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 18

1.5.1 Part I: Foundation & Planning (Chapters 1-3) 18

1.5.2 Part II: Backend Fundamentals (Chapters 4-7) 18

1.5.3 Part III: Frontend Development (Chapters 8-11) 18

1.5.4 Part IV: Advanced Features (Chapters 12-15) 20

1.5.5 Part V: Admin Panel (Chapters 16-18) 20

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

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 21
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 23
2.2 2.2 The "Compact Monolith" Philosophy 23
2.2.1 Key Principles 23
2.3 2.3 Dual Database Strategy 24
2.3.1 Why Two Databases? 24
2.3.2 The Implementation 24
2.3.3 Benefits 24
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 26
2.5.1 Scaling Strategies 27
2.5.2 Our Approach 27
2.6 2.6 Security Considerations from Day One 27
2.6.1 Authentication Security 27
2.6.2 Authorization Layers 29

2.6.3 Content Protection 29
2.6.4 Rate Limiting (Placeholder) 29
2.6.5 SQL Injection Prevention 29
2.7 Summary 29

3 Setting Up the Development Environment 31

3.1 3.1 Installing Bun 31
3.1.1 What is Bun? 31
3.1.2 Installation 31
3.1.3 Why Bun Over Node.js? 32
3.2 3.2 Editor Setup (VS Code / Cursor) 32
3.2.1 VS Code Installation 32
3.2.2 Cursor (AI-Powered Alternative) 32
3.2.3 Recommended Settings 32
3.3 3.3 Essential Extensions 33
3.3.1 Must-Have 33
3.3.2 Recommended 33
3.3.3 Installation via CLI 33
3.4 3.4 Git Configuration 34
3.4.1 Initial Configuration 34
3.4.2 SSH Key Setup (for GitHub) 34
3.4.3 Project .gitignore 34
3.5 3.5 Project Initialization 35
3.5.1 Create Project Directory 35
3.5.2 Initialize with Bun + SvelteKit 35
3.5.3 Install Dependencies 36
3.5.4 SvelteKit Configuration 36

3.5.5 Tailwind Setup . . . 36
3.6 3.6 Directory Structure Best Practices . . . 37
3.6.1 The Complete Structure . . . 37
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? . . . 42
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 46
4.5 4.5 Audit Trail Implementation 46
4.5.1 The Fields 47
4.5.2 Complete Table Definition 47
4.5.3 Using the Audit Trail 47
4.6 4.6 Indexing Strategies 48
4.6.1 When to Index 48
4.6.2 Our Indexes 48
4.6.3 Composite Indexes 49
4.7 4.7 WAL Mode and Performance 49
4.7.1 Enabling WAL Mode 49
4.7.2 What WAL Does 49
4.7.3 Other Performance PRAGMAs 49
4.7.4 When to Use WAL 50
4.8 Summary 50

5 Drizzle ORM Setup 51

5.1 5.1 Introduction to Drizzle 51
5.1.1 Why Drizzle? 51
5.1.2 Core Philosophy 51
5.1.3 Installation 51
5.2 5.2 Schema Definition 52
5.2.1 users.db Schema 52
5.2.2 content.db Schema 53

5.3 5.3 Type-Safe Queries . . . 54
5.3.1 Basic Select . . . 54
5.3.2 Select Specific Columns . . . 55
5.3.3 Insert . . . 55
5.3.4 Update . . . 55
5.3.5 Delete . . . 55
5.4 5.4 Migrations Strategy . . . 56
5.4.1 Why Not Traditional Migrations? . . . 56
5.4.2 Our Approach: Setup Scripts . . . 56
5.4.3 Schema Changes . . . 57
5.5 5.5 Connecting Multiple Databases . . . 57
5.5.1 Connection Module . . . 57
5.5.2 Usage in SvelteKit . . . 58
5.6 5.6 Query Optimization . . . 58
5.6.1 Use Specific Selects . . . 59
5.6.2 Use Limits . . . 59
5.6.3 Use Joins Instead of N+1 . . . 59
5.6.4 Count Queries . . . 59
5.6.5 Aggregations . . . 60
5.7 Summary . . . 60

6 Authentication System 61

6.1 6.1 Session-Based vs JWT . . . 61
6.1.1 JWT (JSON Web Tokens) . . . 61
6.1.2 Session-Based . . . 61
6.1.3 Our Choice: Sessions . . . 61
6.2 6.2 Password Hashing with Bun.password . . . 63

6.2.1 Why Argon2id? . . . 63
6.2.2 Implementation . . . 63
6.2.3 Usage . . . 63
6.3 6.3 Session Management . . . 64
6.3.1 Session Flow . . . 64
6.3.2 Session Schema . . . 64
6.3.3 Session Functions . . . 64
6.4 6.4 Cookie Security (HttpOnly, SameSite) . . . 66
6.4.1 Cookie Options . . . 66
6.4.2 Option Explanations . . . 66
6.5 6.5 Remember Me Functionality . . . 66
6.5.1 Implementation . . . 67
6.5.2 Login Form . . . 67
6.6 6.6 Role-Based Access Control (RBAC) . . . 67
6.6.1 Roles . . . 67
6.6.2 Schema . . . 68
6.6.3 Authorization Helpers . . . 68
6.6.4 Usage in Load Functions . . . 68
6.7 6.7 Protected Routes with Hooks . . . 69
6.7.1 The Hook . . . 69
6.7.2 Type Definitions . . . 69
6.7.3 Accessing User in Routes . . . 70
6.7.4 Layout Data . . . 70
6.8 Complete Auth Module . . . 70
6.9 Summary . . . 72

7 API Design 73

7.1 7.1 SvelteKit Server Routes . . . 73
7.1.1 Route Types . . . 73
7.1.2 API Routes Location . . . 73
7.1.3 Basic Endpoint Structure . . . 73
7.2 7.2 RESTful Endpoints . . . 74
7.2.1 HTTP Methods . . . 74
7.2.2 Progress API Example . . . 74
7.3 7.3 Request Validation . . . 76
7.3.1 Manual Validation . . . 76
7.3.2 Validation Helper . . . 77
7.4 7.4 Error Handling . . . 77
7.4.1 Error Response Format . . . 77
7.4.2 Error Responses . . . 78
7.4.3 Try-Catch Wrapper . . . 78
7.5 7.5 Rate Limiting . . . 79
7.5.1 Simple In-Memory Rate Limiter . . . 79
7.5.2 Usage in Hooks . . . 80
7.6 7.6 CORS Configuration . . . 80
7.6.1 When You Need CORS . . . 80
7.6.2 SvelteKit CORS . . . 81
7.6.3 For Our Platform . . . 81
7.7 Summary . . . 81

8 SvelteKit Fundamentals 83

8.1 8.1 File-Based Routing . . . 83
8.1.1 Basic Routing . . . 83
8.1.2 Route Files . . . 83

8.1.3 Dynamic Parameters . . . 84
8.1.4 Multiple Parameters . . . 84
8.1.5 Optional Parameters . . . 84
8.1.6 Rest Parameters . . . 84
8.2 8.2 Server vs Client Components . . . 84
8.2.1 Server-Only Code . . . 84
8.2.2 Client Code . . . 85
8.2.3 Universal Code . . . 85
8.2.4 When to Use Each . . . 85
8.3 8.3 Load Functions . . . 85
8.3.1 Server Load . . . 86
8.3.2 Using Load Data . . . 86
8.3.3 Accessing User from Locals . . . 86
8.3.4 Dependent Data . . . 86
8.4 8.4 Form Actions . . . 87
8.4.1 Basic Form . . . 87
8.4.2 Action Handler . . . 88
8.4.3 Named Actions . . . 88
8.4.4 Handling Action Results . . . 88
8.4.5 Progressive Enhancement . . . 89
8.5 8.5 Error Pages . . . 89
8.5.1 Route-Level Error Page . . . 89
8.5.2 Global Error Page . . . 89
8.5.3 Throwing Errors in Load . . . 90
8.6 8.6 Layout Nesting . . . 90
8.6.1 Root Layout . . . 90
8.6.2 Layout Data . . . 91

8.6.3 Nested Layouts 91
8.6.4 Breaking Out of Layout 92
8.7 Summary 92

9 UI with DaisyUI & Tailwind 93

9.1 9.1 Setting Up Tailwind CSS 4 93
9.1.1 Installation 93
9.1.2 Vite Configuration 93
9.1.3 CSS Entry Point 93
9.1.4 Import in Layout 93
9.1.5 Tailwind Basics 94
9.2 9.2 DaisyUI Components 94
9.2.1 Installation 94
9.2.2 Configuration 94
9.2.3 Core Components 94
9.3 9.3 Theme System (Light/Dark) 96
9.3.1 Available Themes 96
9.3.2 Setting Theme 96
9.3.3 Dynamic Theme Switching 96
9.3.4 Initialize Theme 96
9.4 9.4 RTL Support for Arabic 97
9.4.1 HTML Direction 97
9.4.2 Dynamic Direction 97
9.4.3 RTL Utilities 97
9.4.4 Logical Properties (CSS) 97
9.4.5 DaisyUI RTL 98
9.5 9.5 Responsive Design Patterns 98

9.5.1 Breakpoints . . . 98
9.5.2 Common Patterns . . . 98
9.5.3 Mobile Navigation Pattern . . . 99
9.6 9.6 Custom Component Library . . . 99
9.6.1 Button Component . . . 99
9.6.2 Card Component . . . 100
9.6.3 Input Component . . . 101
9.6.4 Usage . . . 102
9.7 Summary . . . 102

10 Building Core Pages 104

10.1 10.1 Homepage with Dynamic Stats . . . 104
10.1.1 Load Function . . . 104
10.1.2 Homepage Component . . . 105
10.2 10.2 Content Listing (Tracks, Lessons) . . . 107
10.2.1 Tracks Listing . . . 107
10.2.2 Lessons Listing with Search . . . 108
10.3 10.3 Detail Pages . . . 109
10.3.1 Single Track Page . . . 109
10.3.2 Single Lesson Page . . . 110
10.4 10.4 Search Functionality . . . 110
10.5 10.5 User Dashboard . . . 111
10.6 10.6 Error and 404 Pages . . . 112
10.7 Summary . . . 113

11 Reusable Components 114

11.1 11.1 Navbar with Auth State . . . 114
11.2 11.2 Footer Component . . . 116

11.3 11.3 Card Components . . . 117
11.3.1 LessonCard . . . 117
11.3.2 TrackCard . . . 118
11.4 11.4 Modal System . . . 119
11.4.1 Usage . . . 119
11.5 11.5 Toast Notifications . . . 120
11.5.1 Usage with Store . . . 121
11.6 11.6 Loading States . . . 121
11.6.1 Skeleton Loading . . . 122
11.6.2 Page Loading . . . 122
11.7 Summary . . . 122

12 Video Player Integration 124

12.1 12.1 YouTube IFrame API . . . 124
12.1.1 Loading the API . . . 124
12.1.2 API Reference . . . 124
12.2 12.2 Building VideoPlayer Component . . . 125
12.3 12.3 Progress Tracking . . . 127
12.3.1 The 10-Minute Rule . . . 127
12.3.2 Reporting Progress . . . 127
12.4 12.4 Handling Events . . . 127
12.5 12.5-12.7 Advanced Features . . . 128
12.5.1 Quality Selection (handled by YouTube) . . . 128
12.5.2 Fullscreen Support . . . 128

13 Interactive Quiz System 129

13.1 13.1 Quiz Data Structure . . . 129
13.1.1 Storing in Database . . . 129

13.2 13.2-13.7 Quiz Component 129

14 Progress Tracking **131**

14.1 14.1-14.6 Progress System 131

14.1.1 Database Schema 131

14.1.2 UPSERT Logic 131

15 Global Search **132**

15.1 15.1-15.6 Search Implementation 132

15.1.1 SQLite LIKE Search 132

15.1.2 Search Page 132

16 Part V: Admin Panel **133**

16.1 Chapter 16: Admin Dashboard 133

16.1.1 16.1 Access Control 133

16.1.2 16.2-16.5 Dashboard Implementation 133

16.2 Chapter 17: Content Management 134

16.2.1 17.1-17.6 CRUD Operations 134

16.3 Chapter 18: Audit & Protection 135

16.3.1 18.1-18.5 Audit Trail 135

17 Part VI: Content Pipeline **136**

17.1 Chapter 19: Content Seeding 136

17.1.1 19.1-19.6 Seed Script 136

17.2 Chapter 20: Media Management 136

17.2.1 20.1 Hybrid Video Strategy 137

17.2.2 20.2 Cloudflare R2 Setup 137

17.2.3 20.3 File Structure 137

17.2.4 20.4 Database Integration 137

17.2.5 20.5 No SDK Needed! . . . 137
17.2.6 20.6 Upload via rclone . . . 138
17.2.7 20.7 Future: Admin Upload (مؤجل) . . . 138

18 Part VII: Testing & Quality 139

18.1 Chapter 21: Testing Strategy . . . 139
18.1.1 Unit Tests with Bun . . . 139
18.1.2 Run Tests . . . 139
18.2 Chapter 22: Code Quality . . . 139
18.2.1 ESLint + Prettier . . . 139

19 Part VIII: Deployment 141

19.1 Chapter 23-26: Production Deployment . . . 141
19.1.1 Build . . . 141
19.1.2 VPS Deployment . . . 141
19.1.3 Caddy Config . . . 141

20 Part IX: Scaling & Optimization 142

20.1 Chapter 27-28: Performance . . . 142
20.1.1 Database Optimization . . . 142
20.1.2 Caching . . . 142

21 Part X: Appendices 143

21.1 Appendix A: Code Reference . . . 143
21.2 Appendix B: Commands . . . 143
21.3 Appendix C: Troubleshooting . . . 143
21.4 Appendix D: Resources . . . 143

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.

