Magic Chat - Technical Specifications

Overview

Magic Chat is a TikTok-style short-form video social platform built using Vertical Slice Architecture, combining Go backend services, Next.js frontend, and MongoDB for data persistence.

Architecture Philosophy

Vertical Slice Architecture

Each feature is organized as a complete vertical slice containing all layers (API, business logic, data access) rather than organizing by technical layer. This enables:

- Independent feature deployment
- Reduced coupling between features
- Easier testing and maintenance
- Team autonomy per feature

Tech Stack

Backend

- **Language**: Go 1.21+
- **Framework**: Gin or Echo for HTTP routing
- **Database**: MongoDB 6.0+
- **Caching**: Redis
- **Storage**: S3-compatible object storage (AWS S3, MinIO)
- **Video Processing**: FFmpeg

Frontend

- **Framework**: Next.js 14+ (App Router)
- **Language**: TypeScript
- **Styling**: Tailwind CSS
- **State Management**: Zustand or React Context
- **Video Player**: Video.js or custom HTML5 player

Core Features (Vertical Slices)

1. User Authentication Slice

Endpoints:

- `POST /api/auth/register`
- `POST /api/auth/login`
- `POST /api/auth/logout`
- `GET /api/auth/me`

Go Structure:

```
/slices/auth/
├── handler.go       # HTTP handlers
├── service.go       # Business logic
├── repository.go    # MongoDB operations
├── models.go        # Data structures
└── middleware.go    # JWT validation
```

MongoDB Collections:

- `users`: User profiles, credentials, metadata

Frontend Pages:

- `/login`
- `/signup`
- `/profile/[username]`

2. Video Upload Slice

Endpoints:

- `POST /api/videos/upload` (multipart form)
- `POST /api/videos/process` (webhook from video processor)
- `GET /api/videos/:id/status`

Go Structure:

```
/slices/video-upload/
├── handler.go
├── service.go
├── repository.go
├── models.go
├── processor.go     # Video transcoding coordination
└── storage.go       # S3 client wrapper
```

MongoDB Collections:

- `videos`: Video metadata, processing status, URLs

Frontend Components:

- Upload modal with drag-and-drop
- Progress indicator
- Video preview before posting

3. Video Feed Slice

Endpoints:

- `GET /api/feed/for-you` (personalized algorithm)
- `GET /api/feed/following`
- `GET /api/videos/:id`

Go Structure:

```
/slices/video-feed/
  ├── handler.go
  ├── service.go
  ├── repository.go
  ├── models.go
  └── algorithm.go     # Feed ranking logic
```

MongoDB Collections:

- `videos`: Video data with engagement metrics
- `user_interactions`: View history, watch time

Frontend Pages:

- `/` (For You feed)
- `/following`
- `/video/[id]`

4. Engagement Slice (Likes, Comments, Shares)

Endpoints:

- `POST /api/videos/:id/like`
- `DELETE /api/videos/:id/like`
- `POST /api/videos/:id/comments`
- `GET /api/videos/:id/comments`
- `POST /api/videos/:id/share`

Go Structure:

```
/slices/engagement/
├── handler.go
├── service.go
├── repository.go
└── models.go
```

MongoDB Collections:

- `likes`: User-video like relationships
- `comments`: Comment threads with replies
- `shares`: Share tracking

Frontend Components:

- Like/unlike button with animation
- Comment section with infinite scroll
- Share modal

5. User Following Slice

Endpoints:

- `POST /api/users/:id/follow`
- `DELETE /api/users/:id/follow`
- `GET /api/users/:id/followers`
- `GET /api/users/:id/following`

Go Structure:

```
/slices/following/
├── handler.go
├── service.go
├── repository.go
└── models.go
```

MongoDB Collections:

- `follows`: Follower-following relationships (denormalized)

Frontend Components:

- Follow/unfollow button
- Followers/following lists

6. Search & Discovery Slice

Endpoints:

- `GET /api/search?q=query&type=users|videos|hashtags`
- `GET /api/trending/hashtags`
- `GET /api/hashtags/:tag/videos`

Go Structure:

```
/slices/search/
  ├── handler.go
  ├── service.go
  ├── repository.go
  └── models.go
```

MongoDB Collections:

- `hashtags`: Trending hashtags with video counts
- `videos`: Indexed for text search

Frontend Pages:

- `/search`
- `/tag/[hashtag]`

7. Notifications Slice

Endpoints:

- `GET /api/notifications`
- `PUT /api/notifications/:id/read`
- `WS /api/notifications/stream` (WebSocket)

Go Structure:

```
/slices/notifications/
  ├── handler.go
  ├── service.go
  ├── repository.go
  ├── models.go
  └── websocket.go
```

MongoDB Collections:

- `notifications`: User notifications queue

Frontend Components:

- Notification bell with badge
- Notification panel

Data Models

User Document

json

```
{
  "_id": "ObjectId",
  "username": "string (unique)",
  "email": "string (unique)",
  "password_hash": "string",
  "display_name": "string",
  "bio": "string",
  "avatar_url": "string",
  "follower_count": "int",
  "following_count": "int",
  "video_count": "int",
  "total_likes": "int",
  "created_at": "timestamp",
  "updated_at": "timestamp"
}
```

Video Document

json

```
{
  "_id": "ObjectId",
  "user_id": "ObjectId",
  "title": "string",
  "description": "string",
  "video_url": "string",
  "thumbnail_url": "string",
  "duration": "int (seconds)",
  "hashtags": ["string"],
  "view_count": "int",
  "like_count": "int",
  "comment_count": "int",
  "share_count": "int",
  "processing_status": "enum (pending, processing, completed, failed)",
  "created_at": "timestamp",
  "updated_at": "timestamp"
}
```

Project Structure

Backend (Go)

```
magic-chat-backend/
├── cmd/
│   └── server/
│       └── main.go
├── slices/
│   ├── auth/
│   ├── video-upload/
│   ├── video-feed/
│   ├── engagement/
│   ├── following/
│   ├── search/
│   └── notifications/
├── pkg/
│   ├── config/
│   ├── database/
│   ├── cache/
│   ├── storage/
│   └── utils/
├── migrations/
├── go.mod
└── go.sum
```

Frontend (Next.js)

```
magic-chat-frontend/
├── app/
│   ├── (auth)/
│   │   ├── login/
│   │   └── signup/
│   ├── (feed)/
│   │   ├── page.tsx (For You)
│   │   └── following/
│   ├── video/[id]/
│   ├── profile/[username]/
│   ├── search/
│   └── layout.tsx
├── components/
│   ├── VideoPlayer/
│   ├── VideoCard/
│   ├── CommentSection/
│   └── shared/
├── lib/
│   ├── api/
│   ├── hooks/
│   └── utils/
├── public/
├── styles/
└── package.json
```

API Design Principles

1. **RESTful conventions** for resource operations
2. **JWT authentication** via Bearer tokens
3. **Pagination** using cursor-based approach
4. **Rate limiting** per user/IP
5. **Response format**:

json

```
{
  "success": true,
  "data": {},
  "error": null,
  "metadata": {
    "cursor": "string",
    "has_more": true
  }
}
```

Performance Considerations

1. **Video Processing**: Async job queue (Redis + Go workers)
2. **CDN**: CloudFront or Cloudflare for video delivery
3. **Caching**: Redis for feed data, user profiles
4. **Database Indexes**: On user_id, created_at, hashtags
5. **Pagination**: Limit 20-30 items per request
6. **WebSockets**: For real-time notifications and live metrics

Deployment Strategy

Development

- Docker Compose for local development
- Hot reload for both Go and Next.js

Production

- **Backend**: Kubernetes deployment, horizontal scaling
- **Frontend**: Vercel or self-hosted with Docker
- **Database**: MongoDB Atlas or self-managed replica set
- **Object Storage**: AWS S3 with CloudFront
- **CI/CD**: GitHub Actions

Security Measures

1. JWT token rotation and refresh tokens
2. Rate limiting per endpoint
3. Input validation and sanitization
4. CORS configuration
5. Helmet.js for Next.js security headers
6. MongoDB role-based access control
7. Video file validation (type, size, duration)
8. Content moderation queue (manual or AI-assisted)

Future Enhancements

- Live streaming capability
- Direct messaging
- Stories feature
- Monetization (creator fund, tips)
- Advanced analytics dashboard
- AI-powered content recommendations
- Multi-language support