

# Typeof null 架构设计文档



## 一、团队

### 1.1 团队介绍

Typeof null 由3位大三的开发同学组成。

我们合理分工,处理产品策划、界面设计以及前后端开发和部署测试等工作，团队成员均有丰富的项目开发经验。

我们的队名取自于 `JavaScript` 的一个历史遗留 bug

JavaScript

**`typeof null == 'object'`**

这个 bug 寓意着技术开发中的错误和复杂性，它提醒我们对待技术要谦逊和谨慎，对 coding 能力保持追求，并不断学习和改进。

## 1.2 团队分工

相对于我们的需求期望，比赛时间较为紧迫，需要确保团队的前期工作和管理得到充分的准备。以下是我们团队成员的分工和职责：

成员姓名	技术方向	团队职责
周想	前端	竞赛事项对接、前端部署、测试、ui设计、前端开发
童子欣	前端	团队事项管理、前端架构设计、产品设计、ui设计、前端开发
徐小帅	后端	后端架构设计、后端部署、产品设计、后端开发

## 1.3 团队工具

为保证产品、设计、开发等一系列的流程顺利进行，我们团队采用了以下工具来提高效率和开发体验：

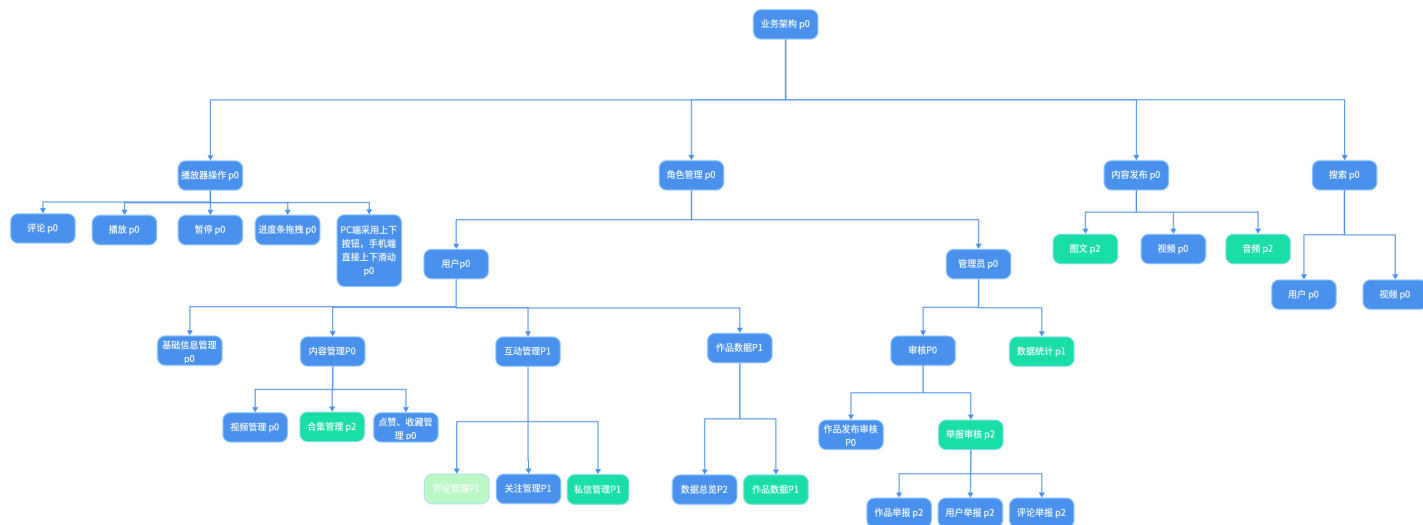
1. Boardmix：需求评审、项目管理工具，提供可视化的看板画板和任务管理功能。
2. Mastergo：设计工具，构建高质量的UI设计图。
3. Apifox：API设计和管理工具，设计、测试和管理API接口，包括定义API规范、生成API文档和模拟接口响应等。