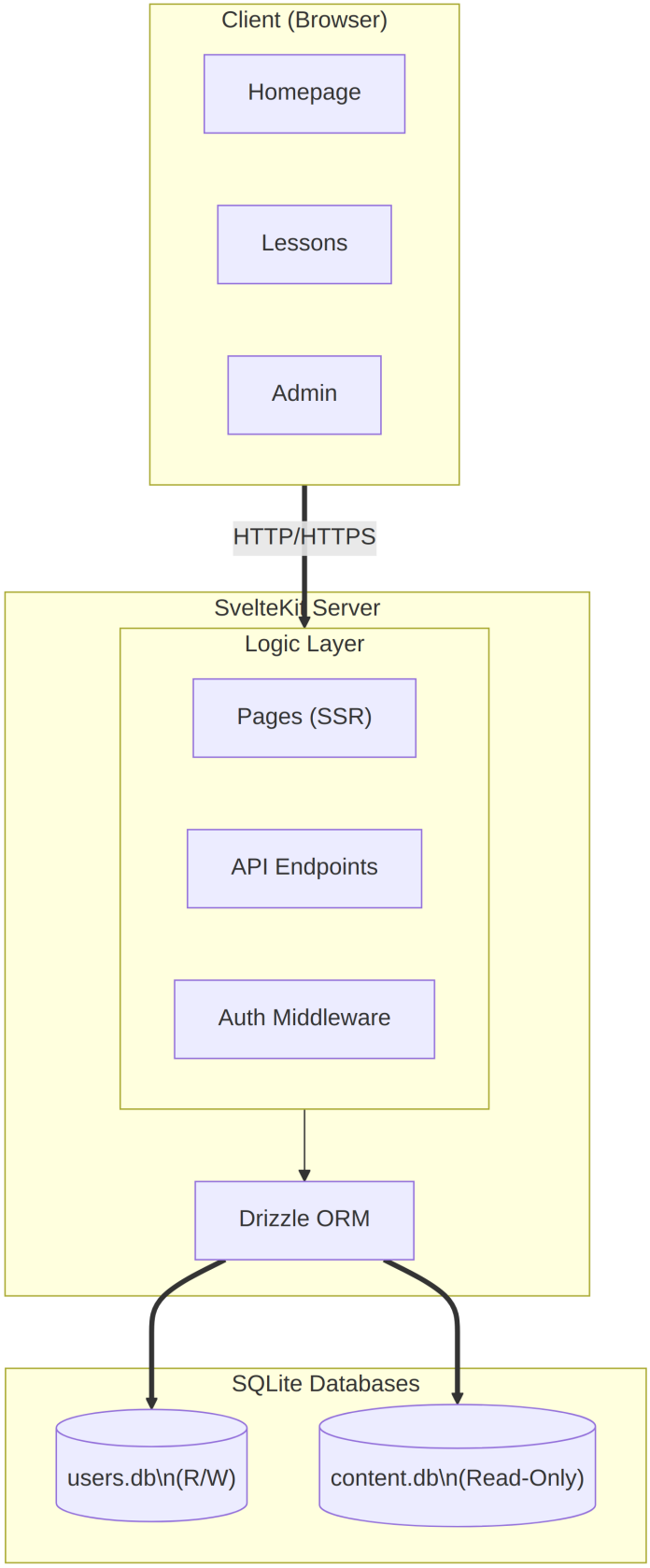

Figure 1.1: High-Level Architecture

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:

```
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

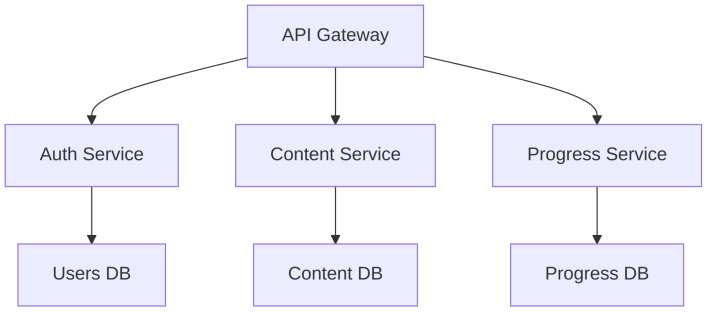

Figure 2.1: Microservices Architecture

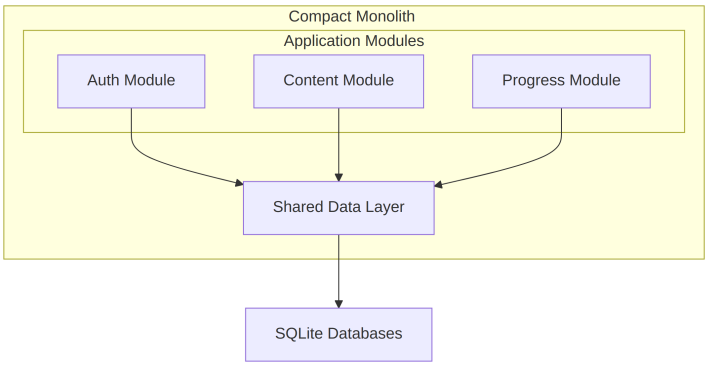

Figure 2.2: Compact Monolith Architecture

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.

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**.

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

```
data/├──
 users.db      # User-generated data|
   ├── users          # Account information|
   ├── sessions       # Login sessions|
   └── lesson_progress # Watch history, quiz scores|└──

 content.db    # Educational content
    ├── tracks         # Learning paths
    ├── series         # Course series
    ├── lessons        # Individual lessons
    └── 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

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

import { drizzle } from 'drizzle-orm/bun-sqlite';
import { Database } from 'bun:sqlite';

// User database (read-write)
const usersSqlite = new Database('data/users.db');
usersSqlite.exec('PRAGMA journal_mode = WAL');
usersSqlite.exec('PRAGMA foreign_keys = ON');

export const db = drizzle(usersSqlite);

// Content database (read-mostly)
const contentSqlite = new Database('data/content.db');
contentSqlite.exec('PRAGMA foreign_keys = ON');

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:

```
-- Who are our users?
CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    email TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    name TEXT NOT NULL,
    role TEXT DEFAULT 'user',  -- user, editor, admin
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

-- Active login sessions
CREATE TABLE sessions (
    id TEXT PRIMARY KEY,  -- UUID
    user_id INTEGER REFERENCES users(id),
    expires_at TEXT NOT NULL
);

-- Learning progress
CREATE TABLE lesson_progress (
    id INTEGER PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    lesson_id TEXT NOT NULL,  -- References content.db
    watched_seconds INTEGER DEFAULT 0,
```

```
    video_completed INTEGER DEFAULT 0,
    quiz_passed INTEGER DEFAULT 0,
    quiz_score INTEGER DEFAULT 0
);
```

2.4.2 content.db — Educational Content

All the learning material:

```
-- Learning tracks (e.g., "Islamic Creed", "Hadith Studies")
CREATE TABLE tracks (
    id TEXT PRIMARY KEY,  -- e.g., "aqeedah", "hadith"
    title TEXT NOT NULL,
    description TEXT,
    icon TEXT,
    "order" INTEGER DEFAULT 0
);

-- Series within tracks
CREATE TABLE series (
    id TEXT PRIMARY KEY,
    track_id TEXT REFERENCES tracks(id),
    title TEXT NOT NULL,
    instructor TEXT,
    is_locked INTEGER DEFAULT 0
);

-- Individual lessons
CREATE TABLE lessons (
    id TEXT PRIMARY KEY,
    track_id TEXT REFERENCES tracks(id),
    series_id TEXT REFERENCES series(id),
    title TEXT NOT NULL,
    slug TEXT UNIQUE NOT NULL,
    is_published INTEGER DEFAULT 0
);

-- Video segments within lessons
CREATE TABLE videos (
    id TEXT PRIMARY KEY,
    lesson_id TEXT REFERENCES lessons(id),
    title TEXT,
    youtube_id TEXT,
    duration INTEGER,
    quiz TEXT  -- JSON array of questions
);
```

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

```
User → CDN → Server
         ↓
    Cached Content
```

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

```
Password Flow:
User Input → bcrypt hash → Store hash
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

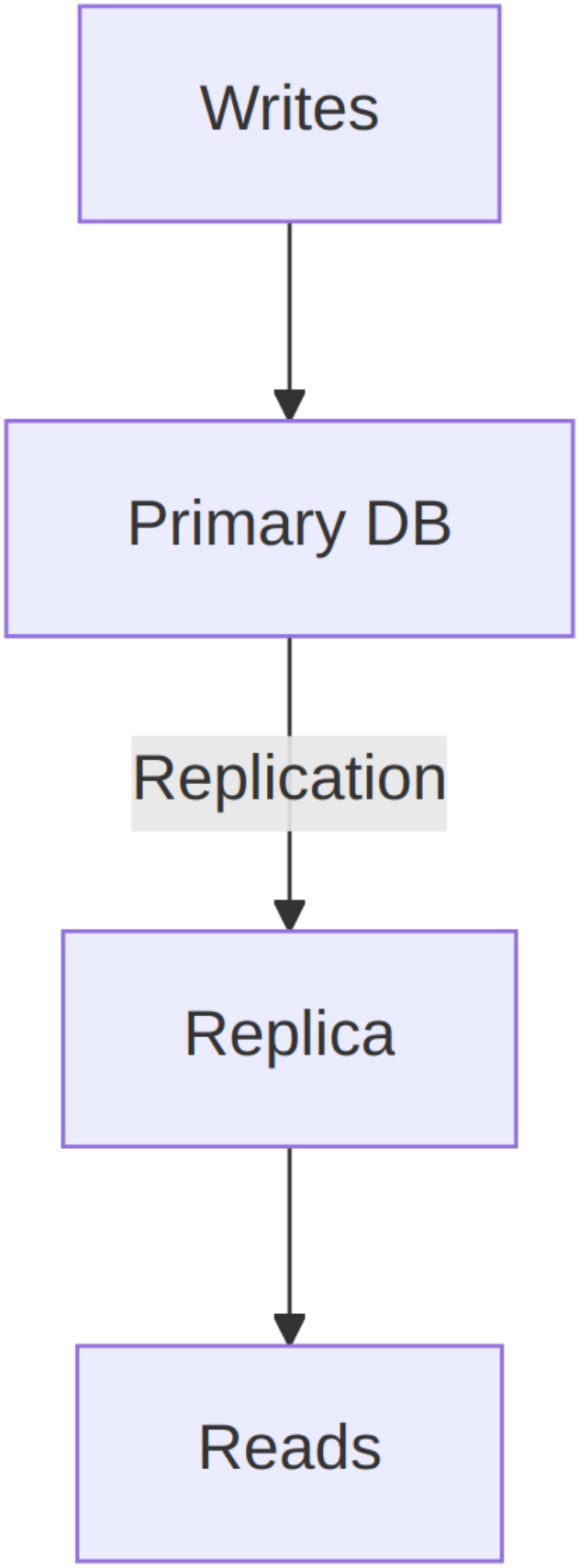

Figure 2.3: Read Replicas

2.6.2 Authorization Layers

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

2.6.3 Content Protection

```
// Audit trail on every content table
{
    created_at: TEXT,       // When created
    updated_at: TEXT,       // When modified
    last_modified_by: TEXT, // Who modified
    is_locked: INTEGER      // Prevent changes
}
```

2.6.4 Rate Limiting (Placeholder)

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

2.6.5 SQL Injection Prevention

Drizzle ORM uses parameterized queries automatically:

```
// Safe — Drizzle handles escaping
db.select()
  .from(users)
  .where(eq(users.email, userInput));

// NEVER do this
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:

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

Windows (via WSL):

```
# First, install WSL if you haven't
wsl --install

# Then install Bun in WSL
curl -fsSL https://bun.sh/install | bash
```

Verify installation:

```
bun --version
# 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:

```
{
    // Editor basics
    "editor.tabSize": 4,
    "editor.formatOnSave": true,
    "editor.defaultFormatter": "esbenp.prettier-vscode",

    // TypeScript
    "typescript.preferences.importModuleSpecifier": "relative",
    "typescript.suggest.autoImports": true,

    // Svelte
    "svelte.enable-ts-plugin": true,

    // File associations
    "files.associations": {
```

```
        "*.svelte": "svelte"
    },

    // Exclude from search
    "search.exclude": {
        "**/node_modules": true,
        "**/dist": true,
        "**/.svelte-kit": true
    }
}
```

3.3 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

```
# Install all at once
code --install-extension svelte.svelte-vscode
code --install-extension bradlc.vscode-tailwindcss
code --install-extension esbenp.prettier-vscode
code --install-extension dbaeumer.vscode-eslint
code --install-extension usernamehw.errorlens
code --install-extension eamodio.gitlens
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

```
# Set your identity
git config --global user.name "Your Name"
git config --global user.email "your@email.com"

# Better defaults
git config --global init.defaultBranch main
git config --global pull.rebase true
git config --global core.autocrlf input  # Linux/Mac
```

3.4.2 SSH Key Setup (for GitHub)

```
# Generate SSH key
ssh-keygen -t ed25519 -C "your@email.com"

# Start SSH agent
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_ed25519

# Copy public key
cat ~/.ssh/id_ed25519.pub
# Paste this in GitHub → Settings → SSH Keys
```

3.4.3 Project .gitignore

Create `.gitignore`:

```
# Dependencies
node_modules/
.pnpm-store/
```

```
# Build outputs
.svelte-kit/
build/
dist/

# Environment
.env
.env.local
.env.*.local

# Databases (optional: you might want to track content.db)
data/*.db
data/*.db-wal
data/*.db-shm

# Editor
.vscode/*
!.vscode/settings.json
!.vscode/extensions.json
.idea/
*.swp
*.swo

# OS
.DS_Store
Thumbs.db

# Logs
*.log
npm-debug.log*
```

3.5 3.5 Project Initialization

Let's create our project from scratch.

3.5.1 Create Project Directory

```
mkdir afidna
cd afidna
```

3.5.2 Initialize with Bun + SvelteKit

```
# Create SvelteKit project
bun create svelte@latest .

# When prompted, choose:
# - Skeleton project
# - TypeScript
```

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

3.5.3 Install Dependencies

```
# Core dependencies (Bun Native - no external SQLite/bcrypt needed!)
bun add drizzle-orm

# Development dependencies
bun add -d drizzle-kit @types/bun

# UI dependencies
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`:

```
import adapter from '@sveltejs/adapter-node';
import { vitePreprocess } from '@sveltejs/vite-plugin-svelte';

/** @type {import('@sveltejs/kit').Config} */
const config = {
    preprocess: vitePreprocess(),
    kit: {
        adapter: adapter(),
        alias: {
            $components: 'src/lib/components',
            $server: 'src/lib/server'
        }
    }
};

export default config;
```

3.5.5 Tailwind Setup

Create `src/app.css`:

```
@import 'tailwindcss';
@plugin 'daisyui';

/* RTL Support */
[dir="rtl"] {
    text-align: right;
```

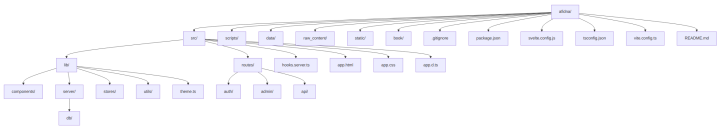

Figure 3.1: Project File Structure

```
}

/* Custom utilities */
@layer utilities {
    .card-hover {
        @apply transition-transform hover:scale-[1.02];
    }
}
```

Update `vite.config.ts`:

```
import { sveltekit } from '@sveltejs/kit/vite';
import tailwindcss from '@tailwindcss/vite';
import { defineConfig } from 'vite';

export default defineConfig({
    plugins: [
        tailwindcss(),
        sveltekit()
    ],
    server: {
        fs: {
            allow: ['.']  // حامسلل ةءارقب تافلم .db
        }
    },
    optimizeDeps: {
        exclude: ['bun:sqlite']  // عنم Vite نم ةلواحم مزح ةبتكملا ةيلصألا
    },
    build: {
        rollupOptions: {
            external: ['bun:sqlite']  // سفن ءيشلا دنع ءانبلا
        }
    }
});
```

3.6 3.6 Directory Structure Best Practices

Let's organize our project for maintainability.

3.6.1 The Complete Structure

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

```
# Create all directories
mkdir -p src/lib/{components,server/db,stores,utils}
mkdir -p src/routes/{tracks,lessons,search,auth/{login,register,logout}}
mkdir -p src/routes/admin/{lessons,videos,series}
mkdir -p src/routes/api/progress
mkdir -p scripts
mkdir -p data
mkdir -p raw_content
mkdir -p static/images
```

3.7 Verification Checklist

Before moving on, verify your setup:

```
# Check Bun
bun --version    # Should be 1.x.x

# Check project runs
bun run dev      # Should start on localhost:5173

# Check TypeScript
bun run check    # Should pass

# Check Git
git status       # Should show your project
```

3.7.1 Expected Output

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

```
  VITE v5.x.x  ready in xxx ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  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

```
# Start development server
bun run dev

# Type checking
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 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)

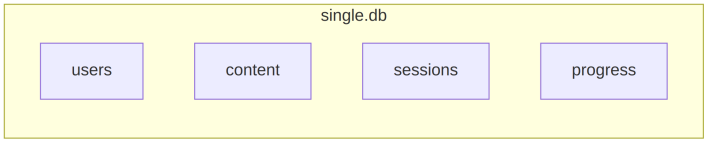

Figure 4.1: Traditional Single Database

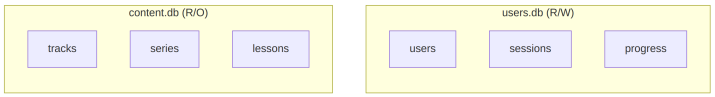

Figure 4.2: Multiple Databases Strategy

- 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)

