The Writer’s Corner - Architecture Analysis for Like & Comment Implementation

Project Overview

Repository: https://github.com/vishesh-here/the-writers-corner
Tech Stack: Next.js 14, TypeScript, Prisma, PostgreSQL, NextAuth, Tailwind CSS
Architecture: Full-stack React application with API routes

Current Database Schema

Existing Models

```
User {
  id: String (cuid)
  name: String?
  firstName: String?
  lastName: String?
  email: String (unique)
  password: String?
  communityPosts: CommunityPost[]
  submissions: ExerciseSubmission[]
  // ... auth fields
}

CommunityPost {
  id: String (cuid)
  title: String
  content: String (Text)
  userId: String
  user: User
  exerciseId: String? (optional link to exercise)
  isPublic: Boolean (default: true)
  createdAt: DateTime
  updatedAt: DateTime
}

ExerciseSubmission {
  id: String (cuid)
  content: String (Text)
  userId: String
  exerciseId: String
  user: User
  exercise: Exercise
  isPublic: Boolean (default: false)
  // ... timestamps
}
```

Required Schema Extensions for Likes & Comments

```
// New models to add:

model PostLike {
  id        String   @id @default(cuid())
  userId    String
  postId    String   // References CommunityPost or ExerciseSubmission
  postType  PostType // Enum: COMMUNITY_POST | EXERCISE_SUBMISSION
  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  createdAt DateTime @default(now())

  @@unique([userId, postId, postType])
}

model PostComment {
  id        String   @id @default(cuid())
  content   String   @db.Text
  userId    String
  postId    String   // References CommunityPost or ExerciseSubmission
  postType  PostType // Enum: COMMUNITY_POST | EXERCISE_SUBMISSION
  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

enum PostType {
  COMMUNITY_POST
  EXERCISE_SUBMISSION
}

// Update User model to include:
model User {
  // ... existing fields
  likes    PostLike[]
  comments PostComment[]
}
```

Current Architecture Patterns

API Route Structure

- **Location**: `/app/api/`
- **Pattern**: RESTful endpoints using Next.js App Router
- **Authentication**: Server-side session validation using `getServerSession(authOptions)`
- **Database**: Prisma ORM with PostgreSQL
- **Error Handling**: Consistent JSON responses with status codes

Example API Pattern (from `/api/community/posts/route.ts`):

```
export async function GET() {
  try {
    const session = await getServerSession(authOptions)
    if (!session?.user?.id) {
      return NextResponse.json({ error: 'Unauthorized' }, { status: 401 })
    }

    const data = await prisma.model.findMany({
      include: { user: { select: { firstName: true, lastName: true, name: true } } },
      orderBy: { createdAt: 'desc' }
    })

    return NextResponse.json({ data })
  } catch (error) {
    console.error('Error:', error)
    return NextResponse.json({ error: 'Internal server error' }, { status: 500 })
  }
}
```

Frontend Component Patterns

State Management

- **React Hooks**: `useState`, `useEffect` for local state
- **Data Fetching**: Native `fetch()` API with async/await
- **No global state management** (Redux/Zustand not actively used for community features)

UI Components

- **Design System**: Custom vintage/literary theme with Tailwind CSS
- **Component Library**: Radix UI primitives with custom styling
- **Typography**:
- Headers: `font-typewriter` (Courier Prime)
- Body: `font-serif` (Crimson Text)
- **Color Palette**: Vintage literary theme
- `--ink`: Dark text color
- `--rust`: Accent color for interactions
- `--forest`: Secondary text
- `--sepia`: Background tones
- `--gold`: Highlights

Animation

- **Framer Motion**: Used for page transitions and component animations
- **Pattern**: `initial={{ opacity: 0, y: 30 }}` → `animate={{ opacity: 1, y: 0 }}`

Current Community Features

Data Flow

1. **Backend**: `/api/community/posts` fetches public `ExerciseSubmission`s
2. **Frontend**: `CommunityOverview` component displays posts with:
   - User information
   - Exercise context

- Content preview
- **Placeholder Like/Comment buttons** (non-functional)

