XR Motion Community Reel - Complete Developer Documentation

Project Overview

The **XR Motion Community Reel** is a modern, full-stack web application designed specifically for XR Motion meetups to showcase community video reels with integrated submission management.

Core Features

- **Public Projector Display**: Full-screen video showcase optimized for large displays
- **QR Code Submissions**: Mobile-friendly submission form accessible via QR code
- **Hidden Admin Panel**: Gesture-activated management interface
- **Drag-and-Drop Sorting**: Intuitive reel reordering
- **Real-time Updates**: Automatic data synchronization
- **Multi-platform Support**: YouTube, Vimeo, and direct video links

Technology Stack

Component	Technology	Version	Purpose
Framework	Next.js	15.0+	React framework with App Router
Frontend	React	19.0+	UI component library
Language	TypeScript	5.3+	Type-safe development
Styling	TailwindCSS	3.4+	Utility-first CSS framework
State	Zustand	4.5+	Lightweight state management
QR Codes	qrcode.react	3.1+	QR code generation
Storage	JSON	-	File-based database

Project Architecture

Directory Structure

```
xr-motion-reel/
├── src/
│   ├── app/                          # Next.js App Router
│   │   ├── layout.tsx                # Root layout
│   │   ├── page.tsx                  # Main projector view
│   │   ├── globals.css               # Global styles
│   │   ├── submit/
│   │   │   └── page.tsx              # Submission form page
```

```
│   │   └── api/
│   │       └── submissions/
│   │           └── route.ts        # REST API endpoints
│   ├── components/                   # React components
│   │   ├── ProjectorView.tsx       # Main display
│   │   ├── AdminPanel.tsx          # Admin UI
│   │   ├── SubmissionForm.tsx      # Form component
│   │   ├── ReelPlayer.tsx          # Video player
│   │   └── QRCodeDisplay.tsx       # QR generator
│   ├── lib/                         # Utilities
│   │   ├── utils.ts                # Helper functions
│   │   └── db.ts                   # Database operations
│   ├── store/                       # State management
│   │   └── submissionStore.ts      # Zustand store
│   ├── hooks/                       # Custom React hooks
│   │   ├── useKeyPress.ts          # Keyboard handling
│   │   ├── useTapDetection.ts      # Touch gestures
│   │   └── useSubmissions.ts       # Data fetching
│   └── types/                       # TypeScript types
│       └── index.ts                # Type definitions
├── data/
│   └── submissions.json             # Database file
├── public/
│   └── favicon.ico
└── Configuration files
```

Data Flow Architecture

1. **Submission Flow**
   - User scans QR code → Opens `/submit` page
   - Fills form → POST request to `/api/submissions`
   - Data saved to `submissions.json`
   - Status set to "pending"
2. **Display Flow**
   - ProjectorView fetches from `/api/submissions`
   - Filters for "accepted" status
   - Sorts by order field
   - Displays current reel with navigation
3. **Admin Flow**
   - Admin activates panel (X key / 10 taps)
   - Views pending submissions
   - Accepts/declines → PATCH to `/api/submissions`
   - Reorders via drag-and-drop → Bulk PATCH request
   - Changes reflect immediately

Installation Guide

Prerequisites

- Node.js 18.0 or higher
- npm, yarn, or pnpm package manager
- Text editor (VS Code recommended)
- Git (optional)

Step-by-Step Setup

1. Create Project Directory

```
mkdir xr-motion-reel
cd xr-motion-reel
```

2. Install Dependencies

```
npm init -y
npm install next@latest react@latest react-dom@latest
npm install zustand qrcode.react clsx date-fns
npm install -D typescript @types/node @types/react @types/react-dom
npm install -D tailwindcss postcss autoprefixer
npm install -D eslint eslint-config-next prettier
```

3. Initialize Configuration

```
npx tsc --init
npx tailwindcss init -p
```

4. Create Directory Structure

```
mkdir -p src/app/api/submissions
mkdir -p src/app/submit
mkdir -p src/components
mkdir -p src/lib
mkdir -p src/types
mkdir -p src/hooks
mkdir -p src/store
mkdir -p data
mkdir -p public
```

5. Copy Source Files

Copy all provided source code files to their respective directories as shown in the directory structure above.

6. Create Initial Data File

```
echo '{"submissions":[]}' &gt; data/submissions.json
```

7. Run Development Server

```
npm run dev
```

Navigate to `http://localhost:3000` in your browser.