```
-- Instead of this:
CREATE TABLE lessons (
    id INTEGER PRIMARY KEY  -- What is lesson 42?
);

-- Do this:
CREATE TABLE lessons (
    id TEXT PRIMARY KEY     -- lesson "hadith-jibril" is clear
);
```

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)

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

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

4.3.3 Principle 3: Store Timestamps as TEXT

```
-- SQLite has no native date type
created_at TEXT DEFAULT CURRENT_TIMESTAMP
-- 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

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

4.3.5 Principle 5: Store JSON as TEXT

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

Parse in application code:

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

4.4 4.4 Hierarchical Data Modeling

Our content has a natural hierarchy:

4.4.1 The Schema

```
-- Level 1: Tracks (Learning Paths)
CREATE TABLE tracks (
    id TEXT PRIMARY KEY,          -- "hadith"
    title TEXT NOT NULL,          -- "Hadith Studies"
    description TEXT,
    icon TEXT,                    -- ""
    "order" INTEGER DEFAULT 0
);

-- Level 2: Series (Course Collections)
CREATE TABLE series (
    id TEXT PRIMARY KEY,          -- "nawawi-40"
    track_id TEXT REFERENCES tracks(id),
    title TEXT NOT NULL,          -- "40 Nawawi Hadith"
    description TEXT,
    instructor TEXT,              -- "Sheikh X"
    thumbnail_url TEXT,
    "order" INTEGER DEFAULT 0
);

-- Level 3: Lessons (Individual Classes)
CREATE TABLE lessons (
    id TEXT PRIMARY KEY,          -- "hadith-jibril"
```

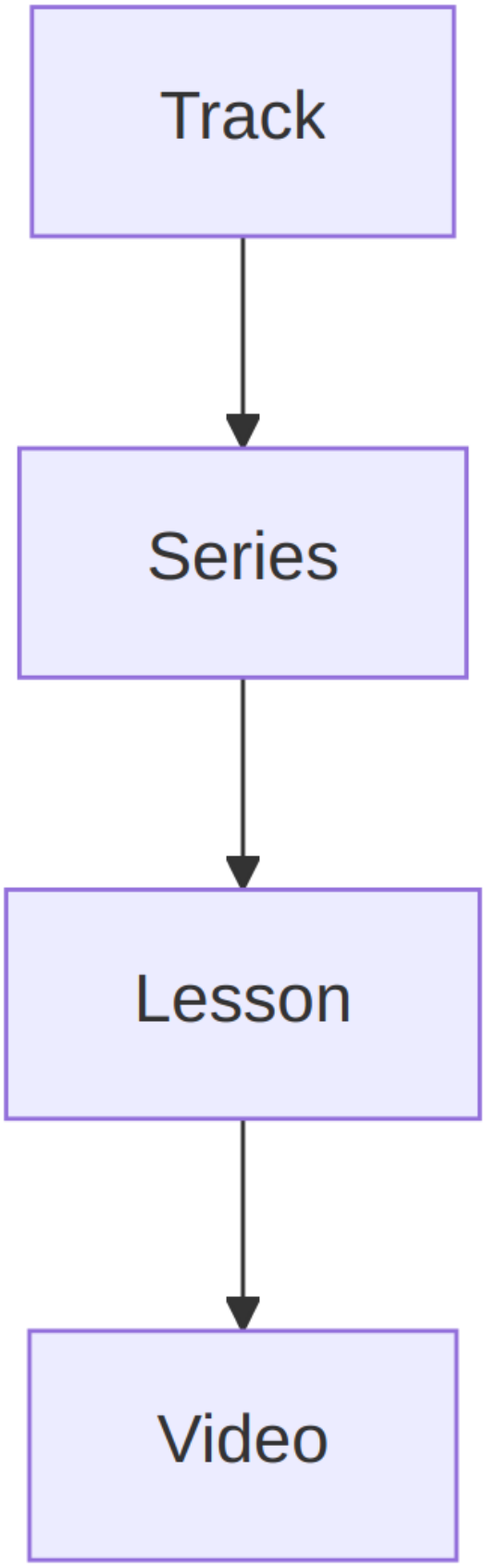

Figure 4.3: Content Hierarchy

```
    track_id TEXT REFERENCES tracks(id),
    series_id TEXT REFERENCES series(id),
    title TEXT NOT NULL,
    slug TEXT UNIQUE NOT NULL,     -- URL-friendly
    description TEXT,
    instructor TEXT,
    pdf_url TEXT,
    is_published INTEGER DEFAULT 0,
    "order" INTEGER DEFAULT 0
);

-- Level 4: Videos (Lesson Segments)
CREATE TABLE videos (
    id TEXT PRIMARY KEY,           -- "hadith-jibril-v1"
    lesson_id TEXT REFERENCES lessons(id),
    title TEXT,
    youtube_id TEXT,               -- YouTube video ID
    download_url TEXT,             -- Direct download link
    duration INTEGER,              -- Seconds
    summary TEXT,
    quiz TEXT,                     -- JSON array
    "order" INTEGER DEFAULT 0
);
```

4.4.2 Querying the Hierarchy

Get full lesson with related data:

```
// Get lesson with track info
const lesson = db
    .select()
    .from(lessons)
    .leftJoin(tracks, eq(lessons.trackId, tracks.id))
    .where(eq(lessons.slug, 'hadith-jibril'))
    .get();

// Get videos for lesson
const videos = db
    .select()
    .from(videos)
    .where(eq(videos.lessonId, lesson.id))
    .orderBy(videos.order)
    .all();
```

4.5 4.5 Audit Trail Implementation

Track who changed what and when.

4.5.1 The Fields

Add these to every content table:

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

4.5.2 Complete Table Definition

```
CREATE TABLE lessons (
    -- Core fields
    id TEXT PRIMARY KEY,
    track_id TEXT,
    series_id TEXT,
    title TEXT NOT NULL,
    slug TEXT UNIQUE NOT NULL,
    description TEXT,
    instructor TEXT,
    pdf_url TEXT,
    is_published INTEGER DEFAULT 0,
    "order" INTEGER DEFAULT 0,

    -- Audit fields
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    updated_at TEXT,
    last_modified_by TEXT DEFAULT 'script',
    is_locked INTEGER DEFAULT 0,

    -- Foreign keys
    FOREIGN KEY (track_id) REFERENCES tracks(id),
    FOREIGN KEY (series_id) REFERENCES series(id)
);
```

4.5.3 Using the Audit Trail

When updating via script:

```
db.update(lessons)
    .set({
        title: newTitle,
        updated_at: new Date().toISOString(),
        last_modified_by: 'script'
    })
    .where(eq(lessons.id, lessonId))
    .run();
```

When updating via admin:

```
db.update(lessons)
    .set({
        title: newTitle,
        updated_at: new Date().toISOString(),
```

```
        last_modified_by: userId.toString()  // Human user
    })
    .where(eq(lessons.id, lessonId))
    .run();
```

Protecting human edits:

```
function shouldUpdate(existing) {
    // Don't overwrite if locked
    if (existing.isLocked) return false;

    // Don't overwrite human edits (unless forced)
    if (existing.lastModifiedBy !== 'script') {
        return FORCE_MODE;
    }

    return true;
}
```

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

```
-- users.db
CREATE INDEX idx_sessions_user ON sessions(user_id);
CREATE INDEX idx_progress_user ON lesson_progress(user_id);
CREATE INDEX idx_progress_lesson ON lesson_progress(lesson_id);

-- content.db
CREATE INDEX idx_lessons_track ON lessons(track_id);
CREATE INDEX idx_lessons_series ON lessons(series_id);
CREATE INDEX idx_lessons_slug ON lessons(slug);
```

```
CREATE INDEX idx_videos_lesson ON videos(lesson_id);
CREATE INDEX idx_series_track ON series(track_id);
```

4.6.3 Composite Indexes

For queries with multiple conditions:

```
-- For: WHERE user_id = ? AND lesson_id = ?
CREATE UNIQUE INDEX idx_progress_user_lesson
    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

```
const db = new Database('data/users.db');
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

```
// Apply these at connection time
db.exec('PRAGMA journal_mode = WAL');       // Enable WAL
db.exec('PRAGMA busy_timeout = 5000');      // Wait 5s if busy
db.exec('PRAGMA synchronous = NORMAL');     // Balance safety/speed
db.exec('PRAGMA foreign_keys = ON');        // Enforce FK constraints
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:

```
// Drizzle - feels like SQL
db.select()
  .from(users)
  .where(eq(users.email, email))
  .limit(1);

// Actual SQL it generates
// SELECT * FROM users WHERE email = ? LIMIT 1
```

5.1.3 Installation

```
bun add drizzle-orm
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`:

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

// ============================================
// USERS DATABASE SCHEMA (users.db)
// ============================================

// Users table
export const users = sqliteTable('users', {
    id: integer('id').primaryKey({ autoIncrement: true }),
    email: text('email').notNull().unique(),
    passwordHash: text('password_hash').notNull(),
    name: text('name').notNull(),
    role: text('role').default('user').notNull(), // user | editor | admin
    avatar: text('avatar'),
    isActive: integer('is_active', { mode: 'boolean' }).default(true).notNull(),
    createdAt: text('created_at').default('CURRENT_TIMESTAMP').notNull(),
    updatedAt: text('updated_at'),
});

// Sessions table
export const sessions = sqliteTable('sessions', {
    id: text('id').primaryKey(), // UUID
    userId: integer('user_id')
        .notNull()
        .references(() => users.id, { onDelete: 'cascade' }),
    expiresAt: text('expires_at').notNull(),
    createdAt: text('created_at').default('CURRENT_TIMESTAMP').notNull(),
});

// Lesson Progress table
export const lessonProgress = sqliteTable('lesson_progress', {
    id: integer('id').primaryKey({ autoIncrement: true }),
    userId: integer('user_id')
        .notNull()
        .references(() => users.id, { onDelete: 'cascade' }),
    lessonId: text('lesson_id').notNull(),
    watchedSeconds: integer('watched_seconds').default(0).notNull(),
    videoCompleted: integer('video_completed', { mode: 'boolean' }).default(false).notNull(),
    quizPassed: integer('quiz_passed', { mode: 'boolean' }).default(false).notNull(),
    quizScore: integer('quiz_score').default(0),
```

```
    completedAt: text('completed_at'),
    updatedAt: text('updated_at').default('CURRENT_TIMESTAMP').notNull(),
});

// Type exports
export type User = typeof users.$inferSelect;
export type NewUser = typeof users.$inferInsert;
export type Session = typeof sessions.$inferSelect;
export type LessonProgress = typeof lessonProgress.$inferSelect;
```

5.2.2 content.db Schema

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

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

// ============================================
// CONTENT DATABASE SCHEMA (content.db)
// ============================================

// Tracks (Learning Paths)
export const tracks = sqliteTable('tracks', {
    id: text('id').primaryKey(),
    title: text('title').notNull(),
    description: text('description'),
    icon: text('icon'),
    order: integer('order').default(0).notNull(),
});

// Series (Course Collections)
export const series = sqliteTable('series', {
    id: text('id').primaryKey(),
    trackId: text('track_id').references(() => tracks.id),
    title: text('title').notNull(),
    description: text('description'),
    instructor: text('instructor'),
    thumbnailUrl: text('thumbnail_url'),
    order: integer('order').default(0).notNull(),
    // Audit fields
    createdAt: text('created_at').default('CURRENT_TIMESTAMP'),
    updatedAt: text('updated_at'),
    lastModifiedBy: text('last_modified_by').default('script'),
    isLocked: integer('is_locked', { mode: 'boolean' }).default(false),
});

// Lessons
export const lessons = sqliteTable('lessons', {
    id: text('id').primaryKey(),
    trackId: text('track_id').references(() => tracks.id),
    seriesId: text('series_id').references(() => series.id),
    title: text('title').notNull(),
    slug: text('slug').notNull().unique(),
    description: text('description'),
    instructor: text('instructor'),
    thumbnailUrl: text('thumbnail_url'),
    pdfUrl: text('pdf_url'),
    order: integer('order').default(0).notNull(),
    isPublished: integer('is_published', { mode: 'boolean' }).default(false).notNull(),
    // Audit fields
    createdAt: text('created_at').default('CURRENT_TIMESTAMP'),
```

```
    updatedAt: text('updated_at'),
    lastModifiedBy: text('last_modified_by').default('script'),
    isLocked: integer('is_locked', { mode: 'boolean' }).default(false),
});

// Videos
export const videos = sqliteTable('videos', {
    id: text('id').primaryKey(),
    lessonId: text('lesson_id')
        .notNull()
        .references(() => lessons.id, { onDelete: 'cascade' }),
    title: text('title'),
    youtubeId: text('youtube_id'),
    downloadUrl: text('download_url'),
    duration: integer('duration'),
    summary: text('summary'),
    rawTranscript: text('raw_transcript'),
    formattedTranscript: text('formatted_transcript'),
    quiz: text('quiz'), // JSON
    order: integer('order').default(0).notNull(),
    // Audit fields
    createdAt: text('created_at').default('CURRENT_TIMESTAMP'),
    updatedAt: text('updated_at'),
    lastModifiedBy: text('last_modified_by').default('script'),
    isLocked: integer('is_locked', { mode: 'boolean' }).default(false),
});

// Type exports
export type Track = typeof tracks.$inferSelect;
export type Series = typeof series.$inferSelect;
export type Lesson = typeof lessons.$inferSelect;
export type Video = typeof videos.$inferSelect;
```

5.3 5.3 Type-Safe Queries

Drizzle provides complete type safety for all queries.

5.3.1 Basic Select

```
import { db } from '$lib/server/db';
import { users } from '$lib/server/db/schema';
import { eq } from 'drizzle-orm';

// Get user by email
const user = db
    .select()
    .from(users)
    .where(eq(users.email, 'test@example.com'))
    .get();

// TypeScript knows: user is User | undefined
```

5.3.2 Select Specific Columns

```
// Only get what you need
const userNames = db
    .select({
        id: users.id,
        name: users.name,
    })
    .from(users)
    .all();

// Type: { id: number; name: string }[]
```

5.3.3 Insert

```
// Insert returns the inserted row
const newUser = db
    .insert(users)
    .values({
        email: 'new@example.com',
        passwordHash: hashedPassword,
        name: 'New User',
    })
    .returning()
    .get();

// newUser is User
```

5.3.4 Update

```
// Update with conditions
db.update(users)
    .set({
        name: 'Updated Name',
        updatedAt: new Date().toISOString(),
    })
    .where(eq(users.id, userId))
    .run();
```

5.3.5 Delete

```
// Delete with conditions
db.delete(sessions)
    .where(eq(sessions.id, sessionId))
    .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`):

```
import { Database } from 'bun:sqlite';
import { resolve } from 'path';

const dataDir = resolve(import.meta.dir, '../data');

console.log('  Creating users.db...');

const db = new Database(resolve(dataDir, 'users.db'), { create: true });

db.exec('PRAGMA journal_mode = WAL');
db.exec('PRAGMA foreign_keys = ON');

db.exec(`
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        email TEXT NOT NULL UNIQUE,
        password_hash TEXT NOT NULL,
        name TEXT NOT NULL,
        role TEXT DEFAULT 'user' NOT NULL,
        avatar TEXT,
        is_active INTEGER DEFAULT 1 NOT NULL,
        created_at TEXT DEFAULT CURRENT_TIMESTAMP NOT NULL,
        updated_at TEXT
    );

    CREATE TABLE IF NOT EXISTS sessions (
        id TEXT PRIMARY KEY,
        user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
        expires_at TEXT NOT NULL,
        created_at TEXT DEFAULT CURRENT_TIMESTAMP NOT NULL
    );

    CREATE TABLE IF NOT EXISTS lesson_progress (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
        user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
        lesson_id TEXT NOT NULL,
        watched_seconds INTEGER DEFAULT 0 NOT NULL,
        video_completed INTEGER DEFAULT 0 NOT NULL,
        quiz_passed INTEGER DEFAULT 0 NOT NULL,
        quiz_score INTEGER DEFAULT 0,
        completed_at TEXT,
        updated_at TEXT DEFAULT CURRENT_TIMESTAMP NOT NULL,
        UNIQUE(user_id, lesson_id)
    );

    CREATE INDEX IF NOT EXISTS idx_sessions_user ON sessions(user_id);
    CREATE INDEX IF NOT EXISTS idx_progress_user ON lesson_progress(user_id);
`);

console.log('✅ Done!');
db.close();
```

Run with:

```
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 5.5 Connecting Multiple Databases

Here's how we connect both databases.