## 1.4 团队规范

### 1.4.1 需求管理规范

为了明确我们的开发方向，我们对所列出的需求进行了多轮评审，评审等级分p0、p1、p2

- p0：核心要求，必须完成；
- p1：复杂需求、亮点需求，尽量要求完成；
- p2：额外需求，有余力则考虑完成；



第一次评审结果

## 1.4.2 todo规范

为保证按时且有序完成开发，我们从两个角度完成我们的 todo 规范：

1. 每日群公告 todo：每天晚上在微信群公告中归纳当日的完成进度，总结第二天的 todo，保证开发有序进行。

群公告  
代码仓库地址：<https://github.com/nonhana/Null-Video>

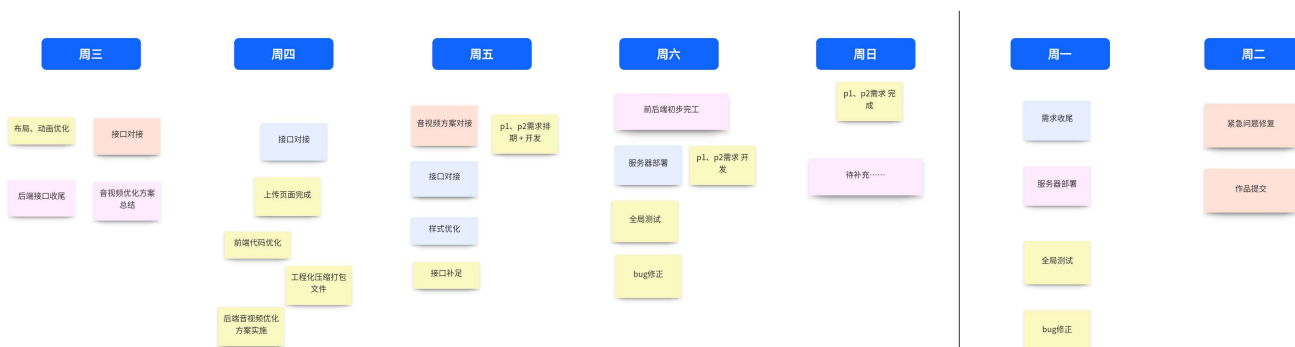
10.24 todolist:  
1.项目需求完善  
2.需求筛选 + 评审  
--- 评审分 p0、p1、p2  
--- p0：核心要求，必须完成；  
--- p1：较复杂需求但尽量要求完成；  
--- p2：额外需求，有余力则考虑完成  
3.前后端技术选型完善  
4.开发辅助工具确认（设计工具、api工具等）+ 开设团队项目  
5.项目仓库开设，向官方提交队伍信息  
6.设计风格确定  
7.工作排期

todolist:  
1.原型图搞定  
2.接口设计完毕  
3.数据库表设计完毕  
4.七牛云服务器获取  
5.前端开发：  
--- 配置 router  
--- 配置 pinia  
--- 配置 axios  
--- 完成 header 组件，app.ts  
6.后端开发：  
--- 相关基本配置  
--- 用户登录接口配置

todolist:  
1.ui设计：上传视频、登陆注册  
2.apifox 完善  
3.前端开发：  
--- 分支合并出 dev，同时创建 test、hotfix 分支  
--- 初步完善 header  
--- 封装基本组件：button、input 等  
--- axios 配置完善，依据接口文档完成接口配置  
--- 封装顶层 style 文件，存储 css 变量  
4.后端开发：  
--- 能开发多少接口是多少：用户->视频.....  
--- 其他优化（缓存之类的是否可以优化

todolist:  
1.童子欣：  
--- 登录、注册、个人中心样式优化  
--- 点赞、收藏、转发、评论 动画 + 响应  
--- 工程化——分包，gzip，dev 和 prod 区分  
2.周想：  
--- 调研音视频优化方案（拆分加载、预加载、懒加载），形成说明文档  
--- 优化方案实施  
--- 个人中心、登录、注册、上传、首页接口对接  
3.徐小帅：  
--- 后端部署  
--- 接口调整、测试  
--- 对未完成的 p1、p2 方案进一步筛选，引导群内讨论  
--- 音视频优化方案实施

2. 每周 todo：在 Boardmix 中根据当前的进度，实现每周排期，保证开发顺利达标。



## 1.4.3 开发规范

为保证开发时确保代码的一致性、可维护性和可扩展性，同时也便于团队协作和代码审查。以下我们团队在开发之前所约定的开发规范：

### 代码风格规范

- 命名规则：采用有意义的变量和函数命名方式，**小驼峰**对于变量和函数，**短横线命名**对于类名，**大驼峰**对于组件。
- 注释规则：为构建工具配置项和复杂的逻辑编写详细注释。单行注释用于解释复杂的代码段，多行注释用于模块和方法的描述。
- 代码格式：统一代码格式，通过 `vue-eslint` 最佳实践进行规范。

