6 git config --global init.defaultBranch main
7 git config --global pull.rebase true
8 git config --global core.autocrlf input # Linux/Mac
```

3.4.2 SSH Key Setup (for GitHub)

```
1 # Generate SSH key
2 ssh-keygen -t ed25519 -C "your@email.com"
3
4 # Start SSH agent
5 eval "$(ssh-agent -s)"
6 ssh-add ~/.ssh/id_ed25519
7
8 # Copy public key
9 cat ~/.ssh/id_ed25519.pub
10 # Paste this in GitHub → Settings → SSH Keys
```

3.4.3 Project .gitignore

Create `.gitignore`:

```
1 # Dependencies
2 node_modules/
3 .pnpm-store/
```

```
4
5 # Build outputs
6 .svelte-kit/
7 build/
8 dist/
9
10 # Environment
11 .env
12 .env.local
13 .env.*.local
14
15 # Databases (optional: you might want to track content.db)
16 data/*.db
17 data/*.db-wal
18 data/*.db-shm
19
20 # Editor
21 .vscode/*
22 ! .vscode/settings.json
23 ! .vscode/extensions.json
24 .idea/
25 *.swp
26 *.swo
27
28 # OS
29 .DS_Store
30 Thumbs.db
31
32 # Logs
33 *.log
34 npm-debug.log*
```

3.5 3.5 Project Initialization

Let's create our project from scratch.

3.5.1 Create Project Directory

```
1 mkdir afidna
2 cd afidna
```

3.5.2 Initialize with Bun + SvelteKit

```
1 # Create SvelteKit project
2 bun create svelte@latest .
3
4 # When prompted, choose:
5 # - Skeleton project
6 # - TypeScript
```

```

7 # - ESLint: Yes
8 # - Prettier: Yes
9 # - Playwright: No (we'll add later)
10 # - Vitest: No (we'll use Bun's test runner)

```

3.5.3 Install Dependencies

```

1 # Core dependencies (Bun Native - no external SQLite/bcrypt needed!)
2 bun add drizzle-orm
3
4 # Development dependencies
5 bun add -d drizzle-kit @types/bun
6
7 # UI dependencies
8 bun add -d tailwindcss @tailwindcss/vite daisyui

```

Note: We use `bun:sqlite` (built-in) instead of `better-sqlite3`, and `Bun.password` instead of `bcrypt` for 25x faster hashing!

3.5.4 SvelteKit Configuration

Update `svelte.config.js`:

```

1 import adapter from '@sveltejs/adapter-node';
2 import { vitePreprocess } from '@sveltejs/vite-plugin-svelte';
3
4 /** @type {import('@sveltejs/kit').Config} */
5 const config = {
6    preprocess: vitePreprocess(),
7    kit: {
8        adapter: adapter(),
9        alias: {
10            $components: 'src/lib/components',
11            $server: 'src/lib/server'
12        }
13    }
14};
15
16 export default config;

```

3.5.5 Tailwind Setup

Create `src/app.css`:

```

1 @import 'tailwindcss';
2 @plugin 'daisyui';
3
4 /* RTL Support */
5 [dir="rtl"] {
6     text-align: right;

```

```

7 }
8
9 /* Custom utilities */
10 @layer utilities {
11     .card-hover {
12         @apply transition-transform hover:scale-[1.02];
13     }
14 }
```

Update `vite.config.ts`:

```

1 import { sveltekit } from '@sveltejs/kit/vite';
2 import tailwindcss from '@tailwindcss/vite';
3 import { defineConfig } from 'vite';
4
5 export default defineConfig({
6     plugins: [
7         tailwindcss(),
8         sveltekit()
9     ],
10    server: {
11        fs: {
12            allow: ['.'] // لفقات بقراط لملفات .db
13        }
14    },
15    optimizeDeps: {
16        exclude: ['bun:sqlite'] // في vite.js يجده مكتوب حجزه في أصل الملف
17    },
18    build: {
19        rollupOptions: {
20            external: ['bun:sqlite'] // عند إنشاء نسخة لبناء
21        }
22    }
23});
```

3.6 3.6 Directory Structure Best Practices

Let's organize our project for maintainability.

3.6.1 The Complete Structure

```

1 afidna/
2   src/
3     lib/                      # Shared code|
4       components/           # Svelte components|
5         Navbar.svelte|
6         Footer.svelte|
7         VideoPlayer.svelte|
8         Quiz.svelte|
9       server/                 # Server-only code|
10      auth.ts                # Authentication logic|
```

```
11 |     |     └── db/|
12 |     |         ├── index.ts      # Database connections|
13 |     |         ├── schemas.ts    # users.db schema|
14 |     |         └── schema-content.ts|
15 |     ├── stores/              # Svelte stores|
16 |     |         └── user.ts|
17 |     ├── utils/               # Utility functions|
18 |     |         └── format.ts|
19 |     └── theme.ts            # Theme configuration|
20 ||
21 └── routes/                # SvelteKit routes|
22     ├── +layout.svelte        # Root layout|
23     ├── +layout.server.ts    # Layout data|
24     ├── +page.svelte          # Homepage|
25     ├── +page.server.ts|
26     ├── tracks/              # /tracks routes|
27     ├── lessons/             # /lessons routes|
28     ├── search/              # /search route|
29     ├── auth/                # /auth routes|
30     |   ├── login/|
31     |   ├── register/|
32     |   └── logout/|
33     ├── admin/               # /admin routes|
34     |   ├── +page.svelte|
35     |   ├── lessons/|
36     |   ├── videos/|
37     |   └── series/|
38     └── api/                 # API routes|
39         └── progress/|
40 ||
41 └── hooks.server.ts        # Server hooks (middleware)|
42 └── app.html                # HTML template|
43 └── app.css                 # Global styles|
44 └── app.d.ts                # Type declarations|——
45
46 scripts/                  # Management scripts|
47     ├── setup-db.ts          # Create users.db|
48     ├── setup-content.ts     # Create content.db|
49     └── seed-content.ts      # Import content|——
50
51 data/                      # Database files|
52     ├── users.db|
53     └── content.db|——
54
55 raw_content/               # Content source files|
56     └── content.json|——
57
58 static/                    # Static assets|
59     ├── favicon.png|
60     └── images/|——
61
62 book/                      # This book!|——
63
64 .gitignore|——
65 package.json|——
66 svelte.config.js|——
67 tsconfig.json|——
68 vite.config.ts|——
69 README.md
```

3.6.2 Directory Purposes

Directory	Purpose	Access
src/lib/components/	Reusable UI components	Client + Server
src/lib/server/	Server-only code	Server only
src/lib/stores/	Reactive state	Client only
src/routes/	Pages and API	Both
scripts/	CLI utilities	Server only
data/	Database files	Server only
static/	Static files	Public

3.6.3 Creating the Structure

```

1 # Create all directories
2 mkdir -p src/lib/{components,server/db,stores,utils}
3 mkdir -p src/routes/{tracks,lessons,search,auth/{login,register,logout}}
4 mkdir -p src/routes/admin/{lessons,videos,series}
5 mkdir -p src/routes/api/progress
6 mkdir -p scripts
7 mkdir -p data
8 mkdir -p raw_content
9 mkdir -p static/images

```

3.7 Verification Checklist

Before moving on, verify your setup:

```

1 # Check Bun
2 bun --version      # Should be 1.x.x
3
4 # Check project runs
5 bun run dev       # Should start on localhost:5173
6
7 # Check TypeScript
8 bun run check     # Should pass
9
10 # Check Git
11 git status        # Should show your project

```

3.7.1 Expected Output

When you run `bun run dev`, you should see:

```
1 VITE v5.x.x  ready in xxx ms
2
3 → Local:   http://localhost:5173/
4 → Network: use --host to expose
5 → press h + enter to show help
```

3.8 Summary

In this chapter, we set up:

- ✓ Bun runtime
- ✓ VS Code with essential extensions
- ✓ Git with proper configuration
- ✓ SvelteKit project
- ✓ Tailwind CSS + DaisyUI
- ✓ Organized directory structure

Our development environment is now ready. In the next part, we'll start building the backend — beginning with database design.

Next Chapter: Part II → Chapter 4: Database Design with SQLite

3.9 Quick Reference

3.9.1 Common Commands

```
1 # Start development server
2 bun run dev
3
4 # Type checking
5 bun run check
```

```
6
7 # Format code
8 bun run format
9
10 # Build for production
11 bun run build
12
13 # Preview production build
14 bun run preview
```

3.9.2 Troubleshooting

Problem: command not found: bun **Solution:** Restart your terminal or run source ~/.bashrc

Problem: Port 5173 already in use **Solution:** Kill the process or use bun run dev -- --port 3000

Problem: TypeScript errors in editor **Solution:** Restart TypeScript server (Cmd/Ctrl + Shift + P → “TypeScript: Restart TS Server”)

Chapter 4

Database Design with SQLite

4.1 Why SQLite for Production?

SQLite is often dismissed as “just for prototyping.” This is a myth.

4.1.1 SQLite in Production

These companies use SQLite in production:

Company	Use Case
Apple	Every iPhone, iPad, Mac
Airbus	Flight software
Dropbox	Desktop client
WhatsApp	Message storage
Adobe	Creative Cloud

4.1.2 The Numbers

SQLite handles:
- **100,000+** simultaneous readers
- **10,000+** writes per second
- **281 TB** maximum database size
- **ACID** compliant transactions

4.1.3 When NOT to Use SQLite

- High-write workloads with many concurrent writers
- Need for replication across multiple servers
- Very large datasets (> 1TB)

- Real-time synchronization requirements

4.1.4 When SQLite Shines

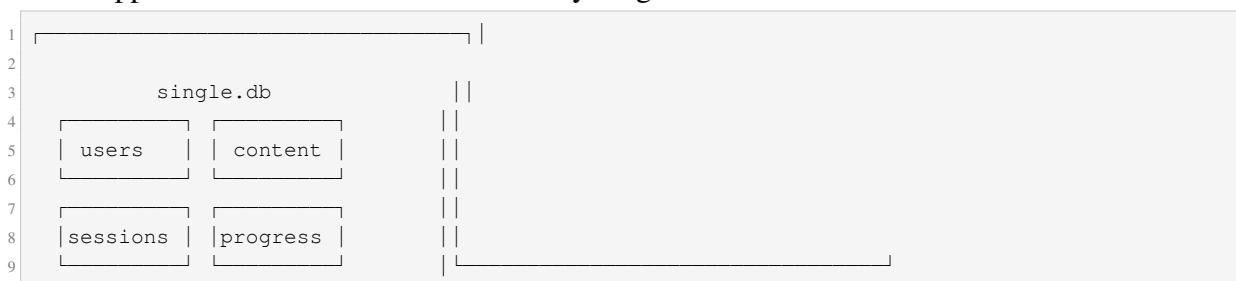
- Read-heavy workloads (like content delivery)
 - Single-server deployments
 - Embedded applications
 - Simplicity is a priority

Our educational platform is **read-heavy** (many users reading content, few admins writing). SQLite is perfect.

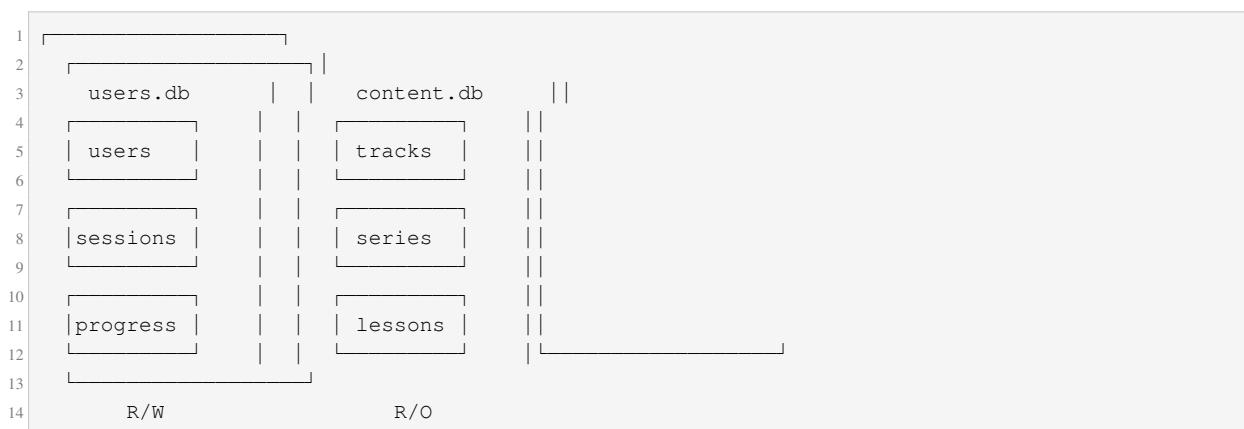
4.2 4.2 Single File vs Multiple Databases

4.2.1 The Traditional Approach

Most applications use one database for everything:



4.2.2 Our Approach: Multiple Databases



4.2.3 Why Separate?

Benefit	Explanation
Independent Backups	Backup user data hourly, content weekly
Easy Updates	Replace content.db without touching users
Clear Ownership	Content team manages content.db
Performance	Content reads don't lock user writes
Security	Content can be read-only in production

4.3 4.3 Schema Design Principles

Good schema design prevents pain later.

4.3.1 Principle 1: Use TEXT for IDs (in content.db)

```

1 -- Instead of this:
2 CREATE TABLE lessons (
3     id INTEGER PRIMARY KEY -- What is lesson 42?
4 );
5
6 -- Do this:
7 CREATE TABLE lessons (
8     id TEXT PRIMARY KEY      -- lesson "hadith-jibril" is clear
9 );

```

Why? Human-readable IDs are:

- Self-documenting
- Easier to debug
- URL-friendly (slugs)
- Content-addressable

4.3.2 Principle 2: Use INTEGER for Auto-Increment (in users.db)

```

1 -- For user-generated data, auto-increment is fine:
2 CREATE TABLE users (
3     id INTEGER PRIMARY KEY AUTOINCREMENT
4 );

```

Why? User IDs don't need to be memorable.

4.3.3 Principle 3: Store Timestamps as TEXT

```

1 -- SQLite has no native date type
2 created_at TEXT DEFAULT CURRENT_TIMESTAMP
3 -- Stores as: "2024-12-11 20:00:00"

```

Why? ISO 8601 strings are:

- Human-readable
- Sortable as text
- Timezone-aware compatible

4.3.4 Principle 4: Use INTEGER for Booleans

```

1 -- SQLite has no native boolean
2 is_published INTEGER DEFAULT 0 -- 0 = false, 1 = true

```

4.3.5 Principle 5: Store JSON as TEXT

```

1 -- For complex data like quizzes
2 quiz TEXT -- JSON array of questions

```

Parse in application code:

```

1 const questions = JSON.parse(video.quiz || '[]');

```

4.4 4.4 Hierarchical Data Modeling

Our content has a natural hierarchy:

```

1 Track (e.g., "Hadith Studies")
2   └ Series (e.g., "40 Nawawi Hadith")
3     └ Lesson (e.g., "Hadith of Jibril")
4       └ Video (e.g., "Part 1: Introduction")

```

4.4.1 The Schema

```

1 -- Level 1: Tracks (Learning Paths)
2 CREATE TABLE tracks (
3   id TEXT PRIMARY KEY,          -- "hadith"
4   title TEXT NOT NULL,         -- "Hadith Studies"
5   description TEXT,
6   icon TEXT,                  -- 📚
7   "order" INTEGER DEFAULT 0
8 );
9
10 -- Level 2: Series (Course Collections)
11 CREATE TABLE series (
12   id TEXT PRIMARY KEY,        -- "nawawi-40"

```

```

13    track_id TEXT REFERENCES tracks(id),
14    title TEXT NOT NULL,           -- "40 Nawawi Hadith"
15    description TEXT,
16    instructor TEXT,             -- "Sheikh X"
17    thumbnail_url TEXT,
18    "order" INTEGER DEFAULT 0
19 );
20
21 -- Level 3: Lessons (Individual Classes)
22 CREATE TABLE lessons (
23     id TEXT PRIMARY KEY,         -- "hadith-jibril"
24     track_id TEXT REFERENCES tracks(id),
25     series_id TEXT REFERENCES series(id),
26     title TEXT NOT NULL,
27     slug TEXT UNIQUE NOT NULL,   -- URL-friendly
28     description TEXT,
29     instructor TEXT,
30     pdf_url TEXT,
31     is_published INTEGER DEFAULT 0,
32     "order" INTEGER DEFAULT 0
33 );
34
35 -- Level 4: Videos (Lesson Segments)
36 CREATE TABLE videos (
37     id TEXT PRIMARY KEY,         -- "hadith-jibril-v1"
38     lesson_id TEXT REFERENCES lessons(id),
39     title TEXT,
40     youtube_id TEXT,            -- YouTube video ID
41     download_url TEXT,          -- Direct download link
42     duration INTEGER,           -- Seconds
43     summary TEXT,
44     quiz TEXT,                  -- JSON array
45     "order" INTEGER DEFAULT 0
46 );

```

4.4.2 Querying the Hierarchy

Get full lesson with related data:

```

1 // Get lesson with track info
2 const lesson = db
3     .select()
4     .from(lessons)
5     .leftJoin(tracks, eq(lessons.trackId, tracks.id))
6     .where(eq(lessons.slug, 'hadith-jibril'))
7     .get();
8
9 // Get videos for lesson
10 const videos = db
11     .select()
12     .from(videos)
13     .where(eq(videos.lessonId, lesson.id))
14     .orderBy(videos.order)
15     .all();

```

4.5 Audit Trail Implementation

Track who changed what and when.

4.5.1 The Fields

Add these to every content table:

```
1 ALTER TABLE lessons ADD COLUMN created_at TEXT DEFAULT CURRENT_TIMESTAMP;
2 ALTER TABLE lessons ADD COLUMN updated_at TEXT;
3 ALTER TABLE lessons ADD COLUMN last_modified_by TEXT; -- "script" or user ID
4 ALTER TABLE lessons ADD COLUMN is_locked INTEGER DEFAULT 0;
```

4.5.2 Complete Table Definition

```
1 CREATE TABLE lessons (
2     -- Core fields
3     id TEXT PRIMARY KEY,
4     track_id TEXT,
5     series_id TEXT,
6     title TEXT NOT NULL,
7     slug TEXT UNIQUE NOT NULL,
8     description TEXT,
9     instructor TEXT,
10    pdf_url TEXT,
11    is_published INTEGER DEFAULT 0,
12    "order" INTEGER DEFAULT 0,
13
14    -- Audit fields
15    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
16    updated_at TEXT,
17    last_modified_by TEXT DEFAULT 'script',
18    is_locked INTEGER DEFAULT 0,
19
20    -- Foreign keys
21    FOREIGN KEY (track_id) REFERENCES tracks(id),
22    FOREIGN KEY (series_id) REFERENCES series(id)
23);
```

4.5.3 Using the Audit Trail

When updating via script:

```
1 db.update(lessons)
2     .set({
3         title: newTitle,
4         updated_at: new Date().toISOString(),
5         last_modified_by: 'script'
6     })
7     .where(eq(lessons.id, lessonId))
8     .run();
```

When updating via admin:

```

1 db.update(lessons)
2   .set({
3     title: newTitle,
4     updated_at: new Date().toISOString(),
5     last_modified_by: userId.toString() // Human user
6   })
7   .where(eq(lessons.id, lessonId))
8   .run();

```

Protecting human edits:

```

1 function shouldUpdate(existing) {
2   // Don't overwrite if locked
3   if (existing.isLocked) return false;
4
5   // Don't overwrite human edits (unless forced)
6   if (existing.lastModifiedBy !== 'script') {
7     return FORCE_MODE;
8   }
9
10  return true;
11 }

```

4.6 4.6 Indexing Strategies

Indexes speed up queries but slow down writes. Use wisely.

4.6.1 When to Index

Scenario	Index?
Column in WHERE clause	Yes
Column in JOIN	Yes
Column in ORDER BY	Maybe
Rarely queried column	No
Frequently updated column	Be careful

4.6.2 Our Indexes

```

1 -- users.db
2 CREATE INDEX idx_sessions_user ON sessions(user_id);
3 CREATE INDEX idx_progress_user ON lesson_progress(user_id);

```

```

4 CREATE INDEX idx_progress_lesson ON lesson_progress(lesson_id);
5
6 -- content.db
7 CREATE INDEX idx_lessons_track ON lessons(track_id);
8 CREATE INDEX idx_lessons_series ON lessons(series_id);
9 CREATE INDEX idx_lessons_slug ON lessons(slug);
10 CREATE INDEX idx_videos_lesson ON videos(lesson_id);
11 CREATE INDEX idx_series_track ON series(track_id);

```

4.6.3 Composite Indexes

For queries with multiple conditions:

```

1 -- For: WHERE user_id = ? AND lesson_id = ?
2 CREATE UNIQUE INDEX idx_progress_user_lesson
3     ON lesson_progress(user_id, lesson_id);

```

4.7 4.7 WAL Mode and Performance

WAL (Write-Ahead Logging) improves SQLite concurrency.

4.7.1 Enabling WAL Mode

```

1 const db = new Database('data/users.db');
2 db.exec('PRAGMA journal_mode = WAL');

```

4.7.2 What WAL Does

Without WAL: - Readers block writers - Writers block readers - Only one writer at a time

With WAL: - Readers never block writers - Writers never block readers - Multiple concurrent readers

4.7.3 Other Performance PRAGMAs

```

1 // Apply these at connection time
2 db.exec('PRAGMA journal_mode = WAL');           // Enable WAL
3 db.exec('PRAGMA busy_timeout = 5000');          // Wait 5s if busy
4 db.exec('PRAGMA synchronous = NORMAL');         // Balance safety/speed
5 db.exec('PRAGMA foreign_keys = ON');             // Enforce FK constraints
6 db.exec('PRAGMA cache_size = -64000');          // 64MB cache

```

4.7.4 When to Use WAL

Scenario	Mode
Read-heavy (content.db)	WAL
Write-heavy (users.db)	WAL
Single reader/writer	DELETE (default)
Network storage	DELETE (WAL needs local)

4.8 Summary

In this chapter, we designed our database architecture:

Aspect	Decision
Engine	SQLite (production-ready)
Strategy	Dual database (users + content)
IDs	TEXT for content, INTEGER for users
Hierarchy	Tracks → Series → Lessons → Videos
Auditing	created_at, updated_at, last_modified_by, is_locked
Performance	WAL mode, strategic indexes

Our schema is now ready. In the next chapter, we'll set up Drizzle ORM to interact with these databases in a type-safe way.

Next Chapter: [Chapter 5: Drizzle ORM Setup](#)

Chapter 5

Drizzle ORM Setup

5.1 5.1 Introduction to Drizzle

Drizzle is a modern TypeScript ORM that's lightweight, type-safe, and perfect for SQLite.

5.1.1 Why Drizzle?

Feature	Drizzle	Prisma	TypeORM
Bundle size	~50KB	~2MB	~500KB
TypeScript	Native	Generated	Decorators
SQLite support	Excellent	Good	Basic
Learning curve	Low	Medium	High
Raw SQL access	Easy	Harder	Medium

5.1.2 Core Philosophy

Drizzle follows “SQL-like” syntax:

```
1 // Drizzle - feels like SQL
2 db.select()
3   .from(users)
4   .where(eq(users.email, email))
5   .limit(1);
6
7 // Actual SQL it generates
8 // SELECT * FROM users WHERE email = ? LIMIT 1
```

5.1.3 Installation

```
1 bun add drizzle-orm
2 bun add -d drizzle-kit
```

5.2 5.2 Schema Definition

Let's define our schemas using Drizzle's declarative syntax.

5.2.1 users.db Schema

Create `src/lib/server/db/schema.ts`:

```
1 import { sqliteTable, text, integer } from 'drizzle-orm/sqlite-core';
2
3 // =====
4 // USERS DATABASE SCHEMA (users.db)
5 // =====
6
7 // Users table
8 export const users = sqliteTable('users', {
9     id: integer('id').primaryKey({ autoIncrement: true }),
10    email: text('email').notNull().unique(),
11    passwordHash: text('password_hash').notNull(),
12    name: text('name').notNull(),
13    role: text('role').default('user').notNull(), // user | editor | admin
14    avatar: text('avatar'),
15    isActive: integer('is_active', { mode: 'boolean' }).default(true).notNull(),
16    createdAt: text('created_at').default('CURRENT_TIMESTAMP').notNull(),
17    updatedAt: text('updated_at'),
18 });
19
20 // Sessions table
21 export const sessions = sqliteTable('sessions', {
22     id: text('id').primaryKey(), // UUID
23     userId: integer('user_id')
24         .notNull()
25         .references(() => users.id, { onDelete: 'cascade' }),
26     expiresAt: text('expires_at').notNull(),
27     createdAt: text('created_at').default('CURRENT_TIMESTAMP').notNull(),
28 });
29
30 // Lesson Progress table
31 export const lessonProgress = sqliteTable('lesson_progress', {
32     id: integer('id').primaryKey({ autoIncrement: true }),
33     userId: integer('user_id')
34         .notNull()
35         .references(() => users.id, { onDelete: 'cascade' }),
36     lessonId: text('lesson_id').notNull(),
37     watchedSeconds: integer('watched_seconds').default(0).notNull(),
38     videoCompleted: integer('video_completed', { mode: 'boolean' }).default(false).notNull(),
39     quizPassed: integer('quiz_passed', { mode: 'boolean' }).default(false).notNull(),
40     quizScore: integer('quiz_score').default(0),
```

```

41     completedAt: text('completed_at'),
42     updatedAt: text('updated_at').default('CURRENT_TIMESTAMP').notNull(),
43   });
44
45 // Type exports
46 export type User = typeof users.$inferSelect;
47 export type NewUser = typeof users.$inferInsert;
48 export type Session = typeof sessions.$inferSelect;
49 export type LessonProgress = typeof lessonProgress.$inferSelect;

```

5.2.2 content.db Schema

Create `src/lib/server/db/schema-content.ts`:

```

1 import { sqliteTable, text, integer } from 'drizzle-orm/sqlite-core';
2
3 // =====
4 // CONTENT DATABASE SCHEMA (content.db)
5 // =====
6
7 // Tracks (Learning Paths)
8 export const tracks = sqliteTable('tracks', {
9   id: text('id').primaryKey(),
10  title: text('title').notNull(),
11  description: text('description'),
12  icon: text('icon'),
13  order: integer('order').default(0).notNull(),
14 });
15
16 // Series (Course Collections)
17 export const series = sqliteTable('series', {
18   id: text('id').primaryKey(),
19   trackId: text('track_id').references(() => tracks.id),
20   title: text('title').notNull(),
21   description: text('description'),
22   instructor: text('instructor'),
23   thumbnailUrl: text('thumbnail_url'),
24   order: integer('order').default(0).notNull(),
25   // Audit fields
26   createdAt: text('created_at').default('CURRENT_TIMESTAMP'),
27   updatedAt: text('updated_at'),
28   lastModifiedBy: text('last_modified_by').default('script'),
29   isLocked: integer('is_locked', { mode: 'boolean' }).default(false),
30 });
31
32 // Lessons
33 export const lessons = sqliteTable('lessons', {
34   id: text('id').primaryKey(),
35   trackId: text('track_id').references(() => tracks.id),
36   seriesId: text('series_id').references(() => series.id),
37   title: text('title').notNull(),
38   slug: text('slug').notNull().unique(),
39   description: text('description'),
40   instructor: text('instructor'),
41   thumbnailUrl: text('thumbnail_url'),
42   pdfUrl: text('pdf_url'),
43   order: integer('order').default(0).notNull(),
44   isPublished: integer('is_published', { mode: 'boolean' }).default(false).notNull(),
45   // Audit fields
46   createdAt: text('created_at').default('CURRENT_TIMESTAMP'),

```

```

47     updatedAt: text('updated_at'),
48     lastModifiedBy: text('last_modified_by').default('script'),
49     isLocked: integer('is_locked', { mode: 'boolean' }).default(false),
50 });
51
52 // Videos
53 export const videos = sqliteTable('videos', {
54     id: text('id').primaryKey(),
55     lessonId: text('lesson_id')
56         .notNull()
57         .references(() => lessons.id, { onDelete: 'cascade' }),
58     title: text('title'),
59     youtubeId: text('youtube_id'),
60     downloadUrl: text('download_url'),
61     duration: integer('duration'),
62     summary: text('summary'),
63     rawTranscript: text('raw_transcript'),
64     formattedTranscript: text('formatted_transcript'),
65     quiz: text('quiz'), // JSON
66     order: integer('order').default(0).notNull(),
67     // Audit fields
68     createdAt: text('created_at').default('CURRENT_TIMESTAMP'),
69     updatedAt: text('updated_at'),
70     lastModifiedBy: text('last_modified_by').default('script'),
71     isLocked: integer('is_locked', { mode: 'boolean' }).default(false),
72 });
73
74 // Type exports
75 export type Track = typeof tracks.$inferSelect;
76 export type Series = typeof series.$inferSelect;
77 export type Lesson = typeof lessons.$inferSelect;
78 export type Video = typeof videos.$inferSelect;

```

5.3 5.3 Type-Safe Queries

Drizzle provides complete type safety for all queries.

5.3.1 Basic Select

```

1 import { db } from '$lib/server/db';
2 import { users } from '$lib/server/db/schema';
3 import { eq } from 'drizzle-orm';
4
5 // Get user by email
6 const user = db
7     .select()
8     .from(users)
9     .where(eq(users.email, 'test@example.com'))
10    .get();
11
12 // TypeScript knows: user is User | undefined

```

5.3.2 Select Specific Columns

```
1 // Only get what you need
2 const userNames = db
3     .select({
4         id: users.id,
5         name: users.name,
6     })
7     .from(users)
8     .all();
9
10 // Type: { id: number; name: string }[]
```

5.3.3 Insert

```
1 // Insert returns the inserted row
2 const newUser = db
3     .insert(users)
4     .values({
5         email: 'new@example.com',
6         passwordHash: hashedPassword,
7         name: 'New User',
8     })
9     .returning()
10    .get();
11
12 // newUser is User
```

5.3.4 Update

```
1 // Update with conditions
2 db.update(users)
3     .set({
4         name: 'Updated Name',
5         updatedAt: new Date().toISOString(),
6     })
7     .where(eq(users.id, userId))
8     .run();
```

5.3.5 Delete

```
1 // Delete with conditions
2 db.delete(sessions)
3     .where(eq(sessions.id, sessionId))
4     .run();
```

5.4 5.4 Migrations Strategy

For our project, we use a simpler approach than traditional migrations.

5.4.1 Why Not Traditional Migrations?

Traditional migration systems (like Prisma Migrate) are great for teams and complex deployments. But for our case:

- Single developer or small team
- SQLite (easy to backup/restore)
- Content database that can be regenerated

5.4.2 Our Approach: Setup Scripts

Create tables script (`scripts/setup-db.ts`):

```
1 import { Database } from 'bun:sqlite';
2 import { resolve } from 'path';
3
4 const dataDir = resolve(import.meta.dir, '../data');
5
6 console.log(` Creating users.db...`);
7
8 const db = new Database(resolve(dataDir, 'users.db'), { create: true });
9
10 db.exec('PRAGMA journal_mode = WAL');
11 db.exec('PRAGMA foreign_keys = ON');
12
13 db.exec(`
14     CREATE TABLE IF NOT EXISTS users (
15         id INTEGER PRIMARY KEY AUTOINCREMENT,
16         email TEXT NOT NULL UNIQUE,
17         password_hash TEXT NOT NULL,
18         name TEXT NOT NULL,
19         role TEXT DEFAULT 'user' NOT NULL,
20         avatar TEXT,
21         is_active INTEGER DEFAULT 1 NOT NULL,
22         created_at TEXT DEFAULT CURRENT_TIMESTAMP NOT NULL,
23         updated_at TEXT
24     );
25
26     CREATE TABLE IF NOT EXISTS sessions (
27         id TEXT PRIMARY KEY,
28         user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
29         expires_at TEXT NOT NULL,
30         created_at TEXT DEFAULT CURRENT_TIMESTAMP NOT NULL
31     );
32
33     CREATE TABLE IF NOT EXISTS lesson_progress (
34         id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```

35     user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
36     lesson_id TEXT NOT NULL,
37     watched_seconds INTEGER DEFAULT 0 NOT NULL,
38     video_completed INTEGER DEFAULT 0 NOT NULL,
39     quiz_passed INTEGER DEFAULT 0 NOT NULL,
40     quiz_score INTEGER DEFAULT 0,
41     completed_at TEXT,
42     updated_at TEXT DEFAULT CURRENT_TIMESTAMP NOT NULL,
43     UNIQUE(user_id, lesson_id)
44 );
45
46 CREATE INDEX IF NOT EXISTS idx_sessions_user ON sessions(user_id);
47 CREATE INDEX IF NOT EXISTS idx_progress_user ON lesson_progress(user_id);
48 `);
49
50 console.log(' Done!');
51 db.close();

```

Run with:

```
1 bun run scripts/setup-db.ts
```

5.4.3 Schema Changes

When you need to change the schema:

1. **Development:** Delete the .db file and re-run setup
 2. **Production with data:** Write ALTER TABLE script
 3. **Complex changes:** Backup, migrate, restore
-

5.5 Connecting Multiple Databases

Here's how we connect both databases.

5.5.1 Connection Module

Create `src/lib/server/db/index.ts`:

```

1 import { drizzle } from 'drizzle-orm/bun-sqlite';
2 import { Database } from 'bun:sqlite';
3 import { resolve } from 'path';
4
5 // Import schemas
6 import * as userSchema from './schema';

```

```

7 import * as contentSchema from './schema-content';
8
9 // Resolve paths
10 const dataDir = resolve(process.cwd(), 'data');
11
12 // =====
13 // USER DATABASE (Read-Write)
14 // =====
15
16 const usersSqlite = new Database(resolve(dataDir, 'users.db'));
17 usersSqlite.exec('PRAGMA journal_mode = WAL');
18 usersSqlite.exec('PRAGMA busy_timeout = 5000');
19 usersSqlite.exec('PRAGMA synchronous = NORMAL');
20 usersSqlite.exec('PRAGMA foreign_keys = ON');
21
22 export const db = drizzle(usersSqlite, { schema: userSchema });
23
24 // =====
25 // CONTENT DATABASE (Read-Mostly)
26 // =====
27
28 const contentSqlite = new Database(resolve(dataDir, 'content.db'));
29 contentSqlite.exec('PRAGMA foreign_keys = ON');
30
31 export const contentDatabase = drizzle(contentSqlite, { schema: contentSchema });
32
33 // =====
34 // RE-EXPORTS
35 // =====
36
37 // Re-export schemas for convenience
38 export * from './schema';
39 export * from './schema-content';

```

5.5.2 Usage in SvelteKit