5.5.1 Connection Module

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

```
import { drizzle } from 'drizzle-orm/bun-sqlite';
import { Database } from 'bun:sqlite';
import { resolve } from 'path';

// Import schemas
import * as userSchema from './schema';
```

```
import * as contentSchema from './schema-content';

// Resolve paths
const dataDir = resolve(process.cwd(), 'data');

// ============================================
// USER DATABASE (Read-Write)
// ============================================

const usersSqlite = new Database(resolve(dataDir, 'users.db'));
usersSqlite.exec('PRAGMA journal_mode = WAL');
usersSqlite.exec('PRAGMA busy_timeout = 5000');
usersSqlite.exec('PRAGMA synchronous = NORMAL');
usersSqlite.exec('PRAGMA foreign_keys = ON');

export const db = drizzle(usersSqlite, { schema: userSchema });

// ============================================
// CONTENT DATABASE (Read-Mostly)
// ============================================

const contentSqlite = new Database(resolve(dataDir, 'content.db'));
contentSqlite.exec('PRAGMA foreign_keys = ON');

export const contentDatabase = drizzle(contentSqlite, { schema: contentSchema });

// ============================================
// RE-EXPORTS
// ============================================

// Re-export schemas for convenience
export * from './schema';
export * from './schema-content';
```

5.5.2 Usage in SvelteKit

```
// In +page.server.ts
import { db, contentDatabase, users, lessons } from '$lib/server/db';

export async function load() {
    // Query user database
    const allUsers = db.select().from(users).all();

    // Query content database
    const allLessons = contentDatabase.select().from(lessons).all();

    return { users: allUsers, lessons: allLessons };
}
```

5.6 5.6 Query Optimization

Write efficient queries from the start.

5.6.1 Use Specific Selects

```
// ✗ Bad: Fetches all columns
const lessons = contentDatabase.select().from(lessons).all();

// ✓ Good: Only fetches needed columns
const lessons = contentDatabase
    .select({
        id: lessons.id,
        title: lessons.title,
        slug: lessons.slug,
    })
    .from(lessons)
    .all();
```

5.6.2 Use Limits

```
// Always limit when you don't need all results
const recent = contentDatabase
    .select()
    .from(lessons)
    .orderBy(desc(lessons.createdAt))
    .limit(10)
    .all();
```

5.6.3 Use Joins Instead of N+1

```
// ✗ Bad: N+1 queries
const lessons = contentDatabase.select().from(lessons).all();
for (const lesson of lessons) {
    const track = contentDatabase
        .select()
        .from(tracks)
        .where(eq(tracks.id, lesson.trackId))
        .get();
}

// ✓ Good: Single join query
const lessonsWithTrack = contentDatabase
    .select()
    .from(lessons)
    .leftJoin(tracks, eq(lessons.trackId, tracks.id))
    .all();
```

5.6.4 Count Queries

```
import { count } from 'drizzle-orm';

// Get count efficiently
const result = contentDatabase
    .select({ count: count() })
    .from(lessons)
```

```
    .where(eq(lessons.isPublished, true))
    .get();

const total = result?.count ?? 0;
```

5.6.5 Aggregations

```
import { count, sql } from 'drizzle-orm';

// Count lessons per track
const stats = contentDatabase
    .select({
        trackId: lessons.trackId,
        lessonCount: count(),
    })
    .from(lessons)
    .groupBy(lessons.trackId)
    .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)

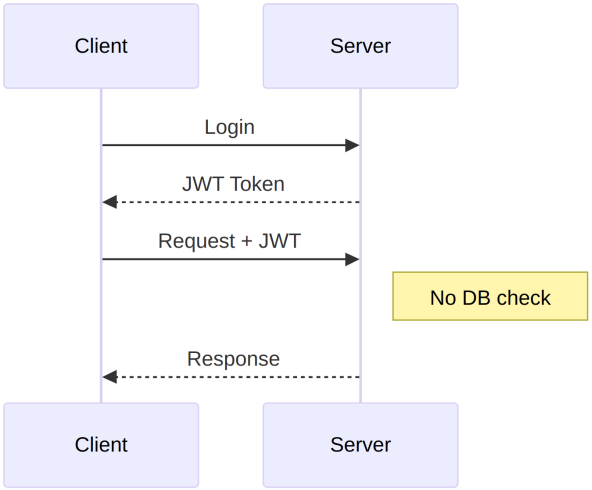

Figure 6.1: JWT Authentication Flow

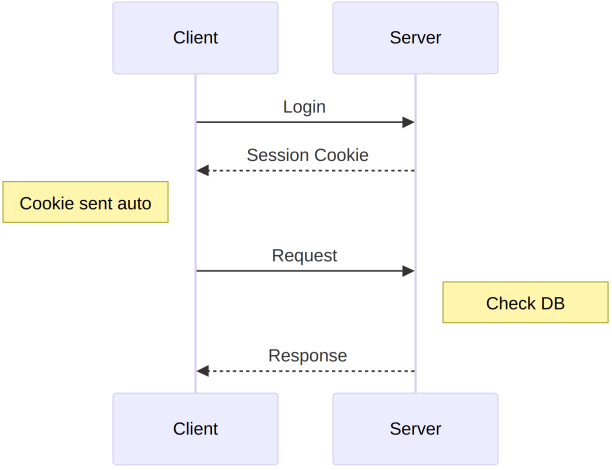

Figure 6.2: Session-Based Authentication Flow

6.2 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

```
// src/lib/server/auth.ts

export async function hashPassword(password: string): Promise<string> {
    return await Bun.password.hash(password, {
        algorithm: 'argon2id',
        memoryCost: 4,
        timeCost: 3,
    });
}

export async function verifyPassword(
    password: string,
    hash: string
): Promise<boolean> {
    return await Bun.password.verify(password, hash);
}
```

6.2.3 Usage

```
// Registration
const passwordHash = await hashPassword(userPassword);
db.insert(users).values({
    email,
    passwordHash,
    name,
}).run();

```

```
// Login
const user = db.select().from(users)
    .where(eq(users.email, email))
    .get();

if (user && await verifyPassword(password, user.passwordHash)) {
    // Login successful
} else {
    // Invalid credentials
}
```

6.3 6.3 Session Management

Sessions link browser requests to user accounts.

6.3.1 Session Flow

```
1. User logs in with email/password
2. Server:
   - Validates credentials
   - Creates session in database
   - Sets cookie with session ID

3. User makes request
4. Server:
   - Reads session ID from cookie
   - Looks up session in database
   - Attaches user to request

5. User logs out
6. Server:
   - Deletes session from database
   - Clears cookie
```

6.3.2 Session Schema

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

6.3.3 Session Functions

```
// src/lib/server/auth.ts
import { db, sessions, users } from '$lib/server/db';
import { eq, and, gt } from 'drizzle-orm';
import { randomUUID } from 'crypto';

const SESSION_DURATION_DAYS = 7;
const REMEMBER_ME_DAYS = 30;

export function createSession(
    userId: number,
    rememberMe: boolean = false
): { sessionId: string; expiresAt: Date } {
    const sessionId = randomUUID();
    const days = rememberMe ? REMEMBER_ME_DAYS : SESSION_DURATION_DAYS;
    const expiresAt = new Date();
    expiresAt.setDate(expiresAt.getDate() + days);

    db.insert(sessions).values({
        id: sessionId,
        userId,
        expiresAt: expiresAt.toISOString(),
    }).run();

    return { sessionId, expiresAt };
}

export function validateSession(sessionId: string) {
    const now = new Date().toISOString();

    const result = db
        .select({
            session: sessions,
            user: users,
        })
        .from(sessions)
        .innerJoin(users, eq(sessions.userId, users.id))
        .where(
            and(
                eq(sessions.id, sessionId),
                gt(sessions.expiresAt, now)
            )
        )
        .get();

    if (!result) return null;

    return {
        session: result.session,
        user: result.user,
    };
}

export function deleteSession(sessionId: string) {
    db.delete(sessions).where(eq(sessions.id, sessionId)).run();
}

export function deleteUserSessions(userId: number) {
    db.delete(sessions).where(eq(sessions.userId, userId)).run();
}
```

6.4 6.4 Cookie Security (HttpOnly, SameSite)

Cookies must be configured securely.

6.4.1 Cookie Options

```
import type { Cookies } from '@sveltejs/kit';

const COOKIE_NAME = 'session';

export function setSessionCookie(
    cookies: Cookies,
    sessionId: string,
    expiresAt: Date
) {
    cookies.set(COOKIE_NAME, sessionId, {
        path: '/',
        httpOnly: true,           // JavaScript can't read it
        sameSite: 'lax',          // Prevents CSRF
        secure: process.env.NODE_ENV === 'production',
        expires: expiresAt,
    });
}

export function getSessionCookie(cookies: Cookies): string | undefined {
    return cookies.get(COOKIE_NAME);
}

export function clearSessionCookie(cookies: Cookies) {
    cookies.delete(COOKIE_NAME, { path: '/' });
}
```

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

```
// Login action
export const actions: Actions = {
    default: async ({ request, cookies }) => {
        const form = await request.formData();
        const email = form.get('email') as string;
        const password = form.get('password') as string;
        const rememberMe = form.get('rememberMe') === 'true';

        // Validate user...

        // Create session with extended expiry if rememberMe
        const { sessionId, expiresAt } = createSession(
            user.id,
            rememberMe  // 30 days vs 7 days
        );

        setSessionCookie(cookies, sessionId, expiresAt);

        return redirect(302, '/');
    }
};
```

6.5.2 Login Form

```
<form method="POST">
    <input type="email" name="email" required />
    <input type="password" name="password" required />

    <label>
        <input type="checkbox" name="rememberMe" value="true" />
        Remember me for 30 days
    </label>

    <button type="submit">Login</button>
</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

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

6.6.3 Authorization Helpers

```
// src/lib/server/auth.ts

export function isAdmin(user: User | null): boolean {
    return user?.role === 'admin';
}

export function isEditor(user: User | null): boolean {
    return user?.role === 'editor' || user?.role === 'admin';
}

export function canEditContent(user: User | null): boolean {
    return isEditor(user);
}

export function canLockContent(user: User | null): boolean {
    return isAdmin(user);
}
```

6.6.4 Usage in Load Functions

```
// src/routes/admin/+page.server.ts
import { redirect } from '@sveltejs/kit';
import { isEditor } from '$lib/server/auth';

export async function load({ locals }) {
    if (!locals.user) {
        throw redirect(302, '/auth/login?redirect=/admin');
    }

    if (!isEditor(locals.user)) {
        throw redirect(302, '/');
    }

    // Load admin data...
}
```

6.7 6.7 Protected Routes with Hooks

SvelteKit hooks provide middleware-like functionality.

6.7.1 The Hook

Create `src/hooks.server.ts`:

```
import type { Handle } from '@sveltejs/kit';
import { validateSession, getSessionCookie } from '$lib/server/auth';

export const handle: Handle = async ({ event, resolve }) => {
    // Get session from cookie
    const sessionId = getSessionCookie(event.cookies);

    if (sessionId) {
        const result = validateSession(sessionId);

        if (result) {
            // Attach user to request
            event.locals.user = result.user;
        } else {
            // Invalid/expired session - clear cookie
            event.cookies.delete('session', { path: '/' });
        }
    }

    // Rate limiting placeholder for auth routes
    if (event.url.pathname.startsWith('/api/auth')) {
        // TODO: Implement rate limiting
    }

    return resolve(event);
};
```

6.7.2 Type Definitions

Update `src/app.d.ts`:

```
import type { User } from '$lib/server/db/schema';

declare global {
    namespace App {
        interface Locals {
            user: User | null;
        }
    }
}

export {};
```

6.7.3 Accessing User in Routes

```
// +page.server.ts
export async function load({ locals }) {
    // locals.user is available everywhere!
    const isLoggedIn = !!locals.user;

    return {
        user: locals.user,
        isLoggedIn,
    };
}
```

6.7.4 Layout Data

Share user across all pages:

```
// src/routes/+layout.server.ts
export async function load({ locals }) {
    return {
        user: locals.user,
    };
}
```

```
<!-- src/routes/+layout.svelte -->
<script>
    let { data, children } = $props();
</script>

{#if data.user}
    <p>Logged in as {data.user.name}</p>
{/if}

{@render children()}
```

6.8 Complete Auth Module

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

```
import { randomUUID } from 'crypto';
import { db, users, sessions, type User } from '$lib/server/db';
import { eq, and, gt } from 'drizzle-orm';
import type { Cookies } from '@sveltejs/kit';

// Configuration
const SESSION_DAYS = 7;
const REMEMBER_ME_DAYS = 30;
const COOKIE_NAME = 'session';

// Password hashing with Bun.password (Argon2id)
```

```
export async function hashPassword(password: string): Promise<string> {
    return await Bun.password.hash(password, {
        algorithm: 'argon2id',
        memoryCost: 4,
        timeCost: 3,
    });
}

export async function verifyPassword(password: string, hash: string): Promise<boolean> {
    return await Bun.password.verify(password, hash);
}

// Session management
export function createSession(userId: number, rememberMe = false) {
    const sessionId = randomUUID();
    const days = rememberMe ? REMEMBER_ME_DAYS : SESSION_DAYS;
    const expiresAt = new Date();
    expiresAt.setDate(expiresAt.getDate() + days);

    db.insert(sessions).values({
        id: sessionId,
        userId,
        expiresAt: expiresAt.toISOString(),
    }).run();

    return { sessionId, expiresAt };
}

export function validateSession(sessionId: string) {
    const now = new Date().toISOString();

    return db
        .select({ session: sessions, user: users })
        .from(sessions)
        .innerJoin(users, eq(sessions.userId, users.id))
        .where(and(eq(sessions.id, sessionId), gt(sessions.expiresAt, now)))
        .get();
}

export function deleteSession(sessionId: string) {
    db.delete(sessions).where(eq(sessions.id, sessionId)).run();
}

// Cookie helpers
export function setSessionCookie(cookies: Cookies, sessionId: string, expiresAt: Date) {
    cookies.set(COOKIE_NAME, sessionId, {
        path: '/',
        httpOnly: true,
        sameSite: 'lax',
        secure: process.env.NODE_ENV === 'production',
        expires: expiresAt,
    });
}

export function getSessionCookie(cookies: Cookies) {
    return cookies.get(COOKIE_NAME);
}

export function clearSessionCookie(cookies: Cookies) {
    cookies.delete(COOKIE_NAME, { path: '/' });
}

// Role helpers
export const isAdmin = (user: User | null) => user?.role === 'admin';
```

```
export const isEditor = (user: User | null) =>
    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