### 代码组织规范

- 模块化：采用模块化、组件化的开发方式，减少代码耦合，提高复用性。
- 文件结构：按照功能划分目录和文件，例如将相关的模型、服务和控制器放在同一目录下。

### 版本控制规范

- 分支策略：采用 `Git Flow` 分支策略，明确各个分支的用途和管理方式。
- 提交信息：要求提交信息包含必要信息，遵循 `Conventional Commits` 规范。例如，`feat(dev): add password reset feature`。

### 测试规范

- 探索性测试：非正式的软件测试技术，依靠自己的直觉、经验和产品思维进行测试。
- 冒烟测试：在项目初步版本上执行基础测试案例，验证关键功能是否能正常工作。

### 性能规范

- 代码审查：每次合并前进行 codeReview，确保代码质量和性能。
- 优化原则：关注循环优化、避免不必要的计算和大量内存分配，以及对关键路径的性能分析。

### 异常处理规范

- 错误设置：自定义异常类和错误码，便于快速定位出错内容
- 错误捕捉：确保适当捕捉和处理异常，在拦截器中抛出。
- 错误记录：用aop切面对controller中的请求进行拦截生成日志，方便后续问题追踪分析

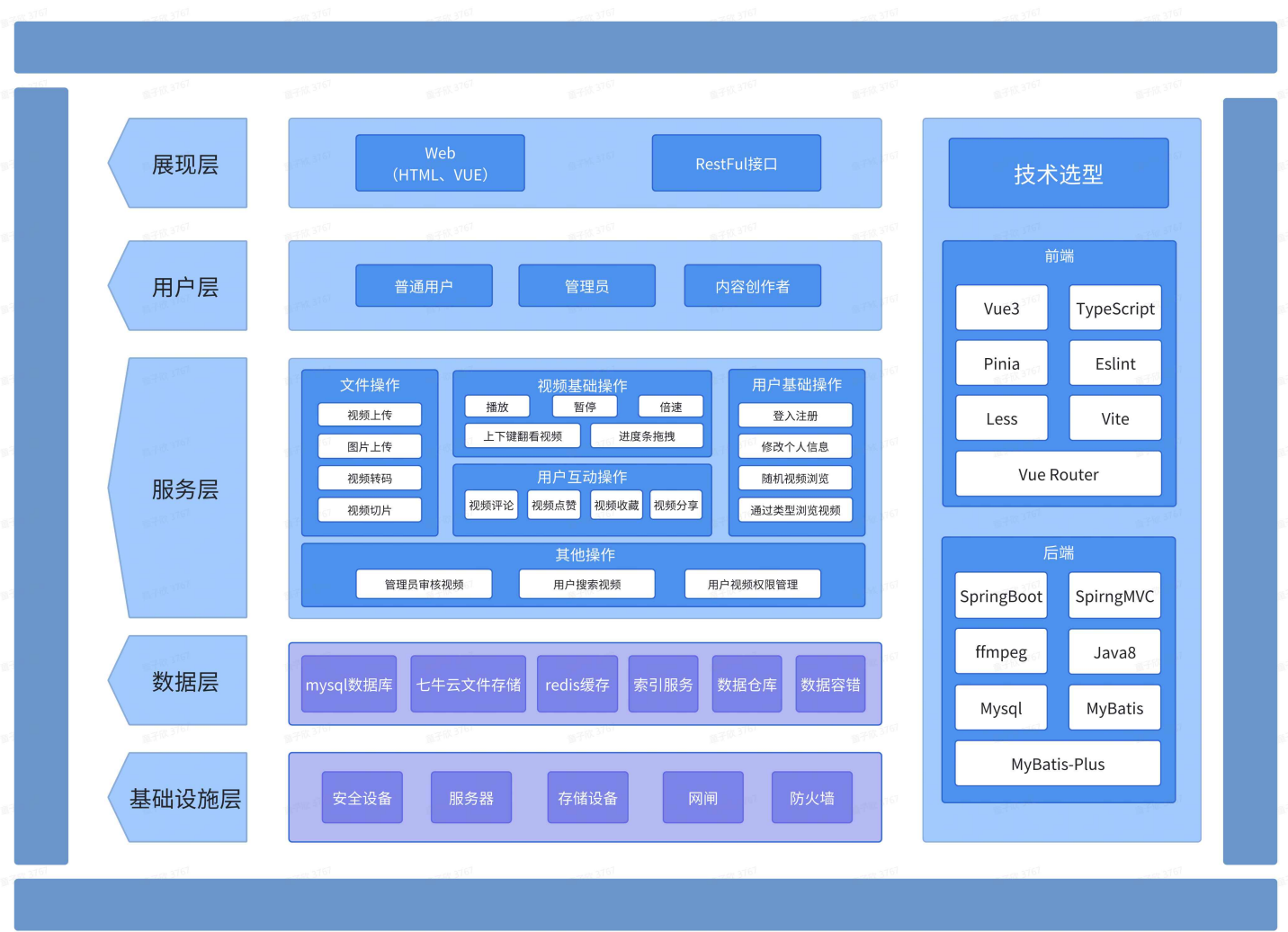
### 持续集成/持续部署 (CI/CD) 规范

- 自动化构建与测试：要求项目中的代码提交后，自动触发构建和测试过程，确保每次变更经过自动验证，降低错误引入。
- 持续交付与部署：要求将经过测试的代码能够自动部署到生产环境，减少人工介入，提高交付速度。