Post Display Pattern

```
interface CommunityPost {
  id: string
  title: string
  content: string
  createdAt: string
  user: { firstName?: string; lastName?: string; name?: string }
  exercise?: { title: string; topic: { title: string; slug: string } }
}
```

Implementation Strategy for Likes & Comments

1. Database Migration

```
-- Add new tables
CREATE TABLE "PostLike" (
  "id" TEXT NOT NULL,
  "userId" TEXT NOT NULL,
  "postId" TEXT NOT NULL,
  "postType" "PostType" NOT NULL,
  "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT "PostLike_pkey" PRIMARY KEY ("id")
);

CREATE TABLE "PostComment" (
  "id" TEXT NOT NULL,
  "content" TEXT NOT NULL,
  "userId" TEXT NOT NULL,
  "postId" TEXT NOT NULL,
  "postType" "PostType" NOT NULL,
  "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  "updatedAt" TIMESTAMP(3) NOT NULL,
  CONSTRAINT "PostComment_pkey" PRIMARY KEY ("id")
);

CREATE TYPE "PostType" AS ENUM ('COMMUNITY_POST', 'EXERCISE_SUBMISSION');
```

2. API Endpoints to Create

Likes

- `POST /api/community/posts/[id]/like` - Toggle like
- `GET /api/community/posts/[id]/likes` - Get like count and user's like status

Comments

- `GET /api/community/posts/[id]/comments` - Get comments for post
- `POST /api/community/posts/[id]/comments` - Add comment
- `PUT /api/community/posts/[id]/comments/[commentId]` - Edit comment (own only)
- `DELETE /api/community/posts/[id]/comments/[commentId]` - Delete comment (own only)

3. Frontend Component Updates

Enhanced Post Display

```
interface EnhancedCommunityPost extends CommunityPost {
  _count: {
    likes: number
    comments: number
  }
  isLikedByUser: boolean
}
```

New Components Needed

- `LikeButton` - Toggle like with optimistic updates
- `CommentSection` - Display comments with pagination
- `CommentForm` - Add new comments
- `CommentItem` - Individual comment display with edit/delete

4. Design Consistency

Button Styling (following existing patterns)

```
// Like button (active state)
className="text-rust hover:text-rust/80 font-typewriter"

// Comment button
className="text-forest hover:text-rust font-typewriter"
```

Comment Styling

- Use existing `Card` components for comment containers
- Follow `font-serif` for comment content
- Use `Badge` for timestamps and user names
- Maintain vintage color scheme

5. Data Fetching Strategy

Optimistic Updates

- Like button: Update UI immediately, rollback on error
- Comments: Add to local state, sync with server

Pagination

- Comments: Load initial 10, “Load more” button
- Follow existing patterns from community posts

Key Considerations

Performance

- **Aggregation**: Use Prisma’s `_count` for like/comment counts
- **Caching**: Consider implementing for frequently accessed posts
- **Pagination**: Essential for comments on popular posts

Security

- **Authorization**: Users can only edit/delete their own comments
- **Validation**: Sanitize comment content, rate limiting
- **CSRF Protection**: Inherent with NextAuth session validation

User Experience

- **Real-time Feel**: Optimistic updates for likes
- **Accessibility**: Proper ARIA labels for interactive elements
- **Mobile Responsive**: Ensure touch-friendly interaction areas

Scalability

- **Database Indexes**: On `(userId, postId, postType)` for likes, `(postId, postType)` for comments
- **Soft Deletes**: Consider for comment moderation
- **Notification System**: Future enhancement for comment replies

File Structure for Implementation

```
app/
├── api/
│   └── community/
│       └── posts/
│           └── [id]/
│               ├── like/
│               │   └── route.ts
│               └── comments/
│                   ├── route.ts
│                   └── [commentId]/
│                       └── route.ts
├── components/
│   └── community/
│       ├── like-button.tsx
│       ├── comment-section.tsx
│       ├── comment-form.tsx
│       └── comment-item.tsx
└── lib/
    └── validations/
        └── comment.ts
```

This analysis provides a comprehensive foundation for implementing like and comment functionality while maintaining consistency with the existing codebase architecture and design patterns.