```
src/routes/└──
 api/
    ├── progress/
    │   └── +server.ts    # /api/progress
    ├── auth/
    │   ├── login/
    │   │   └── +server.ts    # /api/auth/login
    │   └── register/
    │       └── +server.ts    # /api/auth/register
    └── health/
        └── +server.ts    # /api/health
```

7.1.3 Basic Endpoint Structure

```
// src/routes/api/health/+server.ts
import { json } from '@sveltejs/kit';
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(
```

```
                    eq(lessonProgress.userId, locals.user.id),
                    eq(lessonProgress.lessonId, lessonId)
                )
            )
            .get();

        return json({ progress });
    }

    // Get all progress for user
    const allProgress = db
        .select()
        .from(lessonProgress)
        .where(eq(lessonProgress.userId, locals.user.id))
        .all();

    return json({ progress: allProgress });
};

// POST: Update progress
export const POST: RequestHandler = async ({ request, locals }) => {
    if (!locals.user) {
        return json({ error: 'Unauthorized' }, { status: 401 });
    }

    const body = await request.json();
    const { lessonId, watchedSeconds, videoCompleted, quizPassed, quizScore } = body;

    if (!lessonId) {
        return json({ error: 'lessonId required' }, { status: 400 });
    }

    const now = new Date().toISOString();

    // Check if progress exists
    const existing = db
        .select()
        .from(lessonProgress)
        .where(
            and(
                eq(lessonProgress.userId, locals.user.id),
                eq(lessonProgress.lessonId, lessonId)
            )
        )
        .get();

    if (existing) {
        // Update existing
        const updateData: Record<string, unknown> = { updatedAt: now };

        if (watchedSeconds !== undefined) {
            updateData.watchedSeconds = Math.max(existing.watchedSeconds, watchedSeconds);
        }
        if (videoCompleted !== undefined) {
            updateData.videoCompleted = videoCompleted;
        }
        if (quizPassed !== undefined) {
            updateData.quizPassed = quizPassed;
        }
        if (quizScore !== undefined) {
            updateData.quizScore = quizScore;
        }

        db.update(lessonProgress)
```

```
            .set(updateData)
            .where(eq(lessonProgress.id, existing.id))
            .run();
    } else {
        // Create new
        db.insert(lessonProgress).values({
            userId: locals.user.id,
            lessonId,
            watchedSeconds: watchedSeconds ?? 0,
            videoCompleted: videoCompleted ?? false,
            quizPassed: quizPassed ?? false,
            quizScore: quizScore ?? 0,
            updatedAt: now,
        }).run();
    }

    return json({ success: true });
};
```

7.3 7.3 Request Validation

Never trust client input.

7.3.1 Manual Validation

```
export const POST: RequestHandler = async ({ request }) => {
    const body = await request.json();

    // Validate required fields
    const { email, password, name } = body;

    const errors: string[] = [];

    if (!email || typeof email !== 'string') {
        errors.push('Email is required');
    } else if (!email.includes('@')) {
        errors.push('Invalid email format');
    }

    if (!password || typeof password !== 'string') {
        errors.push('Password is required');
    } else if (password.length < 8) {
        errors.push('Password must be at least 8 characters');
    }

    if (!name || typeof name !== 'string') {
        errors.push('Name is required');
    }

    if (errors.length > 0) {
        return json({ errors }, { status: 400 });
    }
```

```
28
29      // Safe to proceed...
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",
```

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

7.4.2 Error Responses

```
// 400 Bad Request - Client error
return json(
    { error: 'Invalid input', code: 'VALIDATION_ERROR' },
    { status: 400 }
);

// 401 Unauthorized - Not logged in
return json(
    { error: 'Authentication required' },
    { status: 401 }
);

// 403 Forbidden - Logged in but not allowed
return json(
    { error: 'Permission denied' },
    { status: 403 }
);

// 404 Not Found - Resource doesn't exist
return json(
    { error: 'Lesson not found' },
    { status: 404 }
);

// 500 Internal Server Error - Our fault
return json(
    { error: 'Something went wrong' },
    { status: 500 }
);
```

7.4.3 Try-Catch Wrapper

```
export const POST: RequestHandler = async ({ request, locals }) => {
    try {
        // ... handler logic
        return json({ success: true });
    } catch (error) {
        console.error('Progress update error:', error);
        return json(
            { error: 'Failed to update progress' },
            { status: 500 }
        );
    }
};
```

7.5 7.5 Rate Limiting

Prevent abuse with request limits.

7.5.1 Simple In-Memory Rate Limiter

```
// src/lib/server/ratelimit.ts

interface RateLimitEntry {
    count: number;
    resetAt: number;
}

const limits = new Map<string, RateLimitEntry>();

// Clean up old entries periodically
setInterval(() => {
    const now = Date.now();
    for (const [key, entry] of limits) {
        if (entry.resetAt < now) {
            limits.delete(key);
        }
    }
}, 60000); // Every minute

export function checkRateLimit(
    key: string,
    maxRequests: number = 10,
    windowMs: number = 60000
): { allowed: boolean; remaining: number; resetIn: number } {
    const now = Date.now();
    const entry = limits.get(key);

    if (!entry || entry.resetAt < now) {
        // New window
        limits.set(key, {
            count: 1,
            resetAt: now + windowMs,
        });
        return {
            allowed: true,
            remaining: maxRequests - 1,
            resetIn: windowMs,
        };
    }

    if (entry.count >= maxRequests) {
        return {
            allowed: false,
            remaining: 0,
            resetIn: entry.resetAt - now,
        };
    }

    entry.count++;
    return {
        allowed: true,
        remaining: maxRequests - entry.count,
        resetIn: entry.resetAt - now,
```

```
    };
}
```

7.5.2 Usage in Hooks

```
// src/hooks.server.ts
import { checkRateLimit } from '$lib/server/ratelimit';
import { json } from '@sveltejs/kit';

export const handle: Handle = async ({ event, resolve }) => {
    // Rate limit auth endpoints
    if (event.url.pathname.startsWith('/api/auth')) {
        const ip = event.getClientAddress();
        const { allowed, resetIn } = checkRateLimit(
            `auth:${ip}`,
            5,      // 5 requests
            60000   // per minute
        );

        if (!allowed) {
            return json(
                { error: 'Too many requests. Try again later.' },
                {
                    status: 429,
                    headers: {
                        'Retry-After': Math.ceil(resetIn / 1000).toString(),
                    },
                }
            );
        }
    }

    return resolve(event);
};
```

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

```
// src/hooks.server.ts

const ALLOWED_ORIGINS = [
    'https://myapp.com',
    'https://mobile.myapp.com',
];

export const handle: Handle = async ({ event, resolve }) => {
    const origin = event.request.headers.get('origin');

    // Handle preflight
    if (event.request.method === 'OPTIONS') {
        if (origin && ALLOWED_ORIGINS.includes(origin)) {
            return new Response(null, {
                headers: {
                    'Access-Control-Allow-Origin': origin,
                    'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE',
                    'Access-Control-Allow-Headers': 'Content-Type',
                    'Access-Control-Max-Age': '86400',
                },
            });
        }
    }

    const response = await resolve(event);

    // Add CORS headers to response
    if (origin && ALLOWED_ORIGINS.includes(origin)) {
        response.headers.set('Access-Control-Allow-Origin', origin);
    }

    return response;
};
```

7.6.3 For Our Platform

We don't need CORS because: - Same-origin requests only - No external API consumers - SvelteKit handles everything

---

API Summary

7.7 Summary

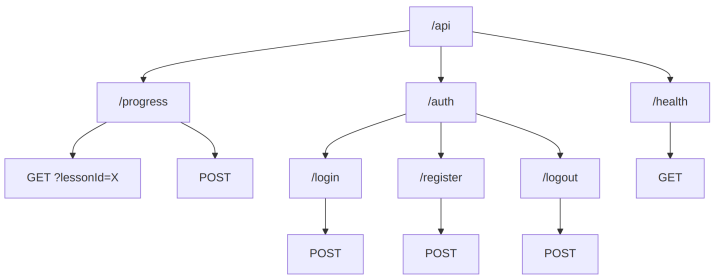

Figure 7.1: API Structure Overview

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

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

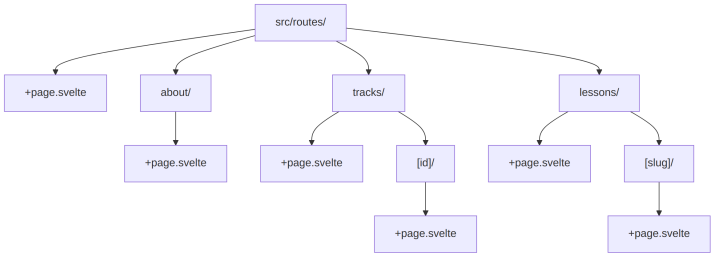

Figure 8.1: SvelteKit Basic Routing

8.1.3 Dynamic Parameters

```
// src/routes/tracks/[id]/+page.server.ts
export async function load({ params }) {
    // params.id = whatever is in the URL
    const trackId = params.id; // "hadith"

    // Fetch track...
}
```

8.1.4 Multiple Parameters

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

8.1.5 Optional Parameters

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

8.1.6 Rest Parameters

```
src/routes/files/[...path]/+page.svelte→
 /files/docs/images/photo.png→
 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/`:

```
// src/routes/admin/+page.server.ts
import { db } from '$lib/server/db';  // ✅ Server only

export async function load() {
```

```
    // This runs ONLY on the server
    const users = db.select().from(users).all();

    return { users };
}
```

8.2.2 Client Code

Regular `.svelte` files:

```
<!-- src/routes/+page.svelte -->
<script>
    // This runs in the browser
    import { onMount } from 'svelte';

    onMount(() => {
        console.log('Browser only!');
    });
</script>
```

8.2.3 Universal Code

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

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

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 8.3 Load Functions

Load functions fetch data before rendering.

8.3.1 Server Load

```
// src/routes/tracks/+page.server.ts
import type { PageServerLoad } from './$types';
import { contentDatabase, tracks } from '$lib/server/db';

export const load: PageServerLoad = async () => {
    const allTracks = contentDatabase
        .select()
        .from(tracks)
        .orderBy(tracks.order)
        .all();

    return {
        tracks: allTracks,
    };
};
```

8.3.2 Using Load Data

```
<!-- src/routes/tracks/+page.svelte -->
<script lang="ts">
    import type { PageData } from './$types';

    let { data }: { data: PageData } = $props();

    // TypeScript knows data.tracks is Track[]
</script>

{#each data.tracks as track}
    <h2>{track.title}</h2>
{/each}
```

8.3.3 Accessing User from Locals

```
// src/routes/dashboard/+page.server.ts
export const load: PageServerLoad = async ({ locals }) => {
    // locals.user comes from hooks.server.ts
    if (!locals.user) {
        throw redirect(302, '/auth/login');
    }

    return {
        user: locals.user,
    };
};
```

8.3.4 Dependent Data

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

```
    const lesson = contentDatabase
        .select()
        .from(lessons)
        .where(eq(lessons.slug, params.slug))
        .get();

    if (!lesson) {
        throw error(404, 'Lesson not found');
    }

    // Get videos for this lesson
    const lessonVideos = contentDatabase
        .select()
        .from(videos)
        .where(eq(videos.lessonId, lesson.id))
        .orderBy(videos.order)
        .all();

    // Get user progress (if logged in)
    let progress = null;
    if (locals.user) {
        progress = db
            .select()
            .from(lessonProgress)
            .where(
                and(
                    eq(lessonProgress.userId, locals.user.id),
                    eq(lessonProgress.lessonId, lesson.id)
                )
            )
            .get();
    }

    return {
        lesson,
        videos: lessonVideos,
        progress,
        isAuthenticated: !!locals.user,
    };
};
```

8.4 8.4 Form Actions

Handle form submissions server-side.

8.4.1 Basic Form

```
<!-- src/routes/auth/login/+page.svelte -->
<form method="POST">
    <input type="email" name="email" required />
    <input type="password" name="password" required />
    <button type="submit">Login</button>
</form>
```

8.4.2 Action Handler

```
// src/routes/auth/login/+page.server.ts
import type { Actions } from './$types';
import { fail, redirect } from '@sveltejs/kit';

export const actions: Actions = {
    default: async ({ request, cookies }) => {
        const form = await request.formData();
        const email = form.get('email') as string;
        const password = form.get('password') as string;

        // Validate
        if (!email || !password) {
            return fail(400, {
                error: 'Email and password required',
                email, // Return email to repopulate form
            });
        }

        // Check user
        const user = await validateUser(email, password);
        if (!user) {
            return fail(400, {
                error: 'Invalid credentials',
                email,
            });
        }

        // Create session
        const { sessionId, expiresAt } = createSession(user.id);
        setSessionCookie(cookies, sessionId, expiresAt);

        throw redirect(302, '/');
    }
};
```

8.4.3 Named Actions

```
<form method="POST" action="?/login">...</form>
<form method="POST" action="?/register">...</form>
```

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

8.4.4 Handling Action Results

```
<script lang="ts">
    import type { ActionData } from './$types';

    let { form }: { form: ActionData } = $props();
</script>

{#if form?.error}
```

```
    <div class="alert alert-error">{form.error}</div>
{/if}

<form method="POST">
    <input
        type="email"
        name="email"
        value={form?.email ?? ''}
    />
    <!-- ... -->
</form>
```

8.4.5 Progressive Enhancement

```
<script>
    import { enhance } from '$app/forms';
</script>

<form method="POST" use:enhance>
    <!-- Works without JS, better with JS -->
</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

```
<!-- src/routes/lessons/[slug]/+error.svelte -->
<script>
    import { page } from '$app/stores';
</script>

<div class="error-container">
    <h1>{$page.status}</h1>
    <p>{$page.error?.message}</p>
    <a href="/">Go Home</a>
</div>
```

8.5.2 Global Error Page

```
<!-- src/routes/+error.svelte -->
<script>
    import { page } from '$app/stores';
</script>

<div class="min-h-screen flex items-center justify-center">
    <div class="text-center">
        <h1 class="text-6xl font-bold text-error">
            {$page.status}
        </h1>
        <p class="text-xl mt-4">
            {#if $page.status === 404}
                ةحفصلا رىثع ذودجوم
            {:else if $page.status === 500}
                ثدح أطخ يف ماظنلا
            {:else}
                {$page.error?.message}
            {/if}
        </p>
        <a href="/" class="btn btn-primary mt-8">
            ةيسيئرلل ةدوعلا
        </a>
    </div>
</div>
```

8.5.3 Throwing Errors in Load

```
import { error } from '@sveltejs/kit';

export const load: PageServerLoad = async ({ params }) => {
    const lesson = getLesson(params.slug);

    if (!lesson) {
        throw error(404, 'Lesson not found');
    }

    if (!lesson.isPublished) {
        throw error(403, 'This lesson is not available');
    }

    return { lesson };
};
```

8.6 8.6 Layout Nesting

Share UI across routes with layouts.

8.6.1 Root Layout

```
<!-- src/routes/+layout.svelte -->
<script>
    import Navbar from '$lib/components/Navbar.svelte';
    import Footer from '$lib/components/Footer.svelte';
    import '../app.css';

    let { children, data } = $props();
</script>

<Navbar user={data.user} />

<main class="min-h-screen">
    {@render children()}
</main>

<Footer />
```

8.6.2 Layout Data

```
// src/routes/+layout.server.ts
export async function load({ locals }) {
    return {
        user: locals.user,
    };
}
```

8.6.3 Nested Layouts

```
src/routes/├──
 +layout.svelte          # Root (Navbar, Footer)├──
 admin/|
   ├── +layout.svelte      # Admin (Sidebar)|
   ├── +page.svelte        # /admin|
   └── lessons/|
       └── +page.svelte    # /admin/lessons
```