- 版本控制与文档：要求使用版本控制系统管理代码，并保持文档和注释的更新，以确保团队的共享和协作。

## 二、产品定位

## 2.1 产品思路

本产品是一个专注于短视频内容的Web应用，旨在为用户提供**快速、直观且互动性强**的视频消费体验。应用通过高效的云存储和视频处理技术，保障视频内容的**稳定、高效传输及优化的播放体验**。我们利用七牛云的存储解决方案和视频处理产品，实现短视频的快速上传、存储和流畅播放，以及提供基于内容的动态截帧功能，增强用户的浏览体验。



## 2.2 设计风格


我们的产品设计强调**现代感和操作便捷性**，采用以**卡片**为基本单元的设计，突出视觉上的清爽和易于用户直观操作的界面布局。

下面是我们在 mastergo 上约定的**设计规范**和一部分**设计稿**。

背景色: #fff

主色  卡片基本色

稍暗色调  按钮hover色等  暗色调 

特色色调 (网页的鲜明色) 

圆角 16px = 1rem

外阴影: box-shadow: 0px 4px 10px 0px rgba(0, 0, 0, 0.3);

内阴影: box-shadow: inset 0px 4px 10px 0px rgba(0, 0, 0, 0.3);

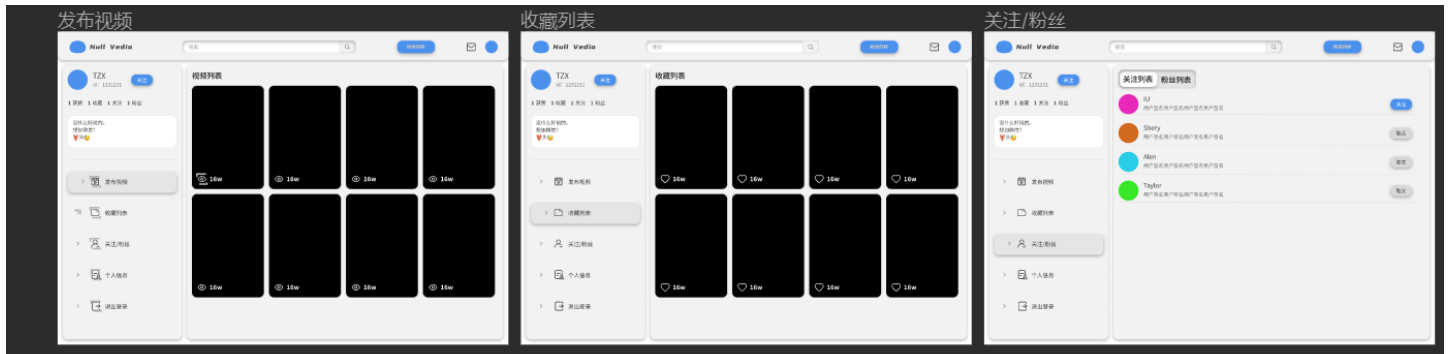
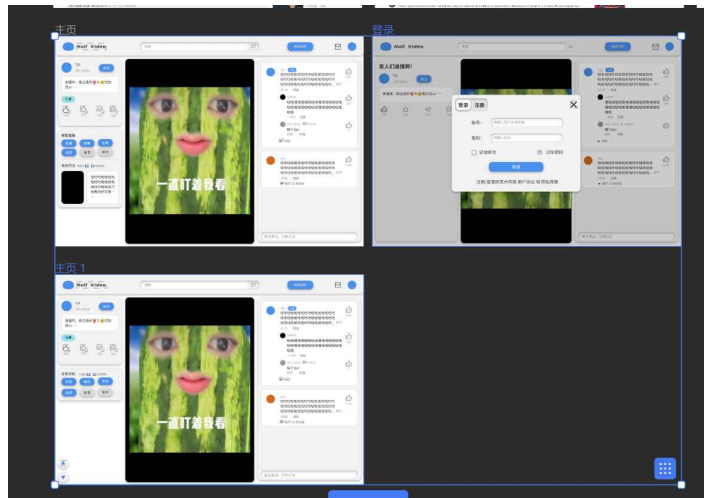
页面新建容器大小: 长1440px, 宽1000px