```

1 // In +page.server.ts
2 import { db, contentDatabase, users, lessons } from '$lib/server/db';
3
4 export async function load() {
5     // Query user database
6     const allUsers = db.select().from(users).all();
7
8     // Query content database
9     const allLessons = contentDatabase.select().from(lessons).all();
10
11     return { users: allUsers, lessons: allLessons };
12 }

```

5.6 5.6 Query Optimization

Write efficient queries from the start.

5.6.1 Use Specific Selects

```

1 // ✗ Bad: Fetches all columns
2 const lessons = contentDatabase.select().from(lessons).all();
3
4 // ✅ Good: Only fetches needed columns
5 const lessons = contentDatabase
6   .select({
7     id: lessons.id,
8     title: lessons.title,
9     slug: lessons.slug,
10    })
11   .from(lessons)
12   .all();
```

5.6.2 Use Limits

```

1 // Always limit when you don't need all results
2 const recent = contentDatabase
3   .select()
4   .from(lessons)
5   .orderBy(desc(lessons.createdAt))
6   .limit(10)
7   .all();
```

5.6.3 Use Joins Instead of N+1

```

1 // ✗ Bad: N+1 queries
2 const lessons = contentDatabase.select().from(lessons).all();
3 for (const lesson of lessons) {
4   const track = contentDatabase
5     .select()
6     .from(tracks)
7     .where(eq(tracks.id, lesson.trackId))
8     .get();
9 }
10
11 // ✅ Good: Single join query
12 const lessonsWithTrack = contentDatabase
13   .select()
14   .from(lessons)
15   .leftJoin(tracks, eq(lessons.trackId, tracks.id))
16   .all();
```

5.6.4 Count Queries

```

1 import { count } from 'drizzle-orm';
2
3 // Get count efficiently
4 const result = contentDatabase
5   .select({ count: count() })
6   .from(lessons)
```

```
7     .where(eq(lessons.isPublished, true))
8     .get();
9
10 const total = result?.count ?? 0;
```

5.6.5 Aggregations

```
1 import { count, sql } from 'drizzle-orm';
2
3 // Count lessons per track
4 const stats = contentDatabase
5     .select({
6         trackId: lessons.trackId,
7         lessonCount: count(),
8     })
9     .from(lessons)
10    .groupBy(lessons.trackId)
11    .all();
```

5.7 Summary

In this chapter, we set up Drizzle ORM:

Topic	Key Points
Schemas	Declarative, type-safe definitions
Queries	SQL-like syntax with full TypeScript support
Connections	Dual database setup with proper pragmas
Optimization	Specific selects, limits, joins

Our data layer is now complete. Next, we'll build the authentication system on top of it.

Next Chapter: [Chapter 6: Authentication System](#)

Chapter 6

Authentication System

6.1 6.1 Session-Based vs JWT

Two main approaches exist for web authentication. Let's compare.

6.1.1 JWT (JSON Web Tokens)

Pros: - Stateless (no database lookup) - Works across domains - Good for APIs

Cons: - Can't revoke easily - Token size adds overhead - Security complexity

6.1.2 Session-Based

Pros: - Easy session revocation - Server controls everything - Simple implementation - Better security by default

Cons: - Requires database lookup - Doesn't work across domains easily

6.1.3 Our Choice: Sessions

For our educational platform: - **Same-origin requests** (no API for other apps) - **Easy logout** (just delete session) - **Session data** (store progress, preferences) - **Simplicity** (less code, fewer bugs)

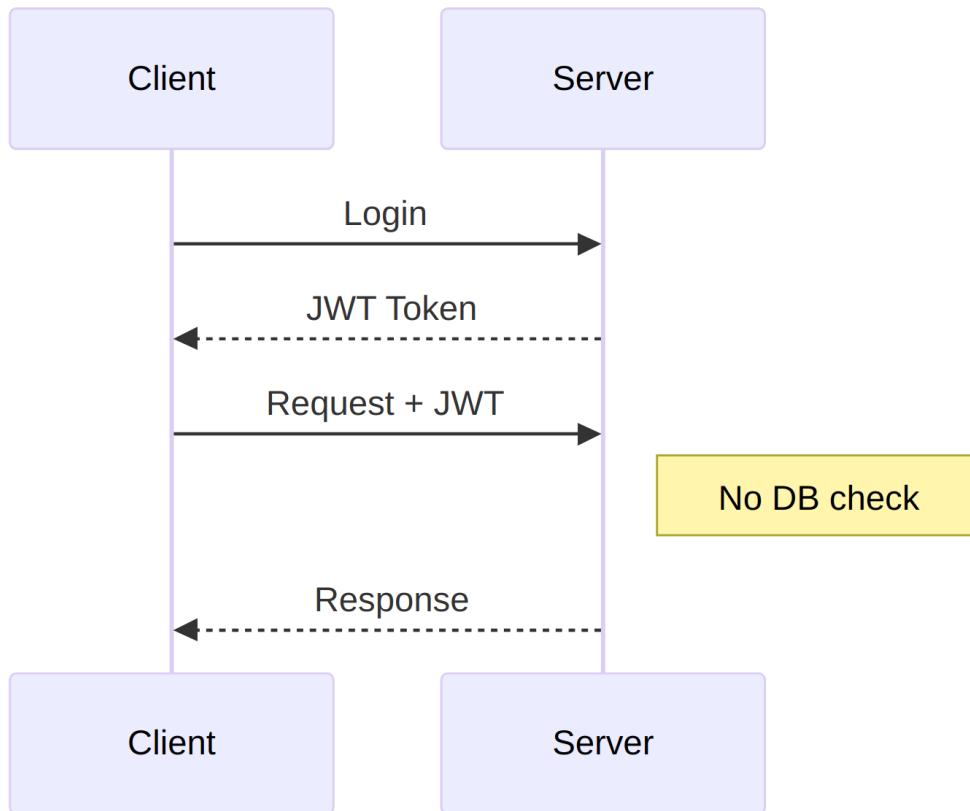


Figure 6.1: JWT Authentication Flow

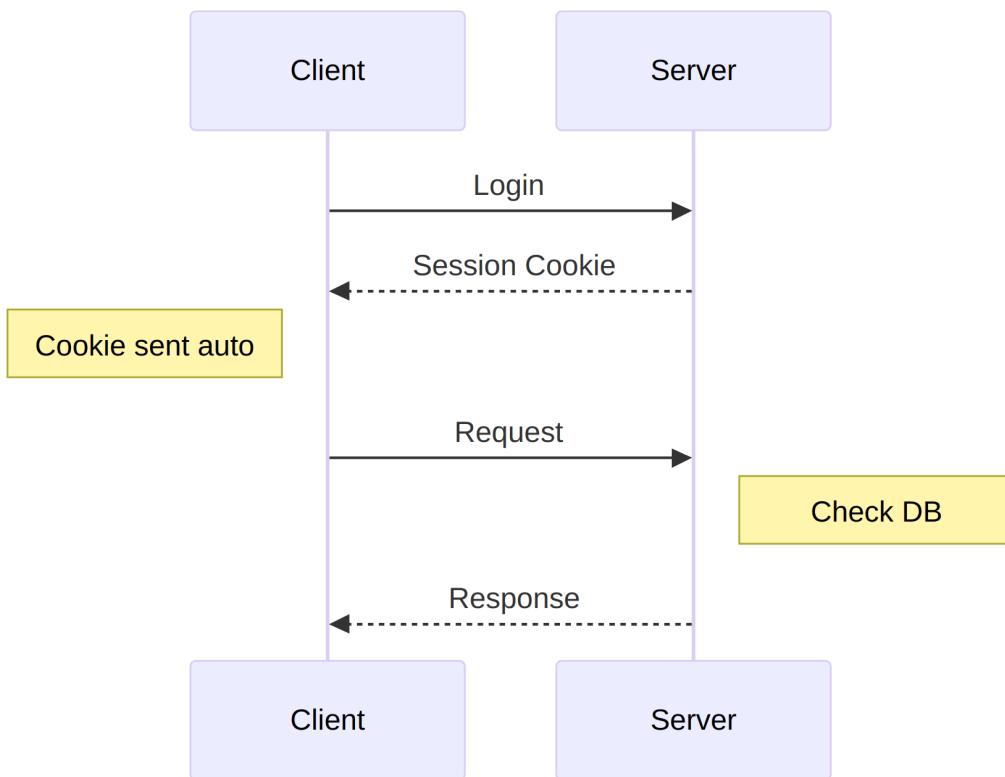


Figure 6.2: Session-Based Authentication Flow

6.2 Password Hashing with Bun.password

Never store plain passwords. Always hash.

6.2.1 Why Argon2id?

Algorithm	Security
MD5	✗ Broken
SHA1	✗ Broken
SHA256	⚠ Too fast
bcrypt	✓ Good but slow
Argon2id	✓ Modern, built into Bun!

Bun's native `Bun.password`: - **25x faster** than bcrypt npm package - **Argon2id** (memory-hard, GPU-resistant) - **Zero dependencies** (native to Bun)

6.2.2 Implementation

```

1 // src/lib/server/auth.ts
2
3 export async function hashPassword(password: string): Promise<string> {
4     return await Bun.password.hash(password, {
5         algorithm: 'argon2id',
6         memoryCost: 4,
7         timeCost: 3,
8     });
9 }
10
11 export async function verifyPassword(
12     password: string,
13     hash: string
14 ): Promise<boolean> {
15     return await Bun.password.verify(password, hash);
16 }
```

6.2.3 Usage

```

1 // Registration
2 const passwordHash = await hashPassword(userPassword);
3 db.insert(users).values({
4     email,
5     passwordHash,
6     name,
7 }).run();
8
```

```
9 // Login
10 const user = db.select().from(users)
11     .where(eq(users.email, email))
12     .get();
13
14 if (user && await verifyPassword(password, user.passwordHash)) {
15     // Login successful
16 } else {
17     // Invalid credentials
18 }
```

6.3 6.3 Session Management

Sessions link browser requests to user accounts.

6.3.1 Session Flow

```
1 1. User logs in with email/password
2 2. Server:
3     - Validates credentials
4     - Creates session in database
5     - Sets cookie with session ID
6
7 3. User makes request
8 4. Server:
9     - Reads session ID from cookie
10    - Looks up session in database
11    - Attaches user to request
12
13 5. User logs out
14 6. Server:
15     - Deletes session from database
16     - Clears cookie
```

6.3.2 Session Schema

```
1 // In schema.ts
2 export const sessions = sqliteTable('sessions', {
3     id: text('id').primaryKey(),           // UUID
4     userId: integer('user_id').notNull(),
5     expiresAt: text('expires_at').notNull(),
6     createdAt: text('created_at').default('CURRENT_TIMESTAMP'),
7});
```

6.3.3 Session Functions

```
1 // src/lib/server/auth.ts
2 import { db, sessions, users } from '$lib/server/db';
3 import { eq, and, gt } from 'drizzle-orm';
4 import { randomUUID } from 'crypto';
5
6 const SESSION_DURATION_DAYS = 7;
7 const REMEMBER_ME_DAYS = 30;
8
9 export function createSession(
10     userId: number,
11     rememberMe: boolean = false
12 ): { sessionId: string; expiresAt: Date } {
13     const sessionId = randomUUID();
14     const days = rememberMe ? REMEMBER_ME_DAYS : SESSION_DURATION_DAYS;
15     const expiresAt = new Date();
16     expiresAt.setDate(expiresAt.getDate() + days);
17
18     db.insert(sessions).values({
19         id: sessionId,
20         userId,
21         expiresAt: expiresAt.toISOString(),
22     }).run();
23
24     return { sessionId, expiresAt };
25 }
26
27 export function validateSession(sessionId: string) {
28     const now = new Date().toISOString();
29
30     const result = db
31         .select({
32             session: sessions,
33             user: users,
34         })
35         .from(sessions)
36         .innerJoin(users, eq(sessions.userId, users.id))
37         .where(
38             and(
39                 eq(sessions.id, sessionId),
40                 gt(sessions.expiresAt, now)
41             )
42         )
43         .get();
44
45     if (!result) return null;
46
47     return {
48         session: result.session,
49         user: result.user,
50     };
51 }
52
53 export function deleteSession(sessionId: string) {
54     db.delete(sessions).where(eq(sessions.id, sessionId)).run();
55 }
56
57 export function deleteUserSessions(userId: number) {
58     db.delete(sessions).where(eq(sessions.userId, userId)).run();
59 }
```

6.4 6.4 Cookie Security (HttpOnly, SameSite)

Cookies must be configured securely.

6.4.1 Cookie Options

```

1 import type { Cookies } from '@sveltejs/kit';
2
3 const COOKIE_NAME = 'session';
4
5 export function setSessionCookie(
6     cookies: Cookies,
7     sessionId: string,
8     expiresAt: Date
9 ) {
10     cookies.set(COOKIE_NAME, sessionId, {
11         path: '/',
12         httpOnly: true,           // JavaScript can't read it
13         sameSite: 'lax',          // Prevents CSRF
14         secure: process.env.NODE_ENV === 'production',
15         expires: expiresAt,
16     });
17 }
18
19 export function getSessionCookie(cookies: Cookies): string | undefined {
20     return cookies.get(COOKIE_NAME);
21 }
22
23 export function clearSessionCookie(cookies: Cookies) {
24     cookies.delete(COOKIE_NAME, { path: '/' });
25 }
```

6.4.2 Option Explanations

Option	Value	Purpose
httpOnly: true	Browser only	Prevents XSS stealing session
sameSite: 'lax'	Same origin + links	Prevents CSRF attacks
secure: true	HTTPS only	Prevents interception
path: '/'	All routes	Session works everywhere

6.5 6.5 Remember Me Functionality

Users can choose to stay logged in longer.

6.5.1 Implementation

```
1 // Login action
2 export const actions: Actions = {
3     default: async ({ request, cookies }) => {
4         const form = await request.formData();
5         const email = form.get('email') as string;
6         const password = form.get('password') as string;
7         const rememberMe = form.get('rememberMe') === 'true';
8
9         // Validate user...
10
11         // Create session with extended expiry if rememberMe
12         const { sessionId, expiresAt } = createSession(
13             user.id,
14             rememberMe // 30 days vs 7 days
15         );
16
17         setSessionCookie(cookies, sessionId, expiresAt);
18
19         return redirect(302, '/');
20     }
21 };
```

6.5.2 Login Form

```
1 <form method="POST">
2     <input type="email" name="email" required />
3     <input type="password" name="password" required />
4
5     <label>
6         <input type="checkbox" name="rememberMe" value="true" />
7         Remember me for 30 days
8     </label>
9
10    <button type="submit">Login</button>
11 </form>
```

6.6 6.6 Role-Based Access Control (RBAC)

Different users have different permissions.

6.6.1 Roles

Role	Permissions
user	View content, track progress
editor	Edit content (unlocked)
admin	Full access, lock/unlock content

6.6.2 Schema

```
1 // In users table
2 role: text('role').default('user').notNull(),
```

6.6.3 Authorization Helpers

```
1 // src/lib/server/auth.ts
2
3 export function isAdmin(user: User | null): boolean {
4     return user?.role === 'admin';
5 }
6
7 export function isEditor(user: User | null): boolean {
8     return user?.role === 'editor' || user?.role === 'admin';
9 }
10
11 export function canEditContent(user: User | null): boolean {
12     return isEditor(user);
13 }
14
15 export function canLockContent(user: User | null): boolean {
16     return isAdmin(user);
17 }
```

6.6.4 Usage in Load Functions

```
1 // src/routes/admin/+page.server.ts
2 import { redirect } from '@sveltejs/kit';
3 import { isEditor } from '$lib/server/auth';
4
5 export async function load({ locals }) {
6     if (!locals.user) {
7         throw redirect(302, '/auth/login?redirect=/admin');
8     }
9
10    if (!isEditor(locals.user)) {
11        throw redirect(302, '/');
12    }
13
14    // Load admin data...
15}
```

6.7 Protected Routes with Hooks

SvelteKit hooks provide middleware-like functionality.

6.7.1 The Hook

Create `src/hooks.server.ts`:

```

1 import type { Handle } from '@sveltejs/kit';
2 import { validateSession, getSessionCookie } from '$lib/server/auth';
3
4 export const handle: Handle = async ({ event, resolve }) => {
5     // Get session from cookie
6     const sessionId = getSessionCookie(event.cookies);
7
8     if (sessionId) {
9         const result = validateSession(sessionId);
10
11     if (result) {
12         // Attach user to request
13         event.locals.user = result.user;
14     } else {
15         // Invalid/expired session - clear cookie
16         event.cookies.delete('session', { path: '/' });
17     }
18 }
19
20 // Rate limiting placeholder for auth routes
21 if (event.url.pathname.startsWith('/api/auth')) {
22     // TODO: Implement rate limiting
23 }
24
25 return resolve(event);
26 };

```

6.7.2 Type Definitions

Update `src/app.d.ts`:

```

1 import type { User } from '$lib/server/db/schema';
2
3 declare global {
4     namespace App {
5         interface Locals {
6             user: User | null;
7         }
8     }
9 }
10
11 export {};

```

6.7.3 Accessing User in Routes