```
<!-- src/routes/admin/+layout.svelte -->
<script>
    let { children } = $props();
</script>

<div class="flex">
    <aside class="w-64 bg-base-200">
        <nav>
            <a href="/admin">Dashboard</a>
            <a href="/admin/lessons">Lessons</a>
            <a href="/admin/videos">Videos</a>
        </nav>
    </aside>

    <main class="flex-1 p-8">
        {@render children()}
    </main>
</div>
```

8.6.4 Breaking Out of Layout

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

```
src/routes/├──
 +layout.svelte          # Has Navbar└──
 auth/
    └── login/
        └── +page@.svelte   # No Navbar (uses root HTML only)
```

8.7 Summary

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

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

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

9.1.2 Vite Configuration

```
// vite.config.ts
import { sveltekit } from '@sveltejs/kit/vite';
import tailwindcss from '@tailwindcss/vite';
import { defineConfig } from 'vite';

export default defineConfig({
    plugins: [
        tailwindcss(),
        sveltekit()
    ]
});
```

9.1.3 CSS Entry Point

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

9.1.4 Import in Layout

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

9.1.5 Tailwind Basics

```
<!-- Spacing -->
<div class="p-4 m-2">Padding 4, Margin 2</div>

<!-- Flexbox -->
<div class="flex items-center justify-between">...</div>

<!-- Grid -->
<div class="grid grid-cols-3 gap-4">...</div>

<!-- Colors -->
<div class="bg-blue-500 text-white">Blue background</div>

<!-- Responsive -->
<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

```
bun add -d daisyui
```

9.2.2 Configuration

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

9.2.3 Core Components

Buttons:

```
<button class="btn">Default</button>
<button class="btn btn-primary">Primary</button>
<button class="btn btn-secondary">Secondary</button>
<button class="btn btn-ghost">Ghost</button>
<button class="btn btn-outline">Outline</button>
<button class="btn btn-sm">Small</button>
<button class="btn btn-lg">Large</button>
```

Cards:

```
<div class="card bg-base-100 shadow-xl">
    <figure><img src="/image.jpg" alt="Cover" /></figure>
    <div class="card-body">
        <h2 class="card-title">Title</h2>
        <p>Description</p>
        <div class="card-actions justify-end">
            <button class="btn btn-primary">Action</button>
        </div>
    </div>
</div>
```

Badges:

```
<span class="badge">Default</span>
<span class="badge badge-primary">Primary</span>
<span class="badge badge-lg">Large</span>
```

Alerts:

```
<div class="alert alert-success">Success message</div>
<div class="alert alert-error">Error message</div>
<div class="alert alert-warning">Warning message</div>
<div class="alert alert-info">Info message</div>
```

Form Controls:

```
<input type="text" class="input input-bordered w-full" />
<select class="select select-bordered">
    <option>Option 1</option>
</select>
<textarea class="textarea textarea-bordered"></textarea>
```

Navigation:

```
<div class="navbar bg-base-100">
    <div class="flex-1">
        <a class="btn btn-ghost text-xl">Brand</a>
    </div>
    <div class="flex-none">
        <ul class="menu menu-horizontal px-1">
            <li><a>Link 1</a></li>
            <li><a>Link 2</a></li>
        </ul>
    </div>
</div>
```

9.3 9.3 Theme System (Light/Dark)

DaisyUI includes multiple themes.

9.3.1 Available Themes

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

9.3.2 Setting Theme

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

9.3.3 Dynamic Theme Switching

```
<!-- ThemeSwitcher.svelte -->
<script lang="ts">
    import { browser } from '$app/environment';

    let theme = $state(
        browser ? localStorage.getItem('theme') || 'light' : 'light'
    );

    function toggleTheme() {
        theme = theme === 'light' ? 'dark' : 'light';
        document.documentElement.setAttribute('data-theme', theme);
        localStorage.setItem('theme', theme);
    }
</script>

<button class="btn btn-ghost" onclick={toggleTheme}>
    {theme === 'light' ? '' : ''}
</button>
```

9.3.4 Initialize Theme

```
<!-- +layout.svelte -->
<script>
    import { browser } from '$app/environment';
    import { onMount } from 'svelte';

    onMount(() => {
        const saved = localStorage.getItem('theme') || 'light';
        document.documentElement.setAttribute('data-theme', saved);
```

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

9.4 9.4 RTL Support for Arabic

Arabic text flows right-to-left.

9.4.1 HTML Direction

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

9.4.2 Dynamic Direction

```
<script>
    let locale = $state('ar');
</script>

<svelte:body dir={locale === 'ar' ? 'rtl' : 'ltr'} />
```

9.4.3 RTL Utilities

```
/* src/app.css */
@layer utilities {
    /* Flip margins/paddings */
    [dir="rtl"] .ml-4 {
        margin-left: 0;
        margin-right: 1rem;
    }

    /* Text alignment */
    [dir="rtl"] {
        text-align: right;
    }
}
```

9.4.4 Logical Properties (CSS)

Modern CSS supports logical properties:

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

9.4.5 DaisyUI RTL

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

```
<div class="navbar bg-base-100" dir="rtl">
    <!-- Components auto-adjust -->
</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:

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

Hidden on Mobile:

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

Responsive Text:

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

Responsive Padding:

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

9.5.3 Mobile Navigation Pattern

```
<script>
    let isMenuOpen = $state(false);
</script>

<!-- Desktop Nav -->
<nav class="hidden md:flex gap-4">
    <a href="/">Home</a>
    <a href="/tracks">Tracks</a>
</nav>

<!-- Mobile Menu Button -->
<button class="md:hidden" onclick={() => isMenuOpen = !isMenuOpen}>
    ≡
</button>

<!-- Mobile Drawer -->
{#if isMenuOpen}
    <div class="fixed inset-0 z-50 md:hidden">
        <div class="bg-black/50" onclick={() => isMenuOpen = false}></div>
        <nav class="fixed right-0 top-0 h-full w-64 bg-base-100 p-4">
            <a href="/">Home</a>
            <a href="/tracks">Tracks</a>
        </nav>
    </div>
{/if}
```

9.6 9.6 Custom Component Library

Build reusable components with consistent styling.

9.6.1 Button Component

```
<!-- src/lib/components/Button.svelte -->
<script lang="ts">
    interface Props {
        variant?: 'primary' | 'secondary' | 'ghost' | 'outline';
        size?: 'sm' | 'md' | 'lg';
        loading?: boolean;
        disabled?: boolean;
        type?: 'button' | 'submit';
        onclick?: () => void;
    }

    let {
        variant = 'primary',
        size = 'md',
        loading = false,
        disabled = false,
        type = 'button',
        onclick,
        children,
    }: Props & { children: any } = $props();

    const variantClasses = {
        primary: 'btn-primary',
        secondary: 'btn-secondary',
        ghost: 'btn-ghost',
        outline: 'btn-outline',
    };

    const sizeClasses = {
        sm: 'btn-sm',
        md: '',
        lg: 'btn-lg',
    };
</script>

<button
    {type}
    class="btn {variantClasses[variant]} {sizeClasses[size]}"
    disabled={disabled || loading}
    {onclick}
>
    {#if loading}
        <span class="loading loading-spinner loading-sm"></span>
    {/if}
    {@render children()}
</button>
```

9.6.2 Card Component

```
<!-- src/lib/components/Card.svelte -->
<script lang="ts">
    interface Props {
        title?: string;
        image?: string;
        href?: string;
        class?: string;
    }

    let { title, image, href, class: className = '', children }: Props & { children: any } =
    $props();
```

```
</script>

<svelte:element
    this={href ? 'a' : 'div'}
    {href}
    class="card bg-base-100 shadow-lg hover:shadow-xl transition-shadow {className}"
>
    {#if image}
        <figure>
            <img src={image} alt={title || ''} class="w-full h-48 object-cover" />
        </figure>
    {/if}
    <div class="card-body">
        {#if title}
            <h3 class="card-title">{title}</h3>
        {/if}
        {@render children()}
    </div>
</svelte:element>
```

9.6.3 Input Component

```
<!-- src/lib/components/Input.svelte -->
<script lang="ts">
    interface Props {
        label?: string;
        name: string;
        type?: 'text' | 'email' | 'password' | 'number';
        placeholder?: string;
        value?: string;
        error?: string;
        required?: boolean;
    }

    let {
        label,
        name,
        type = 'text',
        placeholder,
        value = '',
        error,
        required = false,
    }: Props = $props();
</script>

<div class="form-control w-full">
    {#if label}
        <label class="label" for={name}>
            <span class="label-text">{label}</span>
            {#if required}
                <span class="text-error">*</span>
            {/if}
        </label>
    {/if}

    <input
        {type}
        {name}
        id={name}
        {placeholder}
```

```
        {value}
        {required}
        class="input input-bordered w-full {error ? 'input-error' : ''}"
    />

    {#if error}
        <label class="label">
            <span class="label-text-alt text-error">{error}</span>
        </label>
    {/if}
</div>
```

9.6.4 Usage

```
<script>
    import Button from '$lib/components/Button.svelte';
    import Card from '$lib/components/Card.svelte';
    import Input from '$lib/components/Input.svelte';
</script>

<Button variant="primary" loading={isSubmitting}>
    Submit
</Button>

<Card title="Lesson Title" image="/cover.jpg" href="/lessons/1">
    <p>Lesson description...</p>
</Card>

<Input
    label="Email"
    name="email"
    type="email"
    error={form?.errors?.email}
    required
/>
```

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

```
// src/routes/+page.server.ts
import type { PageServerLoad } from './$types';
import { contentDatabase, tracks, series, lessons, videos } from '$lib/server/db';
import { count, eq, desc } from 'drizzle-orm';

export const load: PageServerLoad = async () => {
    // Get counts
    const tracksCount = contentDatabase
        .select({ count: count() })
        .from(tracks)
        .get()?.count ?? 0;

    const seriesCount = contentDatabase
        .select({ count: count() })
        .from(series)
        .get()?.count ?? 0;

    const lessonsCount = contentDatabase
        .select({ count: count() })
        .from(lessons)
        .where(eq(lessons.isPublished, true))
        .get()?.count ?? 0;

    const videosCount = contentDatabase
        .select({ count: count() })
        .from(videos)
        .get()?.count ?? 0;

    // Get latest lessons
    const latestLessons = contentDatabase
        .select()
        .from(lessons)
        .where(eq(lessons.isPublished, true))
        .orderBy(desc(lessons.createdAt))
```

```
        .limit(6)
        .all();

    // Get all tracks for display
    const allTracks = contentDatabase
        .select()
        .from(tracks)
        .orderBy(tracks.order)
        .all();

    return {
        stats: {
            tracks: tracksCount,
            series: seriesCount,
            lessons: lessonsCount,
            videos: videosCount,
        },
        latestLessons,
        tracks: allTracks,
    };
};
```

10.1.2 Homepage Component

```
<!-- src/routes/+page.svelte -->
<script lang="ts">
    import type { PageData } from './$types';
    import { getTrackColor, getTrackIcon } from '$lib/theme';

    let { data }: { data: PageData } = $props();
</script>

<svelte:head>
    <title>أفدنا - منصة تعليمية للإسلام</title>
</svelte:head>

<!-- Hero Section -->
<section class="hero min-h-[70vh] bg-gradient-to-br from-primary to-primary-focus text-primary
    -content">
    <div class="hero-content text-center">
        <div class="max-w-2xl">
            <h1 class="text-5xl font-bold mb-6">
                أفدنا بما علّمك الله
            </h1>
            <p class="text-xl mb-8 opacity-90">
                منصة تعليمية رشيقة لعلم العقيدة
            </p>
            <div class="flex gap-4 justify-center">
                <a href="/tracks" class="btn btn-secondary btn-lg">
                    ابدأ التعلم
                </a>
                <a href="/about" class="btn btn-ghost btn-lg">
                    تعرف على المزيد
                </a>
            </div>
        </div>
    </div>
</section>

<!-- Stats Section -->
```

```
<section class="py-16 bg-base-200">
    <div class="container mx-auto px-4">
        <div class="stats stats-vertical lg:stats-horizontal shadow w-full">
            <div class="stat">
                <div class="stat-figure text-primary text-3xl"></div>
                <div class="stat-titleالمسارات"></div>
                <div class="stat-value text-primary">{data.stats.tracks}</div>
            </div>
            <div class="stat">
                <div class="stat-figure text-secondary text-3xl"></div>
                <div class="stat-titleالسلاسل"></div>
                <div class="stat-value text-secondary">{data.stats.series}</div>
            </div>
            <div class="stat">
                <div class="stat-figure text-accent text-3xl"></div>
                <div class="stat-titleالدروس"></div>
                <div class="stat-value text-accent">{data.stats.lessons}</div>
            </div>
            <div class="stat">
                <div class="stat-figure text-info text-3xl"></div>
                <div class="stat-titleالفيديوهات"></div>
                <div class="stat-value text-info">{data.stats.videos}</div>
            </div>
        </div>
    </div>
</section>

<!-- Tracks Section -->
<section class="py-16 bg-base-100">
    <div class="container mx-auto px-4">
        <h2 class="text-3xl font-bold text-center mbالمسارات-12">العلمية</h2>

        <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
            {#each data.tracks as track}
                <a
                    href="/tracks/{track.id}"
                    class="card bg-gradient-to-br {getTrackColor(track.id)} text-white shadow-
    lg hover:scale-105 transition-transform"
                >
                    <div class="card-body">
                        <span class="text-4xl">{getTrackIcon(track.id, track.icon)}</span>
                        <h3 class="card-title text-white">{track.title}</h3>
                        {#if track.description}
                            <p class="opacity-90">{track.description}</p>
                        {/if}
                    </div>
                </a>
            {/each}
        </div>
    </div>
</section>

<!-- Latest Lessons -->
<section class="py-16 bg-base-200">
    <div class="container mx-auto px-4">
        <h2 class="text-3xl font-bold text-center mbتدأ-12">الدروس</h2>

        <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
            {#each data.latestLessons as lesson}
                <a href="/lessons/{lesson.slug}" class="card bg-base-100 shadow-lg hover:
    shadow-xl transition-shadow">
                    <div class="card-body">
                        <h3 class="card-title">{lesson.title}</h3>
                        {#if lesson.description}
```

```
                            <p class="text-base-content/70 line-clamp-2">{lesson.description
    }</p>
                        {/if}
                        {#if lesson.instructor}
                            <p class="text-sm text-base-content/60"> {lesson.instructor}</p>
                        {/if}
                    </div>
                </a>
            {/each}
        </div>

        <div class="text-center mt-8">
            <a href="/lessons" class="btn btn-primary">
                عرض جميع الدروس
            </a>
        </div>
    </div>
</section>
```

10.2 10.2 Content Listing (Tracks, Lessons)

List pages with filtering and search.

10.2.1 Tracks Listing

```
// src/routes/tracks/+page.server.ts
export const load: PageServerLoad = async () => {
    const allTracks = contentDatabase
        .select()
        .from(tracks)
        .orderBy(tracks.order)
        .all();

    // Get lesson count per track
    const tracksWithCount = allTracks.map(track => {
        const lessonCount = contentDatabase
            .select({ count: count() })
            .from(lessons)
            .where(and(
                eq(lessons.trackId, track.id),
                eq(lessons.isPublished, true)
            ))
            .get()?.count ?? 0;

        return { ...track, lessonCount };
    });

    return { tracks: tracksWithCount };
};
```

10.2.2 Lessons Listing with Search

```svelte
<!-- src/routes/lessons/+page.svelte -->
<script lang="ts">
    import type { PageData } from './$types';

    let { data }: { data: PageData } = $props();

    let searchQuery = $state('');
    let selectedTrack = $state('all');

    const filteredLessons = $derived(
        data.lessons.filter(lesson => {
            const matchesSearch = lesson.title.toLowerCase()
                .includes(searchQuery.toLowerCase());
            const matchesTrack = selectedTrack === 'all' ||
                lesson.trackId === selectedTrack;
            return matchesSearch && matchesTrack;
        })
    );
</script>