布局之间各个组件间隔 16px = 1rem

文字用色: 基本用#3d3d3d, 一些说明、附加注释之类的可以用#6d757a

hover渐变: 无特殊说明, 一律采用0.3s渐变

icon填充色: 一律用#3d3d3d



## 2.3 产品功能

基于Web端的特性, 产品功能集中于实现用户观看短视频的核心需求。

基础功能:

- 视频播放: 实现基本的**播放、暂停**功能, 进度条可以拖拽。
- 内容分类: 设置多种视频分类, 方便用户按照兴趣浏览视频。
- 视频切换: 支持键盘上下键**快速切换**视频, 为用户提供流畅的浏览体验。
- 视频发布: 用户可以按照要求发布想要展示的视频。

高级功能:

- 账户系统: 允许用户注册和登录, 可**收藏**喜欢的视频。
- 社交互动: 用户可以对视频进行**点赞、分享、评论**, 以及**关注**内容创作者。
- 搜索功能: 用户可以搜索视频, 快速找到想看的内容。

## 2.4 产品细节设计

在细节设计上, 产品专注于**提供流畅和高效的用户体验**。

- 优化的视频加载: 结合**七牛云技术**, 设置**合适的编码参数加快传输速度**, 减少用户等待时间。
- 清晰的内容分类展示: 使用直观的图标和标签清楚地标示视频内容的类别, 使用户可以轻松地**识别和切换**。

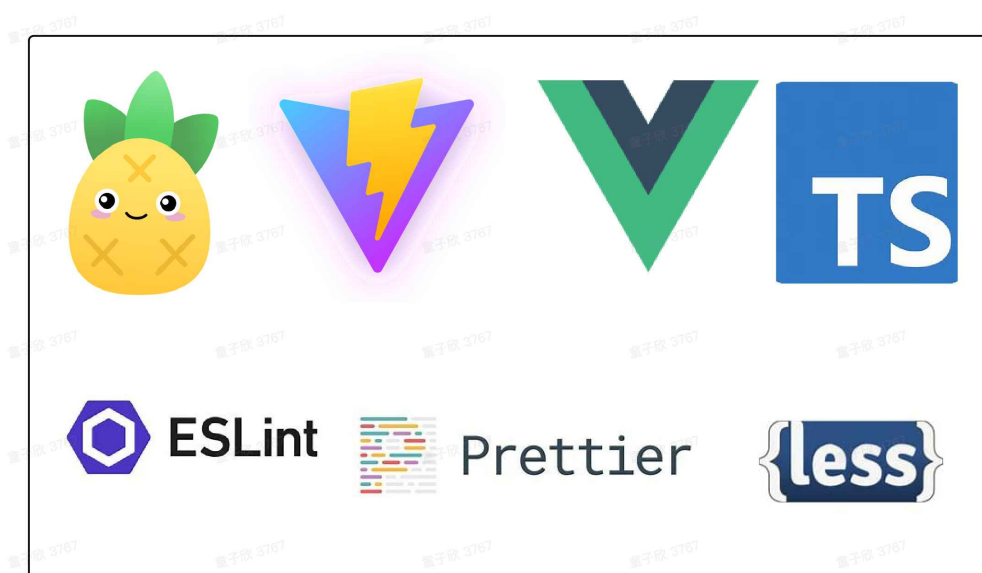
- 自适应播放器控件：播放器控件将根据用户设备和屏幕大小**自适应调整**，优化观看体验。
- 强提示动画：在很多交互中给予用户**强提示**，提高使用体验。
- 界面友好提示：所有操作反馈均采用温馨而不打扰的提示方式，如使用**浮动提示**而不是强制弹窗来优化体验。
- 安全稳定的账户管理：确保用户数据安全，提供稳定的登录和注册体验。