```

1 // +page.server.ts
2 export async function load({ locals }) {
3     // locals.user is available everywhere!
4     const isLoggedIn = !!locals.user;
5
6     return {
7         user: locals.user,
8         isLoggedIn,
9     };
10 }

```

6.7.4 Layout Data

Share user across all pages:

```

1 // src/routes/+layout.server.ts
2 export async function load({ locals }) {
3     return {
4         user: locals.user,
5     };
6 }

```

```

1 <!-- src/routes/+layout.svelte -->
2 <script>
3     let { data, children } = $props();
4 </script>
5
6 {#if data.user}
7     <p>Logged in as {data.user.name}</p>
8 {/if}
9
10 {@render children()}

```

6.8 Complete Auth Module

Here's the full `src/lib/server/auth.ts`:

```

1 import { randomUUID } from 'crypto';
2 import { db, users, sessions, type User } from '$lib/server/db';
3 import { eq, and, gt } from 'drizzle-orm';
4 import type { Cookies } from '@sveltejs/kit';
5
6 // Configuration
7 const SESSION_DAYS = 7;
8 const REMEMBER_ME_DAYS = 30;
9 const COOKIE_NAME = 'session';
10
11 // Password hashing with Bun.password (Argon2id)

```

```
12 export async function hashPassword(password: string): Promise<string> {
13     return await Bun.password.hash(password, {
14         algorithm: 'argon2id',
15         memoryCost: 4,
16         timeCost: 3,
17     });
18 }
19
20 export async function verifyPassword(password: string, hash: string): Promise<boolean> {
21     return await Bun.password.verify(password, hash);
22 }
23
24 // Session management
25 export function createSession(userId: number, rememberMe = false) {
26     const sessionId = randomUUID();
27     const days = rememberMe ? REMEMBER_ME_DAYS : SESSION_DAYS;
28     const expiresAt = new Date();
29     expiresAt.setDate(expiresAt.getDate() + days);
30
31     db.insert(sessions).values({
32         id: sessionId,
33         userId,
34         expiresAt: expiresAt.toISOString(),
35     }).run();
36
37     return { sessionId, expiresAt };
38 }
39
40 export function validateSession(sessionId: string) {
41     const now = new Date().toISOString();
42
43     return db
44         .select({ session: sessions, user: users })
45         .from(sessions)
46         .innerJoin(users, eq(sessions.userId, users.id))
47         .where(and(eq(sessions.id, sessionId), gt(sessions.expiresAt, now)))
48         .get();
49 }
50
51 export function deleteSession(sessionId: string) {
52     db.delete(sessions).where(eq(sessions.id, sessionId)).run();
53 }
54
55 // Cookie helpers
56 export function setSessionCookie(cookies: Cookies, sessionId: string, expiresAt: Date) {
57     cookies.set(COOKIE_NAME, sessionId, {
58         path: '/',
59         httpOnly: true,
60         sameSite: 'lax',
61         secure: process.env.NODE_ENV === 'production',
62         expires: expiresAt,
63     });
64 }
65
66 export function getSessionCookie(cookies: Cookies) {
67     return cookies.get(COOKIE_NAME);
68 }
69
70 export function clearSessionCookie(cookies: Cookies) {
71     cookies.delete(COOKIE_NAME, { path: '/' });
72 }
73
74 // Role helpers
75 export const isAdmin = (user: User | null) => user?.role === 'admin';
```

```
76 export const isEditor = (user: User | null) =>
77   user?.role === 'editor' || user?.role === 'admin';
```

6.9 Summary

Component	Implementation
Strategy	Session-based (simple, secure)
Passwords	Bun.password with Argon2id
Sessions	UUID in SQLite + HttpOnly cookie
Remember Me	Extended cookie expiry
Roles	user / editor / admin
Protection	SvelteKit hooks middleware

Authentication is now complete. Next chapter: API design patterns.

Next Chapter: [Chapter 7: API Design](#)

Chapter 7

API Design

7.1 7.1 SvelteKit Server Routes

SvelteKit makes API creation elegant.

7.1.1 Route Types

File	Purpose
+page.svelte	UI component
+page.server.ts	Page data loading & form actions
+server.ts	Pure API endpoint

7.1.2 API Routes Location

```
1 src/routes/└──
2   api/
3     ├── progress/
4     |   └── +server.ts    # /api/progress
5     ├── auth/
6     |   ├── login/
7     |   |   └── +server.ts    # /api/auth/login
8     |   └── register/
9     |       └── +server.ts    # /api/auth/register
10    └── health/
11      └── +server.ts    # /api/health
```

7.1.3 Basic Endpoint Structure

```
1 // src/routes/api/health/+server.ts
2 import { json } from '@sveltejs/kit';
3 import type { RequestHandler } from './$types';
```

```

4
5 export const GET: RequestHandler = async () => {
6     return json({
7         status: 'ok',
8         timestamp: new Date().toISOString(),
9     });
10 };

```

7.2 7.2 RESTful Endpoints

Follow REST conventions for predictable APIs.

7.2.1 HTTP Methods

Method	Purpose	Example
GET	Read data	Get progress
POST	Create/Action	Update progress
PUT	Replace	Replace resource
PATCH	Partial update	Update field
DELETE	Remove	Delete session

7.2.2 Progress API Example

```

1 // src/routes/api/progress/+server.ts
2 import { json } from '@sveltejs/kit';
3 import type { RequestHandler } from './$types';
4 import { db, lessonProgress } from '$lib/server/db';
5 import { eq, and } from 'drizzle-orm';
6
7 // GET: Fetch user's progress
8 export const GET: RequestHandler = async ({ url, locals }) => {
9     if (!locals.user) {
10         return json({ error: 'Unauthorized' }, { status: 401 });
11     }
12
13     const lessonId = url.searchParams.get('lessonId');
14
15     if (lessonId) {
16         // Get progress for specific lesson
17         const progress = db
18             .select()
19             .from(lessonProgress)
20             .where(
21                 and(

```

```
22         eq(lessonProgress.userId, locals.user.id),
23         eq(lessonProgress.lessonId, lessonId)
24     )
25   )
26 .get();
27
28   return json({ progress });
29 }
30
31 // Get all progress for user
32 const allProgress = db
33   .select()
34   .from(lessonProgress)
35   .where(eq(lessonProgress.userId, locals.user.id))
36   .all();
37
38   return json({ progress: allProgress });
39 };
40
41 // POST: Update progress
42 export const POST: RequestHandler = async ({ request, locals }) => {
43   if (!locals.user) {
44     return json({ error: 'Unauthorized' }, { status: 401 });
45   }
46
47   const body = await request.json();
48   const { lessonId, watchedSeconds, videoCompleted, quizPassed } = body;
49
50   if (!lessonId) {
51     return json({ error: 'lessonId required' }, { status: 400 });
52   }
53
54   const now = new Date().toISOString();
55
56   // Check if progress exists
57   const existing = db
58     .select()
59     .from(lessonProgress)
60     .where(
61       and(
62         eq(lessonProgress.userId, locals.user.id),
63         eq(lessonProgress.lessonId, lessonId)
64       )
65     )
66   .get();
67
68   if (existing) {
69     // Update existing
70     const updateData: Record<string, unknown> = { updatedAt: now };
71
72     if (watchedSeconds !== undefined) {
73       updateData.watchedSeconds = Math.max(existing.watchedSeconds, watchedSeconds);
74     }
75     if (videoCompleted !== undefined) {
76       updateData.videoCompleted = videoCompleted;
77     }
78     if (quizPassed !== undefined) {
79       updateData.quizPassed = quizPassed;
80     }
81     if (quizScore !== undefined) {
82       updateData.quizScore = quizScore;
83     }
84
85     db.update(lessonProgress)
```

```
86     .set(updateData)
87     .where(eq(lessonProgress.id, existing.id))
88     .run();
89 } else {
90     // Create new
91     db.insert(lessonProgress).values({
92         userId: locals.user.id,
93         lessonId,
94         watchedSeconds: watchedSeconds ?? 0,
95         videoCompleted: videoCompleted ?? false,
96         quizPassed: quizPassed ?? false,
97         quizScore: quizScore ?? 0,
98         updatedAt: now,
99     }).run();
100 }
101
102 return json({ success: true });
103 };
```

7.3 7.3 Request Validation

Never trust client input.

7.3.1 Manual Validation

```
1 export const POST: RequestHandler = async ({ request }) => {
2     const body = await request.json();
3
4     // Validate required fields
5     const { email, password, name } = body;
6
7     const errors: string[] = [];
8
9     if (!email || typeof email !== 'string') {
10         errors.push('Email is required');
11     } else if (!email.includes('@')) {
12         errors.push('Invalid email format');
13     }
14
15     if (!password || typeof password !== 'string') {
16         errors.push('Password is required');
17     } else if (password.length < 8) {
18         errors.push('Password must be at least 8 characters');
19     }
20
21     if (!name || typeof name !== 'string') {
22         errors.push('Name is required');
23     }
24
25     if (errors.length > 0) {
26         return json({ errors }, { status: 400 });
27     }
```

```

28     // Safe to proceed...
29 }
30 
```

7.3.2 Validation Helper

```

1 // src/lib/server/validation.ts
2
3 interface ValidationResult {
4     valid: boolean;
5     errors: string[];
6 }
7
8 export function validateEmail(email: unknown): ValidationResult {
9     if (!email || typeof email !== 'string') {
10         return { valid: false, errors: ['Email is required'] };
11     }
12     if (!email.includes('@') || email.length < 5) {
13         return { valid: false, errors: ['Invalid email format'] };
14     }
15     return { valid: true, errors: [] };
16 }
17
18 export function validatePassword(password: unknown): ValidationResult {
19     if (!password || typeof password !== 'string') {
20         return { valid: false, errors: ['Password is required'] };
21     }
22     if (password.length < 8) {
23         return { valid: false, errors: ['Password must be at least 8 characters'] };
24     }
25     return { valid: true, errors: [] };
26 }
27
28 export function validateRequired(value: unknown, name: string): ValidationResult {
29     if (!value || (typeof value === 'string' && !value.trim())) {
30         return { valid: false, errors: [`#${name} is required`] };
31     }
32     return { valid: true, errors: [] };
33 } 
```

7.4 7.4 Error Handling

Consistent error responses make debugging easier.

7.4.1 Error Response Format

```

1 // Standard error response
2 {
3     "error": "Brief error message", 
```

```

4     "code": "ERROR_CODE",           // Optional: machine-readable
5     "details": {}                 // Optional: additional info
6 }
```

7.4.2 Error Responses

```

1 // 400 Bad Request - Client error
2 return json(
3     { error: 'Invalid input', code: 'VALIDATION_ERROR' },
4     { status: 400 }
5 );
6
7 // 401 Unauthorized - Not logged in
8 return json(
9     { error: 'Authentication required' },
10    { status: 401 }
11 );
12
13 // 403 Forbidden - Logged in but not allowed
14 return json(
15     { error: 'Permission denied' },
16     { status: 403 }
17 );
18
19 // 404 Not Found - Resource doesn't exist
20 return json(
21     { error: 'Lesson not found' },
22     { status: 404 }
23 );
24
25 // 500 Internal Server Error - Our fault
26 return json(
27     { error: 'Something went wrong' },
28     { status: 500 }
29 );
```

7.4.3 Try-Catch Wrapper

```

1 export const POST: RequestHandler = async ({ request, locals }) => {
2     try {
3         // ... handler logic
4         return json({ success: true });
5     } catch (error) {
6         console.error('Progress update error:', error);
7         return json(
8             { error: 'Failed to update progress' },
9             { status: 500 }
10        );
11    }
12};
```

7.5 7.5 Rate Limiting

Prevent abuse with request limits.

7.5.1 Simple In-Memory Rate Limiter

```
1 // src/lib/server/ratelimit.ts
2
3 interface RateLimitEntry {
4     count: number;
5     resetAt: number;
6 }
7
8 const limits = new Map<string, RateLimitEntry>();
9
10 // Clean up old entries periodically
11 setInterval(() => {
12     const now = Date.now();
13     for (const [key, entry] of limits) {
14         if (entry.resetAt < now) {
15             limits.delete(key);
16         }
17     }
18 }, 60000); // Every minute
19
20 export function checkRateLimit(
21     key: string,
22     maxRequests: number = 10,
23     windowMs: number = 60000
24 ): { allowed: boolean; remaining: number; resetIn: number } {
25     const now = Date.now();
26     const entry = limits.get(key);
27
28     if (!entry || entry.resetAt < now) {
29         // New window
30         limits.set(key, {
31             count: 1,
32             resetAt: now + windowMs,
33         });
34         return {
35             allowed: true,
36             remaining: maxRequests - 1,
37             resetIn: windowMs,
38         };
39     }
40
41     if (entry.count >= maxRequests) {
42         return {
43             allowed: false,
44             remaining: 0,
45             resetIn: entry.resetAt - now,
46         };
47     }
48
49     entry.count++;
50     return {
51         allowed: true,
52         remaining: maxRequests - entry.count,
53         resetIn: entry.resetAt - now,
```

```
54     });
55 }
```

7.5.2 Usage in Hooks

```
1 // src/hooks.server.ts
2 import { checkRateLimit } from '$lib/server/ratelimit';
3 import { json } from '@sveltejs/kit';
4
5 export const handle: Handle = async ({ event, resolve }) => {
6     // Rate limit auth endpoints
7     if (event.url.pathname.startsWith('/api/auth')) {
8         const ip = event.getClientAddress();
9         const { allowed, resetIn } = checkRateLimit(
10             `auth:${ip}`,
11             5,           // 5 requests
12             60000        // per minute
13         );
14
15         if (!allowed) {
16             return json(
17                 { error: 'Too many requests. Try again later.' },
18                 {
19                     status: 429,
20                     headers: {
21                         'Retry-After': Math.ceil(resetIn / 1000).toString(),
22                     },
23                 }
24             );
25         }
26     }
27
28     return resolve(event);
29};
```

7.6 7.6 CORS Configuration

Usually not needed for same-origin, but good to know.

7.6.1 When You Need CORS

- API consumed by different domain
- Mobile app accessing your API
- Third-party integrations

7.6.2 SvelteKit CORS

```

1 // src/hooks.server.ts
2
3 const ALLOWED_ORIGINS = [
4     'https://myapp.com',
5     'https://mobile.myapp.com',
6 ];
7
8 export const handle: Handle = async ({ event, resolve }) => {
9     const origin = event.request.headers.get('origin');
10
11     // Handle preflight
12     if (event.request.method === 'OPTIONS') {
13         if (origin && ALLOWED_ORIGINS.includes(origin)) {
14             return new Response(null, {
15                 headers: {
16                     'Access-Control-Allow-Origin': origin,
17                     'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE',
18                     'Access-Control-Allow-Headers': 'Content-Type',
19                     'Access-Control-Max-Age': '86400',
20                 },
21             });
22         }
23     }
24
25     const response = await resolve(event);
26
27     // Add CORS headers to response
28     if (origin && ALLOWED_ORIGINS.includes(origin)) {
29         response.headers.set('Access-Control-Allow-Origin', origin);
30     }
31
32     return response;
33 };

```

7.6.3 For Our Platform

We don't need CORS because:

- Same-origin requests only
- No external API consumers
- SvelteKit handles everything

7.7 API Summary

Our complete API structure:

```

1 /api|-
2   /progress           # Lesson progress|
3     |--- GET ?lessonId=X    # Get progress|
4     |--- POST           # Update progress| |
5
6   /auth|-

```

```
7   └── /login|  
8     └── POST          # Login|  
9   └── /register|  
10    └── POST          # Register|  
11   └── /logout|  
12     └── POST          # Logout |└—  
13  
14 /health  
15     └── GET           # Health check
```

7.8 Summary

Topic	Key Points
Routes	File-based, +server.ts for APIs
Methods	GET/POST/PUT/DELETE conventions
Validation	Always validate, never trust input
Errors	Consistent format, proper status codes
Rate Limiting	Protect auth endpoints
CORS	Not needed for same-origin

Part II is complete! We've built our entire backend foundation. Next, we move to the frontend.

Next Chapter: [Part III → Chapter 8: SvelteKit Fundamentals](#)

Chapter 8

SvelteKit Fundamentals

8.1 8.1 File-Based Routing

SvelteKit uses the file system to define routes.

8.1.1 Basic Routing

```
1 src/routes/|—  
2   +page.svelte      → /|—  
3   about/|  
4     └+page.svelte    → /about|—  
5   tracks/|  
6     |+page.svelte    → /tracks|  
7     |[id]/|  
8       |+page.svelte  → /tracks/:id|—  
9   lessons/|  
10    |+page.svelte    → /lessons|  
11    |[slug]/|  
12      |+page.svelte  → /lessons/:slug|—
```

8.1.2 Route Files

File	Purpose
+page.svelte	The page component
+page.ts	Client-side data loading
+page.server.ts	Server-side data loading
+layout.svelte	Shared layout wrapper
+layout.server.ts	Layout data loading
+server.ts	API endpoint (no UI)
+error.svelte	Error page

8.1.3 Dynamic Parameters

```

1 // src/routes/tracks/[id]/+page.server.ts
2 export async function load({ params }) {
3     // params.id = whatever is in the URL
4     const trackId = params.id; // "hadith"
5
6     // Fetch track...
7 }
```

8.1.4 Multiple Parameters