<section class="py-8">
    <div class="container mx-auto px-4">
        <!-- Filters -->
        <div class="card bg-base-100 shadow mb-8">
            <div class="card-body">
                <div class="flex flex-wrap gap-4 items-center">
                    <input
                        type="text"
                        bind:value={searchQuery}
                        placeholderبحث="ابحث عن درس..."
                        class="input input-bordered flex-1 min-w-[200px]"
                    />

                    <select bind:value={selectedTrack} class="select select-bordered">
                        <option value="allكل">جميع المسارات</option>
                        {#each data.tracks as track}
                            <option value={track.id}>{track.title}</option>
                        {/each}
                    </select>

                    <span class="badge badge-lg">
                        {filteredLessons.length} درس
                    </span>
                </div>
            </div>
        </div>

        <!-- Lessons Grid -->
        <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
            {#each filteredLessons as lesson}
                <a href="/lessons/{lesson.slug}" class="card bg-base-100 shadow hover:shadow-
    lg transition-shadow">
                    <div class="card-body">
                        <h3 class="card-title">{lesson.title}</h3>
                        <p class="text-sm text-base-content/70 line-clamp-2">
                            {lesson.description || 'لا يوجد وصف'}
                        </p>
                    </div>
                </a>
            {/each}
        </div>
```

```
        {#if filteredLessons.length === 0}
            <div class="text-center py-16">
                <span class="text-6xl mb-4 block"></span>
                <h3 class="text-xl font-boldا"> دجوت جئاتن</h3>
                <p class="text-base-contentبرج/70"> تاملك تحب ةفلتخم</p>
            </div>
        {/if}
    </div>
</section>
```

10.3 10.3 Detail Pages

Single item pages with rich content.

10.3.1 Single Track Page

```
// src/routes/tracks/[id]/+page.server.ts
export const load: PageServerLoad = async ({ params }) => {
    const track = contentDatabase
        .select()
        .from(tracks)
        .where(eq(tracks.id, params.id))
        .get();

    if (!track) {
        throw error(404, 'Track not found');
    }

    // Get series in this track
    const trackSeries = contentDatabase
        .select()
        .from(series)
        .where(eq(series.trackId, params.id))
        .orderBy(series.order)
        .all();

    // Get standalone lessons (no series)
    const standaloneLessons = contentDatabase
        .select()
        .from(lessons)
        .where(and(
            eq(lessons.trackId, params.id),
            isNull(lessons.seriesId),
            eq(lessons.isPublished, true)
        ))
        .orderBy(lessons.order)
        .all();

    return {
        track,
        series: trackSeries,
```

```
        standaloneLessons,
    };
};
```

10.3.2 Single Lesson Page

See Chapter 12 for the full VideoPlayer integration.

10.4 10.4 Search Functionality

Global search across all content.

```
// src/routes/search/+page.server.ts
export const load: PageServerLoad = async ({ url }) => {
    const query = url.searchParams.get('q')?.trim() || '';

    if (query.length < 2) {
        return { query: '', results: { lessons: [], tracks: [] } };
    }

    const pattern = `%${query}%`;

    const lessonResults = contentDatabase
        .select()
        .from(lessons)
        .where(and(
            eq(lessons.isPublished, true),
            or(
                like(lessons.title, pattern),
                like(lessons.description, pattern)
            )
        ))
        .limit(20)
        .all();

    const trackResults = contentDatabase
        .select()
        .from(tracks)
        .where(or(
            like(tracks.title, pattern),
            like(tracks.description, pattern)
        ))
        .all();

    return {
        query,
        results: {
            lessons: lessonResults,
            tracks: trackResults,
        },
    };
};
```

```
<!-- src/routes/search/+page.svelte -->
<script lang="ts">
    import { goto } from '$app/navigation';

    let { data } = $props();
    let searchInput = $state(data.query);

    function handleSearch(e: Event) {
        e.preventDefault();
        if (searchInput.trim().length >= 2) {
            goto(`/search?q=${encodeURIComponent(searchInput)}`);
        }
    }
</script>

<form onsubmit={handleSearch} class="max-w-xl mx-auto mb-8">
    <div class="join w-full">
        <input
            type="text"
            bind:value={searchInput}
            placeholderتبا="..."
            class="input input-bordered join-item flex-1"
        />
        <button class="btn btn-primary join-itemتب"></button>
    </div>
</form>

{#if data.results.lessons.length > 0}
    <h2 class="text-xl font-bold mbدروسا-4"></h2>
    <!-- Render lessons... -->
{/if}

{#if data.results.tracks.length > 0}
    <h2 class="text-xl font-bold mbتاراسما-4"></h2>
    <!-- Render tracks... -->
{/if}
```

10.5 10.5 User Dashboard

Personal progress page.

```
// src/routes/dashboard/+page.server.ts
export const load: PageServerLoad = async ({ locals }) => {
    if (!locals.user) {
        throw redirect(302, '/auth/login?redirect=/dashboard');
    }

    const progress = db
        .select()
        .from(lessonProgress)
        .where(eq(lessonProgress.userId, locals.user.id))
        .all();

    const completedCount = progress.filter(p => p.videoCompleted).length;
    const totalWatchTime = progress.reduce((sum, p) => sum + p.watchedSeconds, 0);
```

```

    return {
        user: locals.user,
        stats: {
            started: progress.length,
            completed: completedCount,
            totalMinutes: Math.floor(totalWatchTime / 60),
        },
        recentProgress: progress.slice(0, 10),
    };
};
```

10.6 10.6 Error and 404 Pages

Handle errors gracefully.

```
<!-- src/routes/+error.svelte -->
<script>
    import { page } from '$app/stores';
</script>

<div class="min-h-screen flex items-center justify-center bg-base-200">
    <div class="text-center p-8">
        <div class="text-8xl mb-6">
            {#if $page.status === 404}
                🔍
            {:else if $page.status === 403}
                🔒
            {:else}
                ⚠️
            {/if}
        </div>

        <h1 class="text-4xl font-bold mb-4">
            {#if $page.status === 404}
                ةحفصلا رثع ةدوجوم
            {:else if $page.status === 403}
                رثع حرصم لوصولاب
            {:else}
                ثدح أطخ
            {/if}
        </h1>

        <p class="text-base-content/70 mb-8">
            {$page.error?.message || ثدح' أطخ رثع عقوتم'}
        </p>

        <a href="/" class="btn btn-primary">
            ةدوعلا ةيسيئرلل
        </a>
    </div>
</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.

```
<!-- src/lib/components/Navbar.svelte -->
<script lang="ts">
    import type { User } from '$lib/server/db/schema';
    import ThemeSwitcher from './ThemeSwitcher.svelte';

    interface Props {
        user: User | null;
    }

    let { user }: Props = $props();
    let mobileMenuOpen = $state(false);

    const navLinks = [
        { href: '/', label: 'الرئيسية' },
        { href: '/tracks', label: 'المسارات' },
        { href: '/lessons', label: 'الدروس' },
        { href: '/search', label: 'بحث' },
    ];
</script>

<nav class="navbar bg-base-100 shadow-lg sticky top-0 z-50">
    <div class="container mx-auto">
        <!-- Logo -->
        <div class="flex-1">
            <a href="/" class="btn btn-ghost text-xl font-bold">
                 أفدنا
            </a>
        </div>

        <!-- Desktop Menu -->
        <div class="hidden md:flex items-center gap-2">
            {#each navLinks as link}
                <a href={link.href} class="btn btn-ghost btn-sm">
                    {link.label}
                </a>
            {/each}

            <div class="divider divider-horizontal mx-2"></div>

            <ThemeSwitcher />
```

```
41
42              {#if user}
43                  <div class="dropdown dropdown-end">
44                      <button class="btn btn-ghost btn-circle avatar">
45                          <div class="w-10 rounded-full bg-primary text-primary-content flex
    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="/dashboardةحول"> مكتملا</a></li>
52                          {#if user.role === 'admin' || user.role === 'editor'}
53                              <li><a href="/adminةدارإلا"></a></li>
54                          {/if}
55                          <li>
56                              <form method="POST" action="/auth/logout">
57                                  <button type="submit" class="w-full text-right">
58                                      لجتسا جورخلا
59                                  </button>
60                              </form>
61                          </li>
62                      </ul>
63                  </div>
64              {:else}
65                  <a href="/auth/login" class="btn btn-primary btn-sm">
66                      لوخد
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"> اأفدن</span>
94                  <button
95                      class="btn btn-ghost btn-sm"
96                      onclick={() => mobileMenuOpen = false}
97                  >
98                      ×
99                  </button>
100             </div>
101
102             <ul class="menu">
```

```
                {#each navLinks as link}
                    <li>
                        <a href={link.href} onclick={() => mobileMenuOpen = false}>
                            {link.label}
                        </a>
                    </li>
                {/each}
            </ul>

            <div class="divider"></div>

            {#if user}
                <div class="p-4 bg-base-200 rounded-lg mb-4">
                    <p class="font-bold">{user.name}</p>
                    <p class="text-sm text-base-content/70">{user.email}</p>
                </div>
                <form method="POST" action="/auth/logout">
                    <button class="btn btn-outline btn-blockلگیجست"> جورخا</button>
                </form>
            {:else}
                <a href="/auth/login" class="btn btn-primary btn-block">
                    لگیجست ولوخدا
                </a>
            {/if}
        </div>
    </div>
{/if}
```

11.2 11.2 Footer Component

```
<!-- src/lib/components/Footer.svelte -->
<script lang="ts">
    const currentYear = new Date().getFullYear();

    const links = {
        main: [
            { href: '/tracks', label: تاراسمال'' },
            { href: '/lessons', label: سوردال'' },
            { href: '/about', label: نع' عقوملا' },
        ],
        legal: [
            { href: '/privacy', label: ةيصوصخلا'' },
            { href: '/terms', label: طورشلا'' },
        ],
    };
</script>

<footer class="footer footer-center p-10 bg-base-200 text-base-content">
    <nav class="grid grid-flow-col gap-4">
        {#each links.main as link}
            <a href={link.href} class="link link-hover">{link.label}</a>
        {/each}
    </nav>

    <nav>
        <div class="grid grid-flow-col gap-4">
```

```
            <!-- Social Icons -->
            <a href="https://twitter.com" target="_blank" class="btn btn-ghost btn-circle">
                
            </a>
            <a href="https://youtube.com" target="_blank" class="btn btn-ghost btn-circle">
                ▶
            </a>
            <a href="https://telegram.org" target="_blank" class="btn btn-ghost btn-circle">
                ✈
            </a>
        </div>
    </nav>

    <aside>
        <p>جميع الحقوق محفوظة © {currentYear} أفدني</p>
    </aside>
</footer>
```

11.3 11.3 Card Components

11.3.1 LessonCard

```
<!-- src/lib/components/LessonCard.svelte -->
<script lang="ts">
    import type { Lesson } from '$lib/server/db/schema-content';
    import { getTrackColor } from '$lib/theme';

    interface Props {
        lesson: Lesson & { track?: { id: string; title: string } };
        progress?: number;
    }

    let { lesson, progress = 0 }: Props = $props();
</script>

<a
    href="/lessons/{lesson.slug}"
    class="card bg-base-100 shadow-lg hover:shadow-xl transition-all hover:-translate-y-1"
>
    {#if lesson.thumbnailUrl}
        <figure>
            <img src={lesson.thumbnailUrl} alt={lesson.title} class="h-40 w-full object-cover"
 />
        </figure>
    {/if}

    <div class="card-body">
        {#if lesson.track}
            <span class="badge badge-sm {getTrackColor(lesson.track.id)}">
                {lesson.track.title}
            </span>
        {/if}

        <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}% لمتكم</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} درس</span>
26             </div>
27         {/if}
28     </div>
29 </a>
```

11.4 11.4 Modal System

```svelte
<!-- src/lib/components/Modal.svelte -->
<script lang="ts">
    interface Props {
        open: boolean;
        title?: string;
        onClose: () => void;
    }

    let { open, title, onClose, children }: Props & { children: any } = $props();

    function handleKeydown(e: KeyboardEvent) {
        if (e.key === 'Escape' && open) {
            onClose();
        }
    }
</script>

<svelte:window onkeydown={handleKeydown} />

{#if open}
    <div class="modal modal-open">
        <!-- Backdrop -->
        <div class="modal-backdrop bg-black/50" onclick={onClose}></div>

        <!-- Modal Box -->
        <div class="modal-box relative">
            <!-- Close Button -->
            <button
                class="btn btn-sm btn-circle btn-ghost absolute right-2 top-2"
                onclick={onClose}
            >
                ×
            </button>

            {#if title}
                <h3 class="font-bold text-lg mb-4">{title}</h3>
            {/if}

            {@render children()}
        </div>
    </div>
{/if}
```

11.4.1 Usage

```svelte
<script>
    import Modal from '$lib/components/Modal.svelte';

    let showModal = $state(false);
</script>

<button class="btn" onclick={() => showModal = true}>
    Open Modal
</button>

<Modal
    open={showModal}
```

```
    title="Confirm Action"
    onClose={() => showModal = false}
>
    <p>Are you sure you want to continue?</p>
    <div class="modal-action">
        <button class="btn btn-ghost" onclick={() => showModal = false}>
            Cancel
        </button>
        <button class="btn btn-primary">
            Confirm
        </button>
    </div>
</Modal>
```

11.5 11.5 Toast Notifications

```
<!-- src/lib/components/Toast.svelte -->
<script lang="ts">
    import { fly } from 'svelte/transition';

    interface Toast {
        id: string;
        message: string;
        type: 'success' | 'error' | 'warning' | 'info';
    }

    let toasts = $state<Toast[]>([]);

    export function showToast(message: string, type: Toast['type'] = 'info') {
        const id = crypto.randomUUID();
        toasts = [...toasts, { id, message, type }];

        setTimeout(() => {
            toasts = toasts.filter(t => t.id !== id);
        }, 4000);
    }

    const icons = {
        success: ✓'',
        error: ✕'',
        warning: ⚠'',
        info: i'',
    };

    const classes = {
        success: 'alert-success',
        error: 'alert-error',
        warning: 'alert-warning',
        info: 'alert-info',
    };
</script>

<div class="toast toast-top toast-end z-50">
    {#each toasts as toast (toast.id)}
        <div
            class="alert {classes[toast.type]}"
```

```
            transition:fly={{ x: 100, duration: 300 }}
        >
            <span class="text-xl">{icons[toast.type]}</span>
            <span>{toast.message}</span>
        </div>
    {/each}
</div>
```

11.5.1 Usage with Store

```
// src/lib/stores/toast.ts
import { writable } from 'svelte/store';

interface Toast {
    id: string;
    message: string;
    type: 'success' | 'error' | 'warning' | 'info';
}

function createToastStore() {
    const { subscribe, update } = writable<Toast[]>([]);

    return {
        subscribe,
        show(message: string, type: Toast['type'] = 'info') {
            const id = crypto.randomUUID();
            update(toasts => [...toasts, { id, message, type }]);

            setTimeout(() => {
                update(toasts => toasts.filter(t => t.id !== id));
            }, 4000);
        },
        success(message: string) { this.show(message, 'success'); },
        error(message: string) { this.show(message, 'error'); },
        warning(message: string) { this.show(message, 'warning'); },
    };
}

export const toast = createToastStore();
```