Core Components

1. ProjectorView Component

Purpose: Main display for accepted reels during meetups

Features:

- Full-screen video playback
- Creator name and social handle display
- QR code for submissions
- Keyboard navigation (arrow keys)
- Auto-refresh every 30 seconds

Key Props: None (uses Zustand store)

Usage:

```
import ProjectorView from '@/components/ProjectorView';

&lt;ProjectorView /&gt;
```

2. AdminPanel Component

Purpose: Submission management interface

Features:

- View pending submissions
- Accept/decline actions
- Drag-and-drop reordering
- Visual feedback
- Real-time updates

Key Props: None (uses Zustand store)

Activation:

- Desktop: Press X key
- Mobile: Tap screen 10 times rapidly

3. SubmissionForm Component

Purpose: Mobile submission form

Features:

- Input validation
- URL format checking
- Success confirmation
- Error handling
- Responsive design

Fields:

- First Name (required)
- Last Name (required)
- Social Username (required)
- Reel URL (required, must be valid URL)

4. ReelPlayer Component

Purpose: Video playback wrapper

Features:

- YouTube embed support
- Vimeo embed support
- Direct video file support (.mp4, .webm, .ogg)
- Automatic format detection
- Responsive sizing

Props:

```
interface ReelPlayerProps {
  reel: Submission;
}
```

5. QRCodeDisplay Component

Purpose: QR code generation

Features:

- High error correction
- Customizable size
- SVG format
- Clean styling

Props:

```
interface QRCodeDisplayProps {
  url: string;
  size?: number; // default: 120
}
```

State Management

Zustand Store Structure

The application uses a single Zustand store for all state management:

```
interface SubmissionStore {
  // State
  submissions: Submission[];
  acceptedReels: Submission[];
  currentReelIndex: number;
  isAdminMode: boolean;
  isLoading: boolean;
  error: string | null;

  // Actions
  fetchSubmissions: () =&gt; Promise&lt;void&gt;;
  addSubmission: (data: SubmissionFormData) =&gt; Promise&lt;void&gt;;
  updateSubmissionStatus: (id: string, status: Status) =&gt; Promise&lt;void&gt;;
  reorderReels: (reels: Submission[]) =&gt; Promise&lt;void&gt;;
  setCurrentReelIndex: (index: number) =&gt; void;
  nextReel: () =&gt; void;
  previousReel: () =&gt; void;
  toggleAdminMode: () =&gt; void;
}
```

Usage Example

```
import { useSubmissionStore } from '@/store/submissionStore';

function MyComponent() {
  const { acceptedReels, nextReel } = useSubmissionStore();

  return (
    &lt;button onClick={nextReel}&gt;
      Next Reel ({acceptedReels.length} total)
    &lt;/button&gt;
  );
}
```

API Reference

GET /api/submissions

Description: Fetch all submissions

Response:

```
{
  "submissions": [
    {
      "id": "1234567890-abc123",
      "firstName": "John",
      "lastName": "Doe",
      "socialUsername": "@johndoe",
      "reelUrl": "https://youtube.com/watch?v=...",
      "status": "accepted",
      "order": 0,
      "submittedAt": "2025-10-28T12:00:00Z"
    }
  ]
}
```

POST /api/submissions

Description: Create new submission

Request Body:

```
{
  "firstName": "Jane",
  "lastName": "Smith",
  "socialUsername": "@janesmith",
  "reelUrl": "https://vimeo.com/123456789"
}
```

Response: 201 Created

```
{
  "submission": {
    "id": "generated-id",
    "firstName": "Jane",
    "lastName": "Smith",
    "socialUsername": "@janesmith",
    "reelUrl": "https://vimeo.com/123456789",
    "status": "pending",
    "order": 0,
    "submittedAt": "2025-10-28T12:00:00Z"
  }
}
```

PATCH /api/submissions

Description: Update submission or bulk reorder

Single Update Request:

```
{
  "id": "submission-id",
  "status": "accepted"
}
```

Bulk Reorder Request:

```
{
  "bulk": [
    { "id": "id1", "updates": { "order": 0 } },
    { "id": "id2", "updates": { "order": 1 } }
  ]
}
```

Response: 200 OK

DELETE /api/submissions

Description: Remove submission (soft delete to declined)

Query Parameters:

- `id` (required): Submission ID

Response: 200 OK

Keyboard Controls

Key	Action	Context
Right Arrow (→)	Next reel	Projector view
Left Arrow (←)	Previous reel	Projector view
X	Toggle admin panel	Any view
F11	Full screen	Browser (standard)

Mobile Gestures

Gesture	Action	Context
Tap 10 times	Toggle admin panel	Any view
Drag item	Reorder reels	Admin panel

Styling Guide

TailwindCSS Configuration

The project uses a custom TailwindCSS configuration with dark theme support:

Colors:

- `background`: #0a0a0a (deep black)
- `foreground`: #ededed (off-white)
- `primary`: #3b82f6 (blue)
- `secondary`: #6366f1 (indigo)

Animations:

- `fade-in`: 0.5s ease-in opacity transition
- `slide-in`: 0.3s ease-out slide from bottom