```

1 src/routes/[locale]/lessons/[slug]/+page.svelte→
2 /en/lessons/hadith-jibril→
3 params = { locale: 'en', slug: 'hadith-jibril' }
```

8.1.5 Optional Parameters

```

1 src/routes/lessons/[[page]]/+page.svelte→
2 /lessons      → params.page = undefined→
3 /lessons/2    → params.page = '2'
```

8.1.6 Rest Parameters

```

1 src/routes/files/...path/+page.svelte→
2 /files/docs/images/photo.png→
3 params.path = 'docs/images/photo.png'
```

8.2 8.2 Server vs Client Components

Understanding where code runs is crucial.

8.2.1 Server-Only Code

Files with `+page.server.ts` or in `$lib/server/`:

```

1 // src/routes/admin/+page.server.ts
2 import { db } from '$lib/server/db'; // ↴ Server only
3
4 export async function load() {
```

```

5   // This runs ONLY on the server
6   const users = db.select().from(users).all();
7
8   return { users };
9 }
```

8.2.2 Client Code

Regular .svelte files:

```

1 <!-- src/routes/+page.svelte -->
2 <script>
3   // This runs in the browser
4   import { onMount } from 'svelte';
5
6   onMount(() => {
7     console.log('Browser only!');
8   });
9 </script>
```

8.2.3 Universal Code

Files with +page.ts (no .server):

```

1 // src/routes/+page.ts
2 export async function load({ fetch }) {
3   // Runs on server (first load) AND client (navigation)
4   const res = await fetch('/api/data');
5   return { data: await res.json() };
6 }
```

8.2.4 When to Use Each

Type	Use When
+page.server.ts	Database queries, secrets, auth
+page.ts	Public API calls, caching
Component	UI logic, interactivity

8.3 Load Functions

Load functions fetch data before rendering.

8.3.1 Server Load

```

1 // src/routes/tracks/+page.server.ts
2 import type { PageServerLoad } from './$types';
3 import { contentDatabase, tracks } from '$lib/server/db';
4
5 export const load: PageServerLoad = async () => {
6     const allTracks = contentDatabase
7         .select()
8         .from(tracks)
9         .orderBy(tracks.order)
10        .all();
11
12     return {
13         tracks: allTracks,
14     };
15 };

```

8.3.2 Using Load Data

```

1 <!-- src/routes/tracks/+page.svelte -->
2 <script lang="ts">
3     import type { PageData } from './$types';
4
5     let { data }: { data: PageData } = $props();
6
7     // TypeScript knows data.tracks is Track[]
8 </script>
9
10    {#each data.tracks as track}
11        <h2>{track.title}</h2>
12    {/each}

```

8.3.3 Accessing User from Locals

```

1 // src/routes/dashboard/+page.server.ts
2 export const load: PageServerLoad = async ({ locals }) => {
3     // locals.user comes from hooks.server.ts
4     if (!locals.user) {
5         throw redirect(302, '/auth/login');
6     }
7
8     return {
9         user: locals.user,
10    };
11 };

```

8.3.4 Dependent Data

```

1 // src/routes/lessons/[slug]/+page.server.ts
2 export const load: PageServerLoad = async ({ params, locals }) => {
3     // Get lesson

```

```

4   const lesson = contentDatabase
5     .select()
6     .from(lessons)
7     .where(eq(lessons.slug, params.slug))
8     .get();
9
10    if (!lesson) {
11      throw error(404, 'Lesson not found');
12    }
13
14    // Get videos for this lesson
15    const lessonVideos = contentDatabase
16      .select()
17      .from(videos)
18      .where(eq(videos.lessonId, lesson.id))
19      .orderBy(videos.order)
20      .all();
21
22    // Get user progress (if logged in)
23    let progress = null;
24    if (locals.user) {
25      progress = db
26        .select()
27        .from(lessonProgress)
28        .where(
29          and(
30            eq(lessonProgress.userId, locals.user.id),
31            eq(lessonProgress.lessonId, lesson.id)
32          )
33        )
34        .get();
35    }
36
37    return {
38      lesson,
39      videos: lessonVideos,
40      progress,
41      isAuthenticated: !!locals.user,
42    };
43  };

```

8.4 8.4 Form Actions

Handle form submissions server-side.

8.4.1 Basic Form

```

1 <!-- src/routes/auth/login/+page.svelte -->
2 <form method="POST">
3   <input type="email" name="email" required />
4   <input type="password" name="password" required />
5   <button type="submit">Login</button>
6 </form>

```

8.4.2 Action Handler

```

1 // src/routes/auth/login/+page.server.ts
2 import type { Actions } from './$types';
3 import { fail, redirect } from '@sveltejs/kit';
4
5 export const actions: Actions = {
6     default: async ({ request, cookies }) => {
7         const form = await request.formData();
8         const email = form.get('email') as string;
9         const password = form.get('password') as string;
10
11         // Validate
12         if (!email || !password) {
13             return fail(400, {
14                 error: 'Email and password required',
15                 email, // Return email to repopulate form
16             });
17         }
18
19         // Check user
20         const user = await validateUser(email, password);
21         if (!user) {
22             return fail(400, {
23                 error: 'Invalid credentials',
24                 email,
25             });
26         }
27
28         // Create session
29         const { sessionId, expiresAt } = createSession(user.id);
30         setSessionCookie(cookies, sessionId, expiresAt);
31
32         throw redirect(302, '/');
33     }
34 };

```

8.4.3 Named Actions

```

1 <form method="POST" action="/login">...</form>
2 <form method="POST" action="/register">...</form>

```

```

1 export const actions: Actions = {
2     login: async ({ request }) => { /* ... */ },
3     register: async ({ request }) => { /* ... */ },
4 };

```

8.4.4 Handling Action Results

```

1 <script lang="ts">
2     import type { ActionData } from './$types';
3
4     let { form }: { form: ActionData } = $props();
5 </script>
6
7 {#if form?.error}

```

```
8     <div class="alert alert-error">{form.error}</div>
9 {/if}
10
11 <form method="POST">
12   <input
13     type="email"
14     name="email"
15     value={form?.email ?? ''}
16   />
17   <!-- ... -->
18 </form>
```

8.4.5 Progressive Enhancement

```
1 <script>
2   import { enhance } from '$app/forms';
3 </script>
4
5 <form method="POST" use:enhance>
6   <!-- Works without JS, better with JS -->
7 </form>
```

With `use:enhance:` - Form submits via fetch (no page reload) - Automatic loading states - Better UX

8.5 8.5 Error Pages

Handle errors gracefully.

8.5.1 Route-Level Error Page

```
1 <!-- src/routes/lessons/[slug]/+error.svelte -->
2 <script>
3   import { page } from '$app/stores';
4 </script>
5
6 <div class="error-container">
7   <h1>{$page.status}</h1>
8   <p>{$page.error?.message}</p>
9   <a href="/">Go Home</a>
10 </div>
```

8.5.2 Global Error Page

```

1 <!-- src/routes/+error.svelte -->
2 <script>
3   import { page } from '$app/stores';
4 </script>
5
6 <div class="min-h-screen flex items-center justify-center">
7   <div class="text-center">
8     <h1 class="text-6xl font-bold text-error">
9       {$page.status}
10    </h1>
11    <p class="text-xl mt-4">
12      {#if $page.status === 404}
13        ةجْمَلْيَةٌ مُّعَذَّبَةٌ
14      {:else if $page.status === 500}
15        مُّدَافِعٌ مُّنْهَاجٌ فَيَخْلُجُ
16      {:else}
17        {$page.error?.message}
18      {/if}
19    </p>
20    <a href="/" class="btn btn-primary mt-8">
21      ةَدَوْغَلْيَةٌ مُّسَمِّدَةٌ
22    </a>
23  </div>
24</div>

```

8.5.3 Throwing Errors in Load

```

1 import { error } from '@sveltejs/kit';
2
3 export const load: PageServerLoad = async ({ params }) => {
4   const lesson = getLesson(params.slug);
5
6   if (!lesson) {
7     throw error(404, 'Lesson not found');
8   }
9
10  if (!lesson.isPublished) {
11    throw error(403, 'This lesson is not available');
12  }
13
14  return { lesson };
15};

```

8.6 8.6 Layout Nesting

Share UI across routes with layouts.

8.6.1 Root Layout

```

1 <!-- src/routes/+layout.svelte -->
2 <script>
3     import Navbar from '$lib/components/Navbar.svelte';
4     import Footer from '$lib/components/Footer.svelte';
5     import '../app.css';
6
7     let { children, data } = $props();
8 </script>
9
10 <Navbar user={data.user} />
11
12 <main class="min-h-screen">
13     {@render children()}
14 </main>
15
16 <Footer />
```

8.6.2 Layout Data

```

1 // src/routes/+layout.server.ts
2 export async function load({ locals }) {
3     return {
4         user: locals.user,
5     };
6 }
```

8.6.3 Nested Layouts

```

1 src/routes/|—
2   +layout.svelte      # Root (Navbar, Footer) |—
3   admin/|—
4     |— +layout.svelte    # Admin (Sidebar) |
5     |— +page.svelte      # /admin|
6     |— lessons/|—
7       |— +page.svelte    # /admin/lessons
```

```

1 <!-- src/routes/admin/+layout.svelte -->
2 <script>
3     let { children } = $props();
4 </script>
5
6 <div class="flex">
7     <aside class="w-64 bg-base-200">
8         <nav>
9             <a href="/admin">Dashboard</a>
10            <a href="/admin/lessons">Lessons</a>
11            <a href="/admin/videos">Videos</a>
12        </nav>
13    </aside>
14
15    <main class="flex-1 p-8">
16        {@render children()}
17    </main>
18 </div>
```

8.6.4 Breaking Out of Layout

Use `+page@.svelte` to reset to root:

```
1 src/routes/|  
2 +layout.svelte          # Has Navbar |  
3 auth/  
4   └── login/  
5     └── +page@.svelte  # No Navbar (uses root HTML only)
```

8.7 Summary

Concept	Key Points
Routing	File-based, <code>[param]</code> for dynamic
Load	<code>+page.server.ts</code> for data fetching
Actions	Form handling with <code>use:enhance</code>
Layouts	Nested, data flows down
Errors	Custom error pages, <code>throw error()</code>

SvelteKit fundamentals are covered. Next: building beautiful UIs.

Next Chapter: [Chapter 9: UI with DaisyUI & Tailwind](#)

Chapter 9

UI with DaisyUI & Tailwind

9.1 9.1 Setting Up Tailwind CSS 4

Tailwind v4 brings a simpler setup.

9.1.1 Installation

```
1 bun add -d tailwindcss @tailwindcss/vite
```

9.1.2 Vite Configuration

```
1 // vite.config.ts
2 import { sveltekit } from '@sveltejs/kit/vite';
3 import tailwindcss from '@tailwindcss/vite';
4 import { defineConfig } from 'vite';
5
6 export default defineConfig({
7   plugins: [
8     tailwindcss(),
9     sveltekit()
10   ]
11});
```

9.1.3 CSS Entry Point

```
1 /* src/app.css */
2 @import 'tailwindcss';
```

9.1.4 Import in Layout

```
1 <!-- src/routes/+layout.svelte -->
2 <script>
3   import '../app.css';
4 </script>
```

9.1.5 Tailwind Basics

```
1 <!-- Spacing -->
2 <div class="p-4 m-2">Padding 4, Margin 2</div>
3
4 <!-- Flexbox -->
5 <div class="flex items-center justify-between">...</div>
6
7 <!-- Grid -->
8 <div class="grid grid-cols-3 gap-4">...</div>
9
10 <!-- Colors -->
11 <div class="bg-blue-500 text-white">Blue background</div>
12
13 <!-- Responsive -->
14 <div class="text-sm md:text-base lg:text-lg">Responsive text</div>
```

9.2 9.2 DaisyUI Components

DaisyUI adds beautiful, ready-to-use components.

9.2.1 Installation

```
1 bun add -d daisyui
```

9.2.2 Configuration

```
1 /* src/app.css */
2 @import 'tailwindcss';
3 @plugin 'daisyui';
```

9.2.3 Core Components

Buttons:

```

1 <button class="btn">Default</button>
2 <button class="btn btn-primary">Primary</button>
3 <button class="btn btn-secondary">Secondary</button>
4 <button class="btn btn-ghost">Ghost</button>
5 <button class="btn btn-outline">Outline</button>
6 <button class="btn btn-sm">Small</button>
7 <button class="btn btn-lg">Large</button>

```

Cards:

```

1 <div class="card bg-base-100 shadow-xl">
2   <figure></figure>
3   <div class="card-body">
4     <h2 class="card-title">Title</h2>
5     <p>Description</p>
6     <div class="card-actions justify-end">
7       <button class="btn btn-primary">Action</button>
8     </div>
9   </div>
10 </div>

```

Badges:

```

1 <span class="badge">Default</span>
2 <span class="badge badge-primary">Primary</span>
3 <span class="badge badge-lg">Large</span>

```

Alerts:

```

1 <div class="alert alert-success">Success message</div>
2 <div class="alert alert-error">Error message</div>
3 <div class="alert alert-warning">Warning message</div>
4 <div class="alert alert-info">Info message</div>

```

Form Controls:

```

1 <input type="text" class="input input-bordered w-full" />
2 <select class="select select-bordered">
3   <option>Option 1</option>
4 </select>
5 <textarea class="textarea textarea-bordered"></textarea>

```

Navigation:

```

1 <div class="navbar bg-base-100">
2   <div class="flex-1">
3     <a class="btn btn-ghost text-xl">Brand</a>
4   </div>
5   <div class="flex-none">
6     <ul class="menu menu-horizontal px-1">
7       <li><a>Link 1</a></li>
8       <li><a>Link 2</a></li>
9     </ul>
10   </div>
11 </div>

```

9.3 9.3 Theme System (Light/Dark)

DaisyUI includes multiple themes.

9.3.1 Available Themes

```
1 light, dark, cupcake, bumblebee, emerald, corporate,
2 synthwave, retro, cyberpunk, valentine, halloween,
3 garden, forest, aqua, lofi, pastel, fantasy, wireframe,
4 black, luxury, dracula, cmyk, autumn, business, acid,
5 lemonade, night, coffee, winter, dim, nord, sunset
```

9.3.2 Setting Theme

```
1 <!-- In app.html -->
2 <html data-theme="dark">
```

9.3.3 Dynamic Theme Switching

```
1 <!-- ThemeSwitcher.svelte -->
2 <script lang="ts">
3     import { browser } from '$app/environment';
4
5     let theme = $state(
6         browser ? localStorage.getItem('theme') || 'light' : 'light'
7     );
8
9     function toggleTheme() {
10         theme = theme === 'light' ? 'dark' : 'light';
11         document.documentElement.setAttribute('data-theme', theme);
12         localStorage.setItem('theme', theme);
13     }
14 </script>
15
16 <button class="btn btn-ghost" onclick={toggleTheme}>
17     {theme === 'light' ? ☀️ : ⚡}
18 </button>
```

9.3.4 Initialize Theme

```
1 <!-- +layout.svelte -->
2 <script>
3     import { browser } from '$app/environment';
4     import { onMount } from 'svelte';
5
6     onMount(() => {
7         const saved = localStorage.getItem('theme') || 'light';
8         document.documentElement.setAttribute('data-theme', saved);
```

```
9     });
10 </script>
```

9.4 9.4 RTL Support for Arabic

Arabic text flows right-to-left.

9.4.1 HTML Direction

```
1 <!-- app.html -->
2 <html lang="ar" dir="rtl">
```

9.4.2 Dynamic Direction

```
1 <script>
2   let locale = $state('ar');
3 </script>
4
5 <svelte:body dir={locale === 'ar' ? 'rtl' : 'ltr'} />
```

9.4.3 RTL Utilities

```
1 /* src/app.css */
2 @layer utilities {
3   /* Flip margins/paddings */
4   [dir="rtl"] .ml-4 {
5     margin-left: 0;
6     margin-right: 1rem;
7   }
8
9   /* Text alignment */
10  [dir="rtl"] {
11    text-align: right;
12  }
13}
```

9.4.4 Logical Properties (CSS)

Modern CSS supports logical properties:

```

1 /* Instead of margin-left/right */
2 .element {
3     margin-inline-start: 1rem; /* Left in LTR, Right in RTL */
4     margin-inline-end: 1rem;   /* Right in LTR, Left in RTL */
5 }

```

9.4.5 DaisyUI RTL

DaisyUI works well with RTL. Just set `dir="rtl"`:

```

1 <div class="navbar bg-base-100" dir="rtl">
2     <!-- Components auto-adjust -->
3 </div>

```

9.5 9.5 Responsive Design Patterns

Mobile-first approach.

9.5.1 Breakpoints

Prefix	Width	Devices
(none)	< 640px	Mobile
sm:	≥ 640px	Large phones
md:	≥ 768px	Tablets
lg:	≥ 1024px	Laptops
xl:	≥ 1280px	Desktops
2xl:	≥ 1536px	Large screens

9.5.2 Common Patterns

Responsive Grid:

```

1 <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
2     <!-- 1 col on mobile, 2 on tablet, 3 on desktop -->
3 </div>

```

Hidden on Mobile:

```
1 <div class="hidden md:block">Desktop only</div>
2 <div class="md:hidden">Mobile only</div>
```

Responsive Text:

```
1 <h1 class="text-2xl md:text-4xl lg:text-5xl">Responsive Title</h1>
```

Responsive Padding:

```
1 <div class="p-4 md:p-8 lg:p-12">Grows with screen</div>
```

9.5.3 Mobile Navigation Pattern

```
1 <script>
2     let isMenuOpen = $state(false);
3 </script>
4
5 <!-- Desktop Nav -->
6 <nav class="hidden md:flex gap-4">
7     <a href="/">Home</a>
8     <a href="/tracks">Tracks</a>
9 </nav>
10
11 <!-- Mobile Menu Button -->
12 <button class="md:hidden" onclick={() => isMenuOpen = !isMenuOpen}>
13     ≡
14 </button>
15
16 <!-- Mobile Drawer -->
17 {#if isMenuOpen}
18     <div class="fixed inset-0 z-50 md:hidden">
19         <div class="bg-black/50" onclick={() => isMenuOpen = false}></div>
20         <nav class="fixed right-0 top-0 h-full w-64 bg-base-100 p-4">
21             <a href="/">Home</a>
22             <a href="/tracks">Tracks</a>
23         </nav>
24     </div>
25 {/if}
```

9.6 9.6 Custom Component Library

Build reusable components with consistent styling.

9.6.1 Button Component

```

1 <!-- src/lib/components/Button.svelte -->
2 <script lang="ts">
3   interface Props {
4     variant?: 'primary' | 'secondary' | 'ghost' | 'outline';
5     size?: 'sm' | 'md' | 'lg';
6     loading?: boolean;
7     disabled?: boolean;
8     type?: 'button' | 'submit';
9     onclick?: () => void;
10    }
11
12  let {
13    variant = 'primary',
14    size = 'md',
15    loading = false,
16    disabled = false,
17    type = 'button',
18    onclick,
19    children,
20  }: Props & { children: any } = $props();
21
22  const variantClasses = {
23    primary: 'btn-primary',
24    secondary: 'btn-secondary',
25    ghost: 'btn-ghost',
26    outline: 'btn-outline',
27  };
28
29  const sizeClasses = {
30    sm: 'btn-sm',
31    md: '',
32    lg: 'btn-lg',
33  };
34 </script>
35
36 <button
37   {type}
38   class="btn {variantClasses[variant]} {sizeClasses[size]}"
39   disabled={disabled || loading}
40   {onclick}
41 >
42   {#if loading}
43     <span class="loading loading-spinner loading-sm"></span>
44   {/if}
45   {@render children()}
46 </button>

```

9.6.2 Card Component

```

1 <!-- src/lib/components/Card.svelte -->
2 <script lang="ts">
3   interface Props {
4     title?: string;
5     image?: string;
6     href?: string;
7     class?: string;
8   }
9
10  let { title, image, href, class: className = '', children }: Props & { children: any } =
$props();

```

```
11 </script>
12
13 <svelte:element
14     this={href ? 'a' : 'div'}
15     {href}
16     class="card bg-base-100 shadow-lg hover:shadow-xl transition-shadow {className}"
17 >
18     {#if image}
19         <figure>
20             <img src={image} alt={title || ''} class="w-full h-48 object-cover" />
21         </figure>
22     {/if}
23     <div class="card-body">
24         {#if title}
25             <h3 class="card-title">{title}</h3>
26         {/if}
27         {@render children()}
28     </div>
29 </svelte:element>
```

9.6.3 Input Component

```
1 <!-- src/lib/components/Input.svelte -->
2 <script lang="ts">
3     interface Props {
4         label?: string;
5         name: string;
6         type?: 'text' | 'email' | 'password' | 'number';
7         placeholder?: string;
8         value?: string;
9         error?: string;
10        required?: boolean;
11    }
12
13    let {
14        label,
15        name,
16        type = 'text',
17        placeholder,
18        value = '',
19        error,
20        required = false,
21    }: Props = $props();
22 </script>
23
24 <div class="form-control w-full">
25     {#if label}
26         <label class="label" for={name}>
27             <span class="label-text">{label}</span>
28             {#if required}
29                 <span class="text-error">*</span>
30             {/if}
31         </label>
32     {/if}
33
34     <input
35         {type}
36         {name}
37         id={name}
38         {placeholder}
```

```

39      {value}
40      {required}
41      class="input input-bordered w-full {error ? 'input-error' : ''}"
42    />
43
44    {#if error}
45      <label class="label">
46        <span class="label-text-alt text-error">{error}</span>
47      </label>
48    {/if}
49  </div>

```

9.6.4 Usage

```

1 <script>
2   import Button from '$lib/components/Button.svelte';
3   import Card from '$lib/components/Card.svelte';
4   import Input from '$lib/components/Input.svelte';
5 </script>
6
7 <Button variant="primary" loading={isSubmitting}>
8   Submit
9 </Button>
10
11 <Card title="Lesson Title" image="/cover.jpg" href="/lessons/1">
12   <p>Lesson description...</p>
13 </Card>
14
15 <Input
16   label="Email"
17   name="email"
18   type="email"
19   error={form?.errors?.email}
20   required
21 />

```

9.7 Summary

Topic	Key Points
Tailwind	Utility-first, v4 simpler setup
DaisyUI	Ready components, themes
Themes	data-theme attribute, localStorage
RTL	dir="rtl", logical CSS properties
Responsive	Mobile-first, breakpoint prefixes
Components	Reusable, typed props

Our UI foundation is ready. Next: building the actual pages.

Next Chapter: [Chapter 10: Building Core Pages](#)

Chapter 10

Building Core Pages

10.1 10.1 Homepage with Dynamic Stats

The homepage is your platform's first impression.

10.1.1 Load Function

```
1 // src/routes/+page.server.ts
2 import type { PageServerLoad } from './$types';
3 import { contentDatabase, tracks, series, lessons, videos } from '$lib/server/db';
4 import { count, eq, desc } from 'drizzle-orm';
5
6 export const load: PageServerLoad = async () => {
7     // Get counts
8     const tracksCount = contentDatabase
9         .select({ count: count() })
10        .from(tracks)
11        .get()?.count ?? 0;
12
13     const seriesCount = contentDatabase
14         .select({ count: count() })
15        .from(series)
16        .get()?.count ?? 0;
17
18     const lessonsCount = contentDatabase
19         .select({ count: count() })
20        .from(lessons)
21        .where(eq(lessons.isPublished, true))
22        .get()?.count ?? 0;
23
24     const videosCount = contentDatabase
25         .select({ count: count() })
26        .from(videos)
27        .get()?.count ?? 0;
28
29     // Get latest lessons
30     const latestLessons = contentDatabase
31         .select()
32        .from(lessons)
33        .where(eq(lessons.isPublished, true))
34        .orderBy(desc(lessons.createdAt))
```

```

35     .limit(6)
36     .all();
37
38     // Get all tracks for display
39     const allTracks = contentDatabase
40         .select()
41         .from(tracks)
42         .orderBy(tracks.order)
43         .all();
44
45     return {
46       stats: {
47         tracks: tracksCount,
48         series: seriesCount,
49         lessons: lessonsCount,
50         videos: videosCount,
51       },
52       latestLessons,
53       tracks: allTracks,
54     };
55 }

```

10.1.2 Homepage Component

```

1 <!-- src/routes/+page.svelte -->
2 <script lang="ts">
3   import type { PageData } from './$types';
4   import { getTrackColor, getTrackIcon } from '$lib/theme';
5
6   let { data }: { data: PageData } = $props();
7 </script>
8
9 <svelte:head>
10    <title>أفدن - مكتبة كلية التربية والفنون الجميلة</title>
11 </svelte:head>
12
13 <!-- Hero Section -->
14 <section class="hero min-h-[70vh] bg-gradient-to-br from-primary to-primary-focus text-primary
15   -content">
16   <div class="hero-content text-center">
17     <div class="max-w-2xl">
18       <h1 class="text-5xl font-bold mb-6">
19         أفدن مكتبة كلية
20       </h1>
21       <p class="text-xl mb-8 opacity-90">
22         كلية التربية والفنون الجميلة كلية
23       </p>
24       <div class="flex gap-4 justify-center">
25         <a href="/tracks" class="btn btn-secondary btn-lg">
26           ابدأ
27         </a>
28         <a href="/about" class="btn btn-ghost btn-lg">
29           قرارات الكلية
30         </a>
31       </div>
32     </div>
33   </section>
34
35 <!-- Stats Section -->

```

```

36 <section class="py-16 bg-base-200">
37     <div class="container mx-auto px-4">
38         <div class="stats stats-vertical lg:stats-horizontal shadow w-full">
39             <div class="stat">
40                 <div class="stat-figure text-primary text-3xl" style="background-color: #f0f0f0; border-radius: 50%; width: 100px; height: 100px; display: flex; align-items: center; justify-content: center; font-size: 1.5em; margin-bottom: 5px;">سیارک{data.stats.tracks}</div>
43             </div>
44             <div class="stat">
45                 <div class="stat-figure text-secondary text-3xl" style="background-color: #e0e0e0; border-radius: 50%; width: 100px; height: 100px; display: flex; align-items: center; justify-content: center; font-size: 1.5em; margin-bottom: 5px;">سیارک{data.stats.series}</div>
48             </div>
49             <div class="stat">
50                 <div class="stat-figure text-accent text-3xl" style="background-color: #d0c0c0; border-radius: 50%; width: 100px; height: 100px; display: flex; align-items: center; justify-content: center; font-size: 1.5em; margin-bottom: 5px;">سیارک{data.stats.lessons}</div>
53             </div>
54             <div class="stat">
55                 <div class="stat-figure text-info text-3xl" style="background-color: #c0d0c0; border-radius: 50%; width: 100px; height: 100px; display: flex; align-items: center; justify-content: center; font-size: 1.5em; margin-bottom: 5px;">سیارک{data.stats.videos}</div>
58             </div>
59         </div>
60     </div>
61 </section>
62
63 <!-- Tracks Section -->
64 <section class="py-16 bg-base-100">
65     <div class="container mx-auto px-4">
66         <h2 class="text-3xl font-bold text-center mb-12" style="color: #f0f0f0;">سیارک{getTrackIcon(track.id, track.icon)}{track.title}</h3>
74                     {#if track.description}
75                         <p class="opacity-90" style="color: #f0f0f0;">{track.description}</p>
76                     {/if}
77                 </div>
78             </a>
79         {/each}
80     </div>
81 </div>
82 </div>
83 </div>
84 </div>
85 </section>
86
87 <!-- Latest Lessons -->
88 <section class="py-16 bg-base-200">
89     <div class="container mx-auto px-4">
90         <h2 class="text-3xl font-bold text-center mb-12" style="color: #f0f0f0;">سیارک{lesson.title}</h3>
97                         {#if lesson.description}

```

```

98         <p class="text-base-content/70 line-clamp-2">{lesson.description
99     }</p>
100    {/if}
101    {#if lesson.instructor}
102        <p class="text-sm text-base-content/60"> {lesson.instructor}</p>
103    {/if}
104    </div>
105    {/each}
106  </div>
107
108  <div class="text-center mt-8">
109      <a href="/lessons" class="btn btn-primary">
110           ﻊﻓﺮÇ ﻢﻗرಡ
111      </a>
112  </div>
113</div>
114</section>
```

10.2 10.2 Content Listing (Tracks, Lessons)

List pages with filtering and search.

10.2.1 Tracks Listing

```

1 // src/routes/tracks/+page.server.ts
2 export const load: PageServerLoad = async () => {
3     const allTracks = contentDatabase
4         .select()
5         .from(tracks)
6         .orderBy(tracks.order)
7         .all();
8
9     // Get lesson count per track
10    const tracksWithCount = allTracks.map(track => {
11        const lessonCount = contentDatabase
12            .select({ count: count() })
13            .from(lessons)
14            .where(and(
15                eq(lessons.trackId, track.id),
16                eq(lessons.isPublished, true)
17            ))
18            .get()?.count ?? 0;
19
20        return { ...track, lessonCount };
21    });
22
23    return { tracks: tracksWithCount };
24};
```

10.2.2 Lessons Listing with Search

```

1 <!-- src/routes/lessons/+page.svelte -->
2 <script lang="ts">
3     import type { PageData } from './$types';
4
5     let { data }: { data: PageData } = $props();
6
7     let searchQuery = $state('');
8     let selectedTrack = $state('all');
9
10    const filteredLessons = $derived(
11        data.lessons.filter(lesson => {
12            const matchesSearch = lesson.title.toLowerCase()
13                .includes(searchQuery.toLowerCase());
14            const matchesTrack = selectedTrack === 'all' ||
15                lesson.trackId === selectedTrack;
16            return matchesSearch && matchesTrack;
17        })
18    );
19 </script>
20
21 <section class="py-8">
22     <div class="container mx-auto px-4">
23         <!-- Filters -->
24         <div class="card bg-base-100 shadow mb-8">
25             <div class="card-body">
26                 <div class="flex flex-wrap gap-4 items-center">
27                     <input
28                         type="text"
29                         bind:value={searchQuery}
30                         placeholder="جستجوی... "
31                         class="input input-bordered flex-1 min-w-[200px]"
32                     />
33
34                     <select bind:value={selectedTrack} class="select select-bordered">
35                         <option value="all">همه آنلاین کارهای...

```

```
61
62     {#if filteredLessons.length === 0}
63         <div class="text-center py-16">
64             <span class="text-6xl mb-4 block" style="color: #f0f0f0;"></span>
65             <h3 class="text-xl font-bold" style="color: #f0f0f0;">ناتیجہ توجیہ</h3>
66             <p class="text-base-content" style="color: #f0f0f0;">۷۰٪ مکالمات میں اپنے فرمائیں</p>
67         </div>
68     {/if}
69 </div>
70 </section>
```

10.3 10.3 Detail Pages

Single item pages with rich content.

10.3.1 Single Track Page

```
1 // src/routes/tracks/[id]/+page.server.ts
2 export const load: PageServerLoad = async ({ params }) => {
3     const track = contentDatabase
4         .select()
5         .from(tracks)
6         .where(eq(tracks.id, params.id))
7         .get();
8
9     if (!track) {
10         throw error(404, 'Track not found');
11     }
12
13     // Get series in this track
14     const trackSeries = contentDatabase
15         .select()
16         .from(series)
17         .where(eq(series.trackId, params.id))
18         .orderBy(series.order)
19         .all();
20
21     // Get standalone lessons (no series)
22     const standaloneLessons = contentDatabase
23         .select()
24         .from(lessons)
25         .where(and(
26             eq(lessons.trackId, params.id),
27             isNull(lessons.seriesId),
28             eq(lessons.isPublished, true)
29         ))
30         .orderBy(lessons.order)
31         .all();
32
33     return {
34         track,
35         series: trackSeries,
```

```
36     standaloneLessons,
37   };
38 }
```

10.3.2 Single Lesson Page

See Chapter 12 for the full VideoPlayer integration.

10.4 Search Functionality

Global search across all content.

```
1 // src/routes/search/+page.server.ts
2 export const load: PageServerLoad = async ({ url }) => {
3   const query = url.searchParams.get('q')?.trim() || '';
4
5   if (query.length < 2) {
6     return { query: '', results: { lessons: [], tracks: [] } };
7   }
8
9   const pattern = `%${query}%`;
10
11  const lessonResults = contentDatabase
12    .select()
13    .from(lessons)
14    .where(and(
15      eq(lessons.isPublished, true),
16      or(
17        like(lessons.title, pattern),
18        like(lessons.description, pattern)
19      )
20    ))
21    .limit(20)
22    .all();
23
24  const trackResults = contentDatabase
25    .select()
26    .from(tracks)
27    .where(or(
28      like(tracks.title, pattern),
29      like(tracks.description, pattern)
30    ))
31    .all();
32
33  return {
34    query,
35    results: {
36      lessons: lessonResults,
37      tracks: trackResults,
38    },
39  };
40}
```

```

1 <!-- src/routes/search/+page.svelte -->
2 <script lang="ts">
3   import { goto } from '$app/navigation';
4
5   let { data } = $props();
6   let searchInput = $state(data.query);
7
8   function handleSearch(e: Event) {
9     e.preventDefault();
10    if (searchInput.trim().length >= 2) {
11      goto(`/search?q=${encodeURIComponent(searchInput)})`);
12    }
13  }
14 </script>
15
16 <form onsubmit={handleSearch} class="max-w-xl mx-auto mb-8">
17   <div class="join w-full">
18     <input
19       type="text"
20       bind:value={searchInput}
21       placeholder="..." 
22       class="input input-bordered join-item flex-1"
23     />
24     <button class="btn btn-primary join-item"></button>
25   </div>
26 </form>
27
28 {#if data.results.lessons.length > 0}
29   <h2 class="text-xl font-bold mb-4"></h2>
30   <!-- Render lessons... -->
31 {/if}
32
33 {#if data.results.tracks.length > 0}
34   <h2 class="text-xl font-bold mb-4"></h2>
35   <!-- Render tracks... -->
36 {/if}

```

10.5 User Dashboard

Personal progress page.

```

1 // src/routes/dashboard/+page.server.ts
2 export const load: PageServerLoad = async ({ locals }) => {
3   if (!locals.user) {
4     throw redirect(302, '/auth/login?redirect=/dashboard');
5   }
6
7   const progress = db
8     .select()
9     .from(lessonProgress)
10    .where(eq(lessonProgress.userId, locals.user.id))
11    .all();
12
13   const completedCount = progress.filter(p => p.videoCompleted).length;
14   const totalWatchTime = progress.reduce((sum, p) => sum + p.watchedSeconds, 0);

```

```

15
16     return {
17         user: locals.user,
18         stats: {
19             started: progress.length,
20             completed: completedCount,
21             totalMinutes: Math.floor(totalWatchTime / 60),
22         },
23         recentProgress: progress.slice(0, 10),
24     };
25 }

```

10.6 10.6 Error and 404 Pages

Handle errors gracefully.