11.6 11.6 Loading States

```
<!-- src/lib/components/LoadingSpinner.svelte -->
<script lang="ts">
    interface Props {
        size?: 'sm' | 'md' | 'lg';
        text?: string;
    }

    let { size = 'md', text }: Props = $props();

    const sizeClasses = {
```

```
        sm: 'loading-sm',
        md: 'loading-md',
        lg: 'loading-lg',
    };
</script>

<div class="flex flex-col items-center justify-center gap-2">
    <span class="loading loading-spinner {sizeClasses[size]}"></span>
    {#if text}
        <span class="text-sm text-base-content/70">{text}</span>
    {/if}
</div>
```

11.6.1 Skeleton Loading

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

11.6.2 Page Loading

```
<!-- src/routes/+layout.svelte -->
<script>
    import { navigating } from '$app/stores';
    import LoadingSpinner from '$lib/components/LoadingSpinner.svelte';
</script>

{#if $navigating}
    <div class="fixed inset-0 bg-base-100/80 z-50 flex items-center justify-center">
        <LoadingSpinner size="lg" textری|ج=" لۆمەتـل|..." />
    </div>
{/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

```
// src/lib/utils/youtube.ts
let apiLoaded = false;
let apiLoadPromise: Promise<void> | null = null;

export function loadYouTubeAPI(): Promise<void> {
    if (apiLoaded) return Promise.resolve();
    if (apiLoadPromise) return apiLoadPromise;

    apiLoadPromise = new Promise((resolve) => {
        const script = document.createElement('script');
        script.src = 'https://www.youtube.com/iframe_api';

        (window as any).onYouTubeIframeAPIReady = () => {
            apiLoaded = true;
            resolve();
        };

        document.head.appendChild(script);
    });

    return apiLoadPromise;
}
```

12.1.2 API Reference

```
interface YT.Player {
    playVideo(): void;
    pauseVideo(): void;
    seekTo(seconds: number): void;
    getCurrentTime(): number;
    getDuration(): number;
```

```
    getPlayerState(): number;
}

// Player States
const PlayerState = {
    UNSTARTED: -1,
    ENDED: 0,
    PLAYING: 1,
    PAUSED: 2,
    BUFFERING: 3,
    CUED: 5,
};
```

12.2 12.2 Building VideoPlayer Component

```
<!-- src/lib/components/VideoPlayer.svelte -->
<script lang="ts">
    import { onMount, onDestroy } from 'svelte';
    import { loadYouTubeAPI } from '$lib/utils/youtube';
    import { browser } from '$app/environment';

    interface Props {
        youtubeId: string;
        title?: string;
        startTime?: number;
        onProgress?: (seconds: number) => void;
        onComplete?: () => void;
    }

    let {
        youtubeId,
        title = 'Video',
        startTime = 0,
        onProgress,
        onComplete
    }: Props = $props();

    let containerRef: HTMLDivElement;
    let player: YT.Player | null = null;
    let progressInterval: number | null = null;
    let lastReportedTime = 0;

    onMount(async () => {
        if (!browser || !youtubeId) return;

        await loadYouTubeAPI();
        initPlayer();
    });

    onDestroy(() => {
        if (progressInterval) clearInterval(progressInterval);
        player?.destroy();
    });

    function initPlayer() {
        player = new YT.Player(containerRef, {
```

```
            videoId: youtubeId,
            playerVars: {
                autoplay: 0,
                modestbranding: 1,
                rel: 0,
                start: Math.floor(startTime),
            },
            events: {
                onReady: handleReady,
                onStateChange: handleStateChange,
            },
        });
    }

    function handleReady() {
        console.log('Player ready');
    }

    function handleStateChange(event: YT.OnStateChangeEvent) {
        const state = event.data;

        if (state === YT.PlayerState.PLAYING) {
            startProgressTracking();
        } else if (state === YT.PlayerState.PAUSED) {
            stopProgressTracking();
        } else if (state === YT.PlayerState.ENDED) {
            stopProgressTracking();
            onComplete?.();
        }
    }

    function startProgressTracking() {
        if (progressInterval) return;

        progressInterval = setInterval(() => {
            if (!player) return;

            const currentTime = player.getCurrentTime();

            // Report every 30 seconds
            if (currentTime - lastReportedTime >= 30) {
                lastReportedTime = currentTime;
                onProgress?.(currentTime);
            }
        }, 5000) as unknown as number;
    }

    function stopProgressTracking() {
        if (progressInterval) {
            clearInterval(progressInterval);
            progressInterval = null;
        }

        // Report final time
        if (player) {
            onProgress?.(player.getCurrentTime());
        }
    }
</script>

<div class="video-player">
    <div class="aspect-video bg-black rounded-lg overflow-hidden">
        <div bind:this={containerRef} class="w-full h-full"></div>
    </div>
```

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

12.3 12.3 Progress Tracking

12.3.1 The 10-Minute Rule

```
// src/lib/utils/progress.ts

const TEN_MINUTES = 600; // seconds

export function shouldMarkComplete(
    watchedSeconds: number,
    videoDuration: number
): boolean {
    // Complete if watched 10+ minutes OR reached end
    return watchedSeconds >= TEN_MINUTES ||
           watchedSeconds >= videoDuration * 0.9;
}
```

12.3.2 Reporting Progress

```
async function reportProgress(lessonId: string, seconds: number) {
    try {
        await fetch('/api/progress', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({
                lessonId,
                watchedSeconds: Math.floor(seconds),
            }),
        });
    } catch (error) {
        console.error('Failed to report progress:', error);
    }
}
```

12.4 12.4 Handling Events

```
<script>
    function handleProgress(seconds: number) {
        // Save progress
        reportProgress(lesson.id, seconds);
    }

    function handleComplete() {
        // Mark as complete
        fetch('/api/progress', {
            method: 'POST',
            body: JSON.stringify({
                lessonId: lesson.id,
                videoCompleted: true,
            }),
        });

        // Move to next video or show quiz
        if (hasQuiz) {
            showQuiz = true;
        } else if (nextVideo) {
            currentVideoIndex++;
        }
    }
</script>

<VideoPlayer
    youtubeId={video.youtubeId}
    title={video.title}
    startTime={progress?.watchedSeconds || 0}
    {onProgress}
    onComplete={handleComplete}
/>
```

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

```
<button
    class="btn btn-ghost btn-sm"
    onclick={() => containerRef.requestFullscreen()}
>
    ⛶ Fullscreen
</button>
```

Chapter 13

Interactive Quiz System

13.1 13.1 Quiz Data Structure

```
interface Question {
    id: string;
    text: string;
    options: string[];
    correctIndex: number;
    points?: number;
}

interface Quiz {
    questions: Question[];
    passingScore: number; // percentage
}
```

13.1.1 Storing in Database

```
-- In videos table
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

```sql
CREATE TABLE lesson_progress (
    id INTEGER PRIMARY KEY,
    user_id INTEGER NOT NULL,
    lesson_id TEXT NOT NULL,
    watched_seconds INTEGER DEFAULT 0,
    video_completed INTEGER DEFAULT 0,
    quiz_passed INTEGER DEFAULT 0,
    quiz_score INTEGER DEFAULT 0,
    updated_at TEXT,
    UNIQUE(user_id, lesson_id)
);
```

14.1.2 UPSERT Logic

```javascript
// Check if exists
const existing = db.select()...

if (existing) {
    db.update(lessonProgress)
        .set({
            watchedSeconds: Math.max(existing.watchedSeconds, newSeconds),
            ...
        })
        .where(eq(lessonProgress.id, existing.id));
} else {
    db.insert(lessonProgress).values({...});
}
```

Chapter 15

Global Search

15.1 15.1-15.6 Search Implementation

15.1.1 SQLite LIKE Search

```
const pattern = `%${query}%`;

const results = contentDatabase
    .select()
    .from(lessons)
    .where(or(
        like(lessons.title, pattern),
        like(lessons.description, pattern)
    ))
    .all();
```

15.1.2 Search Page

```
<form onsubmit={handleSearch}>
    <input bind:value={query} placeholder="Search..." />
    <button>Search</button>
</form>

{#each results.lessons as lesson}
    <LessonCard {lesson} />
{/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

```
// src/routes/admin/+layout.server.ts
import { redirect } from '@sveltejs/kit';
import { isEditor } from '$lib/server/auth';

export async function load({ locals }) {
    if (!locals.user) {
        throw redirect(302, '/auth/login?redirect=/admin');
    }

    if (!isEditor(locals.user)) {
        throw redirect(302, '/');
    }

    return { user: locals.user };
}
```

16.1.2 16.2-16.5 Dashboard Implementation

```
<!-- src/routes/admin/+page.svelte -->
<script lang="ts">
    let { data } = $props();
</script>

<div class="grid grid-cols-1 md:grid-cols-4 gap-6 mb-8">
    <div class="stat bg-base-100 rounded-lg shadow">
        <div class="stat-titleتراسم ال"></div>
        <div class="stat-value text-primary">{data.stats.tracks}</div>
    </div>
    <div class="stat bg-base-100 rounded-lg shadow">
        <div class="stat-titleلسالسل ا"></div>
        <div class="stat-value text-secondary">{data.stats.series}</div>
```

```
    </div>
    <div class="stat bg-base-100 rounded-lg shadow">
        <div class="stat-titleسدروال"></div>
        <div class="stat-value text-accent">{data.stats.lessons}</div>
    </div>
    <div class="stat bg-base-100 rounded-lg shadow">
        <div class="stat-titleتاهويديفيال"></div>
        <div class="stat-value text-info">{data.stats.videos}</div>
    </div>
</div>

<!-- Quick Actions -->
<div class="grid grid-cols-2 md:grid-cols-4 gap-4">
    <a href="/admin/lessons" class="btn btn-outline"> سدروال</a>
    <a href="/admin/videos" class="btn btn-outline"> تاهويديفيال</a>
    <a href="/admin/series" class="btn btn-outline"> لسلسالا</a>
    <a href="/admin/tracks" class="btn btn-outline"> تاراسمال</a>
</div>
```

16.2 Chapter 17: Content Management

16.2.1 17.1-17.6 CRUD Operations

```
// src/routes/admin/lessons/[id]/+page.server.ts
export const actions: Actions = {
    update: async ({ request, params, locals }) => {
        const form = await request.formData();

        contentDatabase
            .update(lessons)
            .set({
                title: form.get('title') as string,
                description: form.get('description') as string,
                updatedAt: new Date().toISOString(),
                lastModifiedBy: locals.user.id.toString(),
            })
            .where(eq(lessons.id, params.id))
            .run();

        return { success: true };
    },

    toggleLock: async ({ params, locals }) => {
        if (locals.user.role !== 'admin') {
            return fail(403, { error: 'Admin only' });
        }

        const lesson = contentDatabase
            .select()
            .from(lessons)
            .where(eq(lessons.id, params.id))
            .get();

        contentDatabase
```

```
            .update(lessons)
            .set({ isLocked: !lesson.isLocked })
            .where(eq(lessons.id, params.id))
            .run();

        return { success: true };
    },
};
```

16.3 Chapter 18: Audit & Protection

16.3.1 18.1-18.5 Audit Trail

```
// All content tables have:
{
    createdAt: TEXT,          // When created
    updatedAt: TEXT,          // Last modified
    lastModifiedBy: TEXT,     // 'script' or user ID
    isLocked: INTEGER,        // Prevent changes
}

// Check before update:
function canModify(item, forceMode = false) {
    if (item.isLocked) return false;
    if (item.lastModifiedBy !== 'script' && !forceMode) return false;
    return true;
}
```

Chapter 17

Part VI: Content Pipeline

17.1 Chapter 19: Content Seeding

17.1.1 19.1-19.6 Seed Script

```
// scripts/seed-content.ts
async function seedLesson(lesson, forceMode) {
    const existing = contentDatabase
        .select()
        .from(lessons)
        .where(eq(lessons.id, lesson.id))
        .get();

    if (existing) {
        // Check if can update
        if (existing.isLocked) {
            console.log(`  Skipping locked: ${lesson.id}`);
            return;
        }
        if (existing.lastModifiedBy !== 'script' && !forceMode) {
            console.log(`  Skipping human edit: ${lesson.id}`);
            return;
        }

        // Update
        contentDatabase.update(lessons).set({...lesson}).run();
    } else {
        // Insert
        contentDatabase.insert(lessons).values({...lesson}).run();
    }
}
```

17.2 Chapter 20: Media Management

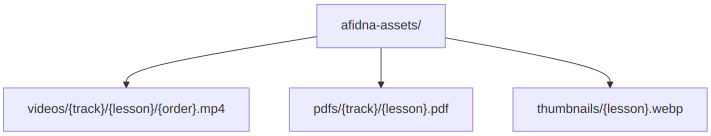

Figure 17.1: Cloudflare R2 File Structure

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

17.2.4 20.4 Database Integration

```
-- videos table stores direct URLs
downloadUrl = 'https://assets.afidna.com/videos/hadith/jibril/01.mp4'

-- lessons table for PDFs
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!

```
// In content.json
{
    "youtubeId": "abc123",        // للمشاهدة
    "downloadUrl": "https://..."  // للتحميل
}
```

17.2.6 20.6 Upload via rclone

```
# Configure rclone for R2
rclone config
# Provider: Cloudflare R2

# Upload
rclone sync ./processed/ r2:afidna-assets/
```

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

Only if you need browser uploads:

```
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';
import { getSignedUrl } from '@aws-sdk/s3-request-presigner';

export async function getUploadUrl(key: string) {
    const command = new PutObjectCommand({
        Bucket: 'afidna-assets',
        Key: key,
    });
    return getSignedUrl(R2, command, { expiresIn: 3600 });
}
```

Chapter 18

Part VII: Testing & Quality

18.1 Chapter 21: Testing Strategy

18.1.1 Unit Tests with Bun

```
// tests/auth.test.ts
import { describe, expect, test } from 'bun:test';
import { hashPassword, verifyPassword } from '$lib/server/auth';

describe('Auth', () => {
    test('password hashing', async () => {
        const password = 'test123';
        const hash = await hashPassword(password);

        expect(hash).not.toBe(password);
        expect(await verifyPassword(password, hash)).toBe(true);
        expect(await verifyPassword('wrong', hash)).toBe(false);
    });
});
```

18.1.2 Run Tests

```
bun test
```

18.2 Chapter 22: Code Quality

18.2.1 ESLint + Prettier

```
// .prettierrc
{
    "tabWidth": 4,
    "singleQuote": true,
    "trailingComma": "es5"
}
```

```
bun run format
bun run lint
```

Chapter 19

Part VIII: Deployment

19.1 Chapter 23-26: Production Deployment

19.1.1 Build

```
bun run build
```

19.1.2 VPS Deployment

```
# Ubuntu server
sudo apt update
curl -fsSL https://bun.sh/install | bash
git clone <repo>
cd afidna
bun install
bun run db:setup
bun run db:content
bun run build

# Run with systemd
sudo systemctl enable afidna
sudo systemctl start afidna
```

19.1.3 Caddy Config

```
afidna.com {
    reverse_proxy localhost:3000
}
```

Chapter 20

Part IX: Scaling & Optimization

20.1 Chapter 27-28: Performance

20.1.1 Database Optimization

```
// Use indexes
CREATE INDEX idx_lessons_slug ON lessons(slug);

// Use specific selects
db.select({ id, title }).from(lessons);

// Use limits
.limit(10)
```

20.1.2 Caching

```
// Cache headers
return new Response(body, {
    headers: {
        'Cache-Control': 'public, max-age=3600',
    },
});
```

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

```
# Development
bun run dev

# Database
bun run db:setup
bun run db:content
bun run seed

# Build
bun run build
bun run preview
```

21.3 Appendix C: Troubleshooting

Problem	Solution
Port in use	`kill -9 $(lsof -ti:5173)`
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.