Usage Example:

```
<div>
  &lt;button className="bg-primary hover:bg-blue-600 px-4 py-2 rounded-lg"&gt;
    Submit
  &lt;/button&gt;
</div>
```

Deployment

Vercel (Recommended)

1. Install Vercel CLI:

```
npm install -g vercel
```

2. Deploy:

```
vercel
```

3. Follow prompts to link project and deploy

Netlify

1. Build project:

```
npm run build
```

2. Deploy via Netlify CLI or dashboard

Docker

1. Create Dockerfile:

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
EXPOSE 3000
CMD ["npm", "start"]
```

2. Build and run:

```
docker build -t xr-motion-reel .
docker run -p 3000:3000 xr-motion-reel
```

Testing Strategy

Unit Tests

Use Jest and React Testing Library:

```
import { render, screen } from '@testing-library/react';
import ProjectorView from '@/components/ProjectorView';

test('renders projector view', () =&gt; {
  render(&lt;ProjectorView /&gt;);
  expect(screen.getByText(/XR Motion/i)).toBeInTheDocument();
});
```

E2E Tests

Use Playwright for end-to-end testing:

```
import { test, expect } from '@playwright/test';

test('submission workflow', async ({ page }) =&gt; {
  await page.goto('http://localhost:3000/submit');
  await page.fill('#firstName', 'John');
  await page.fill('#lastName', 'Doe');
  await page.fill('#socialUsername', '@johndoe');
  await page.fill('#reelUrl', 'https://youtube.com/watch?v=test');
  await page.click('button[type="submit"]');
  await expect(page.locator('text=Submission Received')).toBeVisible();
});
```

Troubleshooting

Common Issues

Videos not playing:

- Verify URL format is correct
- Check browser console for CORS errors
- Test URL separately in browser
- Ensure video is publicly accessible

Admin mode won't activate:

- Clear browser cache
- Check console for JavaScript errors
- Ensure X key is pressed (not Shift+X)
- On mobile, tap rapidly in quick succession

Submissions not saving:

- Verify `data/submissions.json` exists
- Check file permissions (read/write)
- Ensure server has write access
- Check API endpoint responses in network tab

QR code not scanning:

- Increase QR code size (default: 120px)
- Improve lighting conditions
- Use high-quality camera
- Verify URL is correct

Drag-and-drop not working:

- Check browser compatibility (needs HTML5 support)
- Verify mouse/touch events are enabled
- Test in different browser
- Check console for errors

Performance Optimization

Best Practices

1. **Code Splitting**: Next.js automatically splits code by route
2. **Image Optimization**: Use Next.js Image component for thumbnails
3. **Lazy Loading**: Components load on demand
4. **Caching**: API responses cached for 30 seconds
5. **Bundle Size**: Keep dependencies minimal

Monitoring

- Monitor bundle size: `npm run build` shows size report
- Use Lighthouse for performance audits
- Track Core Web Vitals in production
- Set up error monitoring (Sentry recommended)

Security Considerations

Input Validation

- All form inputs validated client and server-side
- URL format checked before processing
- SQL injection not applicable (JSON storage)
- XSS prevention via React's auto-escaping

API Security

- Validate request bodies
- Check content types
- Rate limiting recommended for production
- CORS configured appropriately

Best Practices

- Keep dependencies updated
- Review security advisories regularly
- Use HTTPS in production
- Implement authentication for admin (future)
- Sanitize user inputs
- Validate URLs before embedding

Future Enhancements

Planned Features

1. **Database Migration**: Move from JSON to PostgreSQL/SQLite
2. **Authentication**: Admin login system with OAuth
3. **Real-time Updates**: WebSocket integration
4. **Analytics Dashboard**: Track submission statistics
5. **Multi-event Support**: Manage multiple meetup events
6. **Video Upload**: Direct video hosting
7. **Live Voting**: Audience engagement features
8. **Email Notifications**: Alert admins of new submissions

Contribution Guidelines

1. Fork the repository
2. Create feature branch
3. Follow existing code style
4. Add tests for new features
5. Update documentation
6. Submit pull request

Support and Resources

Documentation

- **Next.js**: https://nextjs.org/docs
- **React**: https://react.dev
- **TailwindCSS**: https://tailwindcss.com/docs
- **Zustand**: https://github.com/pmndrs/zustand

Community

- XR Motion community channels
- GitHub repository (if applicable)
- Discord server (if applicable)

Contact

For support or questions, contact XR Motion organizers

License

MIT License - Free to use and modify for personal and commercial projects

Credits

Developed for: XR Motion Community
Framework: Next.js 15
Built with: React 19, TypeScript, TailwindCSS
Maintained by: Community contributors

Version: 1.0.0
Last Updated: October 2025
Documentation Author: AI Assistant