```

1 <!-- src/routes/+error.svelte -->
2 <script>
3     import { page } from '$app/stores';
4 </script>
5
6 <div class="min-h-screen flex items-center justify-center bg-base-200">
7     <div class="text-center p-8">
8         <div class="text-8xl mb-6">
9             {#if $page.status === 404}
10                🚫
11            {:else if $page.status === 403}
12                🔒
13            {:else}
14                ▲
15           {/if}
16        </div>
17
18        <h1 class="text-4xl font-bold mb-4">
19            {#if $page.status === 404}
20                ةدوام حلاق رارم
21            {:else if $page.status === 403}
22                رارم حلاق ملوك
23            {:else}
24                ملوك حلاق
25           {/if}
26        </h1>
27
28        <p class="text-base-content/70 mb-8">
29            {$page.error?.message || 'ملوك حلاق عذوقتكم'}
30        </p>
31
32        <a href="/" class="btn btn-primary">
33            ملوك حلاق
34        </a>
35    </div>
36 </div>

```

10.7 Summary

Page	Features
Homepage	Stats, tracks, latest lessons
Listings	Search, filter, pagination
Detail	Full content, related items
Search	Global search with results
Dashboard	User progress, stats
Error	Friendly error messages

Core pages are complete. Next: reusable components.

Next Chapter: [Chapter 11: Reusable Components](#)

Chapter 11

Reusable Components

11.1 11.1 Navbar with Auth State

A responsive navbar that adapts to authentication status.

```
1 <!-- src/lib/components/Navbar.svelte -->
2 <script lang="ts">
3     import type { User } from '$lib/server/db/schema';
4     import ThemeSwitcher from './ThemeSwitcher.svelte';
5
6     interface Props {
7         user: User | null;
8     }
9
10    let { user }: Props = $props();
11    let mobileMenuOpen = $state(false);
12
13    const navLinks = [
14        { href: '/', label: 'خانه' },
15        { href: '/tracks', label: 'آهنگ‌ها' },
16        { href: '/lessons', label: 'دروس' },
17        { href: '/search', label: 'جستجو' },
18    ];
19 </script>
20
21 <nav class="navbar bg-base-100 shadow-lg sticky top-0 z-50">
22     <div class="container mx-auto">
23         <!-- Logo -->
24         <div class="flex-1">
25             <a href="/" class="btn btn-ghost text-xl font-bold">
26                 فدن
27             </a>
28         </div>
29
30         <!-- Desktop Menu -->
31         <div class="hidden md:flex items-center gap-2">
32             {#each navLinks as link}
33                 <a href={link.href} class="btn btn-ghost btn-sm">
34                     {link.label}
35                 </a>
36             {/each}
37
38             <div class="divider divider-horizontal mx-2"></div>
39
40             <ThemeSwitcher />
```

```
41      {#if user}
42          <div class="dropdown dropdown-end">
43              <button class="btn btn-ghost btn-circle avatar">
44                  <div class="w-10 rounded-full bg-primary text-primary-content flex
45 items-center justify-center">
46                      {user.name.charAt(0)}
47                  </div>
48              </button>
49              <ul class="dropdown-content z-[1] menu p-2 shadow bg-base-100 rounded-box
w-52">
50                  <li class="menu-title">{user.name}</li>
51                  <li><a href="/dashboard">  </a></li>
52                  {#if user.role === 'admin' || user.role === 'editor'}
53                      <li><a href="/admin">  </a></li>
54                  {/if}
55                  <li>
56                      <form method="POST" action="/auth/logout">
57                          <button type="submit" class="w-full text-right">
58                               
59                          </button>
60                      </form>
61                  </li>
62              </ul>
63          </div>
64      {:else}
65          <a href="/auth/login" class="btn btn-primary btn-sm">
66               
67          </a>
68      {/if}
69  </div>
70
71  <!-- Mobile Menu Button -->
72  <button
73      class="btn btn-ghost md:hidden"
74      onclick={() => mobileMenuOpen = !mobileMenuOpen}
75  >
76      {mobileMenuOpen ? '☰' : '≡'}
77  </button>
78 </div>
79 </nav>
80
81 <!-- Mobile Drawer -->
82 {#if mobileMenuOpen}
83     <div class="fixed inset-0 z-40 md:hidden">
84         <!-- Backdrop -->
85         <div
86             class="absolute inset-0 bg-black/50"
87             onclick={() => mobileMenuOpen = false}
88         ></div>
89
90         <!-- Menu -->
91         <div class="absolute right-0 top-0 h-full w-72 bg-base-100 shadow-xl p-4">
92             <div class="flex justify-between items-center mb-8">
93                 <span class="text-xl font-bold" style="color: #007bff;">! 
```

```

103     {#each navLinks as link}
104         <li>
105             <a href={link.href} onclick={() => mobileMenuOpen = false}>
106                 {link.label}
107             </a>
108         </li>
109     {/each}
110 </ul>
111
112     <div class="divider"></div>
113
114     {#if user}
115         <div class="p-4 bg-base-200 rounded-lg mb-4">
116             <p class="font-bold">{user.name}</p>
117             <p class="text-sm text-base-content/70">{user.email}</p>
118         </div>
119         <form method="POST" action="/auth/logout">
120             <button class="btn btn-outline btn-block">Logout</button>
121         </form>
122     {:else}
123         <a href="/auth/login" class="btn btn-primary btn-block">
124             جوچسے جوچاں!
125         </a>
126     {:if}
127     </div>
128 </div>
129 {/if}

```

11.2 Footer Component

```

1 <!-- src/lib/components/Footer.svelte -->
2 <script lang="ts">
3     const currentYear = new Date().getFullYear();
4
5     const links = {
6         main: [
7             { href: '/tracks', label: 'آرٹریکس' },
8             { href: '/lessons', label: 'ویڈیو' },
9             { href: '/about', label: 'نئی عوامیں' },
10        ],
11        legal: [
12            { href: '/privacy', label: 'خصوصیات' },
13            { href: '/terms', label: 'بُروجیں' },
14        ],
15    };
16 </script>
17
18 <footer class="footer footer-center p-10 bg-base-200 text-base-content">
19     <nav class="grid grid-flow-col gap-4">
20         {#each links.main as link}
21             <a href={link.href} class="link link-hover">{link.label}</a>
22         {/each}
23     </nav>
24
25     <nav>
26         <div class="grid grid-flow-col gap-4">

```

```

27    <!-- Social Icons -->
28    <a href="https://twitter.com" target="_blank" class="btn btn-ghost btn-circle">
29        
30    </a>
31    <a href="https://youtube.com" target="_blank" class="btn btn-ghost btn-circle">
32        
33    </a>
34    <a href="https://telegram.org" target="_blank" class="btn btn-ghost btn-circle">
35        
36    </a>
37  </div>
38 </nav>
39
40 <aside>
41     <p>© {currentYear} Afidna | All Rights Reserved</p>
42 </aside>
43 </footer>

```

11.3 11.3 Card Components

11.3.1 LessonCard

```

1 <!-- src/lib/components/LessonCard.svelte -->
2 <script lang="ts">
3     import type { Lesson } from '$lib/server/db/schema-content';
4     import { getTrackColor } from '$lib/theme';
5
6     interface Props {
7         lesson: Lesson & { track?: { id: string; title: string } };
8         progress?: number;
9     }
10
11     let { lesson, progress = 0 }: Props = $props();
12 </script>
13
14 <a
15     href="/lessons/{lesson.slug}"
16     class="card bg-base-100 shadow-lg hover:shadow-xl transition-all hover:-translate-y-1"
17 >
18     {#if lesson.thumbnailUrl}
19         <figure>
20             <img src={lesson.thumbnailUrl} alt={lesson.title} class="h-40 w-full object-cover"
21         />
22         </figure>
23     {/if}
24
25     <div class="card-body">
26         {#if lesson.track}
27             <span class="badge badge-sm {getTrackColor(lesson.track.id)}">
28                 {lesson.track.title}
29             </span>
30         {/if}
31
32         <h3 class="card-title line-clamp-2">{lesson.title}</h3>

```

```

32
33     {#if lesson.description}
34         <p class="text-sm text-base-content/70 line-clamp-2">
35             {lesson.description}
36         </p>
37     {/if}

38
39     {#if lesson.instructor}
40         <p class="text-sm"> {lesson.instructor}</p>
41     {/if}

42
43     {#if progress > 0}
44         <div class="mt-2">
45             <progress
46                 class="progress progress-primary w-full"
47                 value={progress}
48                 max="100"
49             ></progress>
50             <span class="text-xs text-base-content/60">{progress}% 完成</span>
51         </div>
52     {/if}
53 </div>
54 </a>
```

11.3.2 TrackCard

```

1 <!-- src/lib/components/TrackCard.svelte -->
2 <script lang="ts">
3     import type { Track } from '$lib/server/db/schema-content';
4     import { getTrackColor, getTrackIcon } from '$lib/theme';
5
6     interface Props {
7         track: Track & { lessonCount?: number };
8     }
9
10    let { track }: Props = $props();
11 </script>
12
13 <a
14     href="/tracks/{track.id}"
15     class="card bg-gradient-to-br {getTrackColor(track.id)} text-white shadow-lg hover:scale-105 transition-transform"
16 >
17     <div class="card-body">
18         <span class="text-5xl">{getTrackIcon(track.id, track.icon)}</span>
19         <h3 class="card-title text-white">{track.title}</h3>
20         {#if track.description}
21             <p class="opacity-90 line-clamp-2">{track.description}</p>
22         {/if}
23         {#if track.lessonCount !== undefined}
24             <div class="card-actions justify-end">
25                 <span class="badge badge-ghost">{track.lessonCount} 课时</span>
26             </div>
27         {/if}
28     </div>
29 </a>
```

11.4 11.4 Modal System

```

1 <!-- src/lib/components/Modal.svelte -->
2 <script lang="ts">
3     interface Props {
4         open: boolean;
5         title?: string;
6         onClose: () => void;
7     }
8
9     let { open, title, onClose, children }: Props & { children: any } = $props();
10
11    function handleKeydown(e: KeyboardEvent) {
12        if (e.key === 'Escape' && open) {
13            onClose();
14        }
15    }
16 </script>
17
18 <svelte:window onkeydown={handleKeydown} />
19
20 {#if open}
21     <div class="modal modal-open">
22         <!-- Backdrop -->
23         <div class="modal-backdrop bg-black/50" onclick={onClose}></div>
24
25         <!-- Modal Box -->
26         <div class="modal-box relative">
27             <!-- Close Button -->
28             <button
29                 class="btn btn-sm btn-circle btn-ghost absolute right-2 top-2"
30                 onclick={onClose}>
31                 >
32                 X
33             </button>
34
35             {#if title}
36                 <h3 class="font-bold text-lg mb-4">{title}</h3>
37             {/if}
38
39             {@render children() }
40         </div>
41     </div>
42 {/if}

```

11.4.1 Usage

```

1 <script>
2     import Modal from '$lib/components/Modal.svelte';
3
4     let showModal = $state(false);
5 </script>
6
7 <button class="btn" onclick={() => showModal = true}>
8     Open Modal
9 </button>
10
11 <Modal
12     open={showModal}

```

```
13     title="Confirm Action"
14     onClose={() => showModal = false}
15 >
16     <p>Are you sure you want to continue?</p>
17     <div class="modal-action">
18       <button class="btn btn-ghost" onclick={() => showModal = false}>
19         Cancel
20       </button>
21       <button class="btn btn-primary">
22         Confirm
23       </button>
24     </div>
25   </Modal>
```

11.5 11.5 Toast Notifications

```
1 <!-- src/lib/components/Toast.svelte -->
2 <script lang="ts">
3   import { fly } from 'svelte/transition';
4
5   interface Toast {
6     id: string;
7     message: string;
8     type: 'success' | 'error' | 'warning' | 'info';
9   }
10
11  let toasts = $state<Toast[]>([]);
12
13  export function showToast(message: string, type: Toast['type'] = 'info') {
14    const id = crypto.randomUUID();
15    toasts = [...toasts, { id, message, type }];
16
17    setTimeout(() => {
18      toasts = toasts.filter(t => t.id !== id);
19    }, 4000);
20  }
21
22  const icons = {
23    success: '✓',
24    error: '✗',
25    warning: '⚠',
26    info: 'ℹ',
27  };
28
29  const classes = {
30    success: 'alert-success',
31    error: 'alert-error',
32    warning: 'alert-warning',
33    info: 'alert-info',
34  };
35 </script>
36
37 <div class="toast toast-top toast-end z-50">
38   {#each toasts as toast (toast.id)}
39     <div
40       class="alert {classes[toast.type]}"
```

```

41         transition:fly={{ x: 100, duration: 300 }}
42     >
43         <span class="text-xl">{icons[toast.type]}</span>
44         <span>{toast.message}</span>
45     </div>
46  {/each}
47</div>

```

11.5.1 Usage with Store

```

1 // src/lib/stores/toast.ts
2 import { writable } from 'svelte/store';
3
4 interface Toast {
5     id: string;
6     message: string;
7     type: 'success' | 'error' | 'warning' | 'info';
8 }
9
10 function createToastStore() {
11     const { subscribe, update } = writable<Toast[]>([]);
12
13     return {
14         subscribe,
15         show(message: string, type: Toast['type'] = 'info') {
16             const id = crypto.randomUUID();
17             update(toasts => [...toasts, { id, message, type }]);
18
19             setTimeout(() => {
20                 update(toasts => toasts.filter(t => t.id !== id));
21             }, 4000);
22         },
23         success(message: string) { this.show(message, 'success'); },
24         error(message: string) { this.show(message, 'error'); },
25         warning(message: string) { this.show(message, 'warning'); },
26     };
27 }
28
29 export const toast = createToastStore();

```

11.6 11.6 Loading States

```

1 <!-- src/lib/components>LoadingSpinner.svelte -->
2 <script lang="ts">
3     interface Props {
4         size?: 'sm' | 'md' | 'lg';
5         text?: string;
6     }
7
8     let { size = 'md', text }: Props = $props();
9
10    const sizeClasses = {

```

```

11     sm: 'loading-sm',
12     md: 'loading-md',
13     lg: 'loading-lg',
14   };
15 </script>
16
17 <div class="flex flex-col items-center justify-center gap-2">
18   <span class="loading loading-spinner {sizeClasses[size]}></span>
19   {#if text}
20     <span class="text-sm text-base-content/70">{text}</span>
21   {/if}
22 </div>

```

11.6.1 Skeleton Loading

```

1 <!-- src/lib/components/SkeletonCard.svelte -->
2 <div class="card bg-base-100 shadow">
3   <div class="card-body">
4     <div class="skeleton h-4 w-24 mb-2"></div>
5     <div class="skeleton h-6 w-full mb-4"></div>
6     <div class="skeleton h-4 w-full"></div>
7     <div class="skeleton h-4 w-3/4"></div>
8   </div>
9 </div>

```

11.6.2 Page Loading

```

1 <!-- src/routes/+layout.svelte -->
2 <script>
3   import { navigating } from '$app/stores';
4   import LoadingSpinner from '$lib/components>LoadingSpinner.svelte';
5 </script>
6
7 {#if $navigating}
8   <div class="fixed inset-0 bg-base-100/80 z-50 flex items-center justify-center">
9     <LoadingSpinner size="lg" text="جاری..."/>
10   </div>
11 {/if}

```

11.7 Summary

Component	Use Case
Navbar	Navigation, auth state, mobile menu
Footer	Links, social, copyright

Component	Use Case
Cards	Display content items
Modal	Dialogs, confirmations
Toast	Notifications, feedback
Loading	Spinners, skeletons

Part III is complete! We've built the entire frontend foundation. Next: advanced features.

Next Chapter: [Part IV → Chapter 12: Video Player Integration](#)

Chapter 12

Video Player Integration

12.1 12.1 YouTube IFrame API

We'll embed YouTube videos with full control over playback.

12.1.1 Loading the API

```
1 // src/lib/utils/youtube.ts
2 let apiLoaded = false;
3 let apiLoadPromise: Promise<void> | null = null;
4
5 export function loadYouTubeAPI(): Promise<void> {
6     if (apiLoaded) return Promise.resolve();
7     if (apiLoadPromise) return apiLoadPromise;
8
9     apiLoadPromise = new Promise((resolve) => {
10         const script = document.createElement('script');
11         script.src = 'https://www.youtube.com/iframe_api';
12
13         (window as any).onYouTubeIframeAPIReady = () => {
14             apiLoaded = true;
15             resolve();
16         };
17
18         document.head.appendChild(script);
19     });
20
21     return apiLoadPromise;
22 }
```

12.1.2 API Reference

```
1 interface YT.Player {
2     playVideo(): void;
3     pauseVideo(): void;
4     seekTo(seconds: number): void;
5     getCurrentTime(): number;
6     getDuration(): number;
```

```
7     getPlayerState(): number;
8 }
9
10 // Player States
11 const PlayerState = {
12     UNSTARTED: -1,
13     ENDED: 0,
14     PLAYING: 1,
15     PAUSED: 2,
16     BUFFERING: 3,
17     CUED: 5,
18 };
```

12.2 12.2 Building VideoPlayer Component

```
1 <!-- src/lib/components/VideoPlayer.svelte -->
2 <script lang="ts">
3     import { onMount, onDestroy } from 'svelte';
4     import { loadYouTubeAPI } from '$lib/utils/youtube';
5     import { browser } from '$app/environment';
6
7     interface Props {
8         youtubeId: string;
9         title?: string;
10        startTime?: number;
11        onProgress?: (seconds: number) => void;
12        onComplete?: () => void;
13    }
14
15    let {
16        youtubeId,
17        title = 'Video',
18        startTime = 0,
19        onProgress,
20        onComplete
21    }: Props = $props();
22
23    let containerRef: HTMLDivElement;
24    let player: YT.Player | null = null;
25    let progressInterval: number | null = null;
26    let lastReportedTime = 0;
27
28    onMount(async () => {
29        if (!browser || !youtubeId) return;
30
31        await loadYouTubeAPI();
32        initPlayer();
33    });
34
35    onDestroy(() => {
36        if (progressInterval) clearInterval(progressInterval);
37        player?.destroy();
38    });
39
40    function initPlayer() {
41        player = new YT.Player(containerRef, {
```

```
42     videoId: youtubeId,
43     playerVars: {
44         autoplay: 0,
45         modestbranding: 1,
46         rel: 0,
47         start: Math.floor(startTime),
48     },
49     events: {
50         onReady: handleReady,
51         onStateChange: handleStateChange,
52     },
53 });
54 }
55
56 function handleReady() {
57     console.log('Player ready');
58 }
59
60 function handleStateChange(event: YT.OnStateChangeEvent) {
61     const state = event.data;
62
63     if (state === YT.PlayerState.PLAYING) {
64         startProgressTracking();
65     } else if (state === YT.PlayerState.PAUSED) {
66         stopProgressTracking();
67     } else if (state === YT.PlayerState.ENDED) {
68         stopProgressTracking();
69         onComplete?.();
70     }
71 }
72
73 function startProgressTracking() {
74     if (progressInterval) return;
75
76     progressInterval = setInterval(() => {
77         if (!player) return;
78
79         const currentTime = player.getCurrentTime();
80
81         // Report every 30 seconds
82         if (currentTime - lastReportedTime >= 30) {
83             lastReportedTime = currentTime;
84             onProgress?.(currentTime);
85         }
86     }, 5000) as unknown as number;
87 }
88
89 function stopProgressTracking() {
90     if (progressInterval) {
91         clearInterval(progressInterval);
92         progressInterval = null;
93     }
94
95     // Report final time
96     if (player) {
97         onProgress?.(player.getCurrentTime());
98     }
99 }
100</script>
101
102<div class="video-player">
103    <div class="aspect-video bg-black rounded-lg overflow-hidden">
104        <div bind:this={containerRef} class="w-full h-full"></div>
105    </div>
```

```
106     {#if title}
107         <h3 class="text-lg font-bold mt-3">{title}</h3>
108     {/if}
109 </div>
```

12.3 12.3 Progress Tracking

12.3.1 The 10-Minute Rule

```
1 // src/lib/utils/progress.ts
2
3 const TEN_MINUTES = 600; // seconds
4
5 export function shouldMarkComplete(
6     watchedSeconds: number,
7     videoDuration: number
8 ): boolean {
9     // Complete if watched 10+ minutes OR reached end
10    return watchedSeconds >= TEN_MINUTES ||
11        watchedSeconds >= videoDuration * 0.9;
12 }
```

12.3.2 Reporting Progress

```
1 async function reportProgress(lessonId: string, seconds: number) {
2     try {
3         await fetch('/api/progress', {
4             method: 'POST',
5             headers: { 'Content-Type': 'application/json' },
6             body: JSON.stringify({
7                 lessonId,
8                 watchedSeconds: Math.floor(seconds),
9             }),
10            });
11        } catch (error) {
12            console.error('Failed to report progress:', error);
13        }
14    }
```

12.4 12.4 Handling Events

```
1 <script>
2     function handleProgress(seconds: number) {
3         // Save progress
4         reportProgress(lesson.id, seconds);
5     }
6
7     function handleComplete() {
8         // Mark as complete
9         fetch('/api/progress', {
10             method: 'POST',
11             body: JSON.stringify({
12                 lessonId: lesson.id,
13                 videoCompleted: true,
14             }),
15         });
16
17         // Move to next video or show quiz
18         if (hasQuiz) {
19             showQuiz = true;
20         } else if (nextVideo) {
21             currentVideoIndex++;
22         }
23     }
24 </script>
25
26 <VideoPlayer
27     youtubeId={video.youtubeId}
28     title={video.title}
29     startTime={progress?.watchedSeconds || 0}
30     {onProgress}
31     onComplete={handleComplete}
32 />
```

12.5 12.5-12.7 Advanced Features

12.5.1 Quality Selection (handled by YouTube)

YouTube automatically handles quality based on connection.

12.5.2 Fullscreen Support

```
1 <button
2     class="btn btn-ghost btn-sm"
3     onclick={() => containerRef.requestFullscreen()}
4 >
5     [<] Fullscreen
6 </button>
```

Chapter 13

Interactive Quiz System

13.1 13.1 Quiz Data Structure

```
1 interface Question {
2     id: string;
3     text: string;
4     options: string[];
5     correctIndex: number;
6     points?: number;
7 }
8
9 interface Quiz {
10    questions: Question[];
11    passingScore: number; // percentage
12 }
```

13.1.1 Storing in Database

```
1 -- In videos table
2 quiz TEXT -- JSON array of questions
```

13.2 13.2-13.7 Quiz Component

See `src/lib/components/Quiz.svelte` for the full implementation:

- Multiple choice with A/B/C/D options
- Instant feedback (correct/incorrect)
- Score calculation
- Pass/fail determination

- Automatic progression
-

Chapter 14

Progress Tracking

14.1 14.1-14.6 Progress System

14.1.1 Database Schema

```
1 CREATE TABLE lesson_progress (
2     id INTEGER PRIMARY KEY,
3     user_id INTEGER NOT NULL,
4     lesson_id TEXT NOT NULL,
5     watched_seconds INTEGER DEFAULT 0,
6     video_completed INTEGER DEFAULT 0,
7     quiz_passed INTEGER DEFAULT 0,
8     quiz_score INTEGER DEFAULT 0,
9     updated_at TEXT,
10    UNIQUE(user_id, lesson_id)
11 );
```

14.1.2 UPSERT Logic

```
1 // Check if exists
2 const existing = db.select()...
3
4 if (existing) {
5     db.update(lessonProgress)
6         .set({
7             watchedSeconds: Math.max(existing.watchedSeconds, newSeconds),
8             ...
9         })
10        .where(eq(lessonProgress.id, existing.id));
11 } else {
12     db.insert(lessonProgress).values({...});
13 }
```

Chapter 15

Global Search

15.1 15.1-15.6 Search Implementation

15.1.1 SQLite LIKE Search

```
1 const pattern = `%"${query}"%`;
2
3 const results = contentDatabase
4     .select()
5     .from(lessons)
6     .where(or(
7         like(lessons.title, pattern),
8         like(lessons.description, pattern)
9     ))
10    .all();
```

15.1.2 Search Page

```
1 <form onsubmit={handleSearch}>
2     <input bind:value={query} placeholder="Search..." />
3     <button>Search</button>
4 </form>
5
6 {#each results.lessons as lesson}
7     <LessonCard {lesson} />
8 {/each}
```

Next Part: Part V → Chapter 16: Admin Dashboard

Chapter 16

Part V: Admin Panel

16.1 Chapter 16: Admin Dashboard

16.1.1 16.1 Access Control

```
1 // src/routes/admin/+layout.server.ts
2 import { redirect } from '@sveltejs/kit';
3 import { isEditor } from '$lib/server/auth';
4
5 export async function load({ locals }) {
6     if (!locals.user) {
7         throw redirect(302, '/auth/login?redirect=/admin');
8     }
9
10    if (!isEditor(locals.user)) {
11        throw redirect(302, '/');
12    }
13
14    return { user: locals.user };
15}
```

16.1.2 16.2-16.5 Dashboard Implementation

```
1 <!-- src/routes/admin/+page.svelte -->
2 <script lang="ts">
3     let { data } = $props();
4 </script>
5
6 <div class="grid grid-cols-1 md:grid-cols-4 gap-6 mb-8">
7     <div class="stat bg-base-100 rounded-lg shadow">
8         <div class="stat-title">الإجمالي:</div>
9         <div class="stat-value text-primary">{data.stats.tracks}</div>
10    </div>
11    <div class="stat bg-base-100 rounded-lg shadow">
12        <div class="stat-title">السلسلات:</div>
13        <div class="stat-value text-secondary">{data.stats.series}</div>
```

```

14     </div>
15     <div class="stat bg-base-100 rounded-lg shadow">
16       <div class="stat-title" data-i="icon-stat-lessons"></div>
17       <div class="stat-value text-accent">{data.stats.lessons}</div>
18     </div>
19     <div class="stat bg-base-100 rounded-lg shadow">
20       <div class="stat-title" data-i="icon-stat-videos"></div>
21       <div class="stat-value text-info">{data.stats.videos}</div>
22     </div>
23   </div>
24
25   <!-- Quick Actions -->
26   <div class="grid grid-cols-2 md:grid-cols-4 gap-4">
27     <a href="/admin/lessons" class="btn btn-outline" data-i="icon-admin-lessons"> دروس </a>
28     <a href="/admin/videos" class="btn btn-outline" data-i="icon-admin-videos"> فيديوهات </a>
29     <a href="/admin/series" class="btn btn-outline" data-i="icon-admin-series"> سلسلات </a>
30     <a href="/admin/tracks" class="btn btn-outline" data-i="icon-admin-tracks"> مسار </a>
31   </div>

```

16.2 Chapter 17: Content Management

16.2.1 17.1-17.6 CRUD Operations

```

1 // src/routes/admin/lessons/[id]/+page.server.ts
2 export const actions: Actions = {
3   update: async ({ request, params, locals }) => {
4     const form = await request.formData();
5
6     contentDatabase
7       .update(lessons)
8       .set({
9         title: form.get('title') as string,
10        description: form.get('description') as string,
11        updatedAt: new Date().toISOString(),
12        lastModifiedBy: locals.user.id.toString(),
13      })
14     .where(eq(lessons.id, params.id))
15     .run();
16
17     return { success: true };
18   },
19
20   toggleLock: async ({ params, locals }) => {
21     if (locals.user.role !== 'admin') {
22       return fail(403, { error: 'Admin only' });
23     }
24
25     const lesson = contentDatabase
26       .select()
27       .from(lessons)
28       .where(eq(lessons.id, params.id))
29       .get();
30
31     contentDatabase

```

```
32     .update(lessons)
33     .set({ isLocked: !lesson.isLocked })
34     .where(eq(lessons.id, params.id))
35     .run();
36
37     return { success: true };
38 },
39 };
```

16.3 Chapter 18: Audit & Protection

16.3.1 18.1-18.5 Audit Trail

```
1 // All content tables have:
2 {
3     createdAt: TEXT,           // When created
4     updatedAt: TEXT,          // Last modified
5     lastModifiedBy: TEXT,    // 'script' or user ID
6     isLocked: INTEGER,       // Prevent changes
7 }
8
9 // Check before update:
10 function canModify(item, forceMode = false) {
11     if (item.isLocked) return false;
12     if (item.lastModifiedBy !== 'script' && !forceMode) return false;
13     return true;
14 }
```

Chapter 17

Part VI: Content Pipeline

17.1 Chapter 19: Content Seeding

17.1.1 19.1-19.6 Seed Script

```
1 // scripts/seed-content.ts
2 async function seedLesson(lesson, forceMode) {
3     const existing = contentDatabase
4         .select()
5         .from(lessons)
6         .where(eq(lessons.id, lesson.id))
7         .get();
8
9     if (existing) {
10         // Check if can update
11         if (existing.isLocked) {
12             console.log(` Skipping locked: ${lesson.id}`);
13             return;
14         }
15         if (existing.lastModifiedBy !== 'script' && !forceMode) {
16             console.log(` Skipping human edit: ${lesson.id}`);
17             return;
18         }
19
20         // Update
21         contentDatabase.update(lessons).set({...lesson}).run();
22     } else {
23         // Insert
24         contentDatabase.insert(lessons).values({...lesson}).run();
25     }
26 }
```

17.2 Chapter 20: Media Management

17.2.1 20.1 Hybrid Video Strategy

We use a **hybrid approach**: - **YouTube**: Streaming (watching) - free, fast CDN - **Cloudflare R2**: Download (offline) - cheap storage, free egress

17.2.2 20.2 Cloudflare R2 Setup

Why R2? - S3-compatible API - **Free egress** (no bandwidth charges!) - Global CDN via Cloudflare - ~\$0.015/GB storage

Setup Steps:

1. Create bucket: afidna-assets
2. Enable **Public Access** or Custom Domain
3. Get public URL: <https://assets.afidna.com>

17.2.3 20.3 File Structure

```

1 afidna-assets/
2   videos/{track}/{lesson}/{order}.mp4
3   pdfs/{track}/{lesson}.pdf
4   thumbnails/{lesson}.webp

```

17.2.4 20.4 Database Integration

```

1 -- videos table stores direct URLs
2 downloadUrl = 'https://assets.afidna.com/videos/hadith/jibril/01.mp4'
3
4 -- lessons table for PDFs
5 pdfUrl = 'https://assets.afidna.com/pdfs/hadith/jibril.pdf'

```

17.2.5 20.5 No SDK Needed!

For **public download**, we just store URLs in the database. No `aws-sdk` or any library required!

```

1 // In content.json
2 {
3     "youtubeId": "abc123",      // ڈاکٹ ایڈیڈ
4     "downloadUrl": "https://" // جسے حکم دیں
5 }

```

17.2.6 20.6 Upload via rclone

```
1 # Configure rclone for R2
2 rclone config
3 # Provider: Cloudflare R2
4
5 # Upload
6 rclone sync ./processed/ r2:afidna-assets/
```

17.2.7 20.7 Future: Admin Upload (????)

Only if you need browser uploads:

```
1 import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';
2 import { getSignedUrl } from '@aws-sdk/s3-request-presigner';
3
4 export async function getUploadUrl(key: string) {
5     const command = new PutObjectCommand({
6         Bucket: 'afidna-assets',
7         Key: key,
8     });
9     return getSignedUrl(R2, command, { expiresIn: 3600 });
10 }
```

Chapter 18

Part VII: Testing & Quality

18.1 Chapter 21: Testing Strategy

18.1.1 Unit Tests with Bun

```
1 // tests/auth.test.ts
2 import { describe, expect, test } from 'bun:test';
3 import { hashPassword, verifyPassword } from '$lib/server/auth';
4
5 describe('Auth', () => {
6     test('password hashing', async () => {
7         const password = 'test123';
8         const hash = await hashPassword(password);
9
10        expect(hash).not.toBe(password);
11        expect(await verifyPassword(password, hash)).toBe(true);
12        expect(await verifyPassword('wrong', hash)).toBe(false);
13    });
14});
```

18.1.2 Run Tests

```
1 bun test
```

18.2 Chapter 22: Code Quality

18.2.1 ESLint + Prettier

```
1 // .prettierrc
2 {
3     "tabWidth": 4,
4     "singleQuote": true,
5     "trailingComma": "es5"
6 }
```

```
1 bun run format  
2 bun run lint
```

Chapter 19

Part VIII: Deployment

19.1 Chapter 23-26: Production Deployment

19.1.1 Build

```
1 bun run build
```

19.1.2 VPS Deployment

```
1 # Ubuntu server
2 sudo apt update
3 curl -fsSL https://bun.sh/install | bash
4 git clone <repo>
5 cd afidna
6 bun install
7 bun run db:setup
8 bun run db:content
9 bun run build
10
11 # Run with systemd
12 sudo systemctl enable afidna
13 sudo systemctl start afidna
```

19.1.3 Caddy Config

```
1 afidna.com {
2     reverse_proxy localhost:3000
3 }
```

Chapter 20

Part IX: Scaling & Optimization

20.1 Chapter 27-28: Performance

20.1.1 Database Optimization

```
1 // Use indexes
2 CREATE INDEX idx_lessons_slug ON lessons(slug);
3
4 // Use specific selects
5 db.select({ id, title }).from(lessons);
6
7 // Use limits
8 .limit(10)
```

20.1.2 Caching

```
1 // Cache headers
2 return new Response(body, {
3     headers: {
4         'Cache-Control': 'public, max-age=3600',
5     },
6 }) ;
```

Chapter 21

Part X: Appendices

21.1 Appendix A: Code Reference

All schemas, APIs, and components are in the codebase.

21.2 Appendix B: Commands

```
1 # Development
2 bun run dev
3
4 # Database
5 bun run db:setup
6 bun run db:content
7 bun run seed
8
9 # Build
10 bun run build
11 bun run preview
```

21.3 Appendix C: Troubleshooting

Problem	Solution
Port in use	<code>kill -9 \$(lsof -ti:5173)</code>
DB locked	Stop other processes
Build fails	Check TypeScript errors

21.4 Appendix D: Resources

- SvelteKit Docs
- Drizzle Docs

- [DaisyUI Docs](#)
 - [Tailwind Docs](#)
-

End of Book

Thank you for reading! Build something amazing. [?](#)