## 三、前端架构

### 3.1 技术选型

在技术栈的选择上，我们决定采用以下技术与工具：

- 框架与库：Vue 3 + TypeScript，结合 Composition API 提供更优的**代码组织和复用**。
- 状态管理：Pinia，轻量的、更适合 Vue 3 的状态管理解决方案。
- 样式处理：Less，为CSS预处理，提供变量、嵌套等强大功能。
- 构建工具：Vite，提供极速的热更新和构建性能。
- 路由管理：Vue Router，管理和优化页面的路由跳转和状态。
- 代码规范：ESLint + Prettier，统一代码风格，提升代码质量。



### 3.2 工程化方案

在工程化方向，我们团队做了深入的思考，决定将其作为一个**项目亮点**仔细打磨。

我们放弃了开箱即用的配置方案，希望通过一些工程化方案来提高我们的开发效率，解决打包体积、首屏加载等性能问题。

对此相关负责的同学做了大量的配置

#### 3.2.1 代码分割

通过配置 `manualChunks` 选项，实现自定义的代码分割：

```
1 // vite.config.ts
2 build: {
3   // rollup 配置
4   rollupOptions: {
5     output: {
6       // key自定义
7       // value[] 插件名 (package.json中的名称) 或 src/相对路径下的指定文件
8       manualChunks: {
9         // vue vue-router合并打包
10        vue: ['vue', 'vue-router'],
11        lodash: ['lodash'],
12        // 合并两文件为 index
13        index: ['src/components/index1.vue', 'src/components/index2.vue']
14        ...
15      }
16    }
17  }
18 }
```

### 3.2.2 gzip 压缩

使用 `vite-plugin-compress` 插件对文件进行 gzip 压缩，减小构建包体积，与此同时需要部署工具配置支持 gzip 文件。

### 3.2.3 图片压缩

使用 `vite-plugin-imagemin` 插件压缩图片，减小构建包体积。

### 3.2.4 CDN 引入

Vite 会将外部库默认打包，我们可以采用CDN的方式引入这些外部库。

首先安装 `rollup-plugin-external-globals` 插件，用于将外部全局变量作为模块导入。它可以帮助我们在打包过程中将全局变量作为外部依赖引入，而不是将其打包到最终的输出文件中。

### 3.2.5 动态导入

1. 通过动态导入实现路由懒加载
2. 通过异步导入视频、组件

### 3.2.6 别名路径

为方便引用组件、配置项，我们按照模块功能配置了部分的 alias，并在 tsconfig 中注册路径。



```

1 // vite.config.ts
2 alias: {
3   '@': path.resolve(__dirname, './src'), // 设置`@`指向`src`目录
4   '@nullVideo': path.resolve(__dirname, './src/components'), // 通用组件
5   '@nullSvg': path.resolve(__dirname, './src/assets/svg') // svg
6 }

```

```

1 // tsconfig.json
2 "baseUrl": "./",
3   "paths": {
4     "@/*": ["src/*"],
5     "@nullVideo/*": ["src/components/*"],
6     "@nullSvg/*": ["src/assets/svg/*"]
7   },

```

### 3.2.7 全局less变量注册

为**最大程度解耦css参数**，我们将经过设计规范确定的**常用属性定义别名**，并通过 vite **全局导出**，使得项目中任何组件皆可通过变量定义样式。

```

1 // vite.config.ts
2 css: {
3   postcss: {
4     plugins: [
5       postCssPxToRem({
6         // px转rem插件配置
7         rootValue: 16,
8         propList: ['*']
9       })
10    ]
11  },
12  preprocessorOptions: {
13    less: {
14      charset: false,
15      additionalData: `@import '@/variables.less';` // 全局引入变量文件
16    }
17  }
18 },

```

```

1 // variables.less
2 // background color

```

```

3 @bg-color: #f2f2f2;
4 @bg-color-primary: #4a91ee;
5 @bg-color-secondary: #e5e5e5;
6 @bg-color-third: #d8d8d8;
7
8 // text
9 // 外部引入字体 优设标题黑
10 @font-face {
11   font-family: 'YouSheBiaoTiHei';
12   src: url('./assets/fonts/YouSheBiaoTiHei-2.ttf') format('truetype');
13 }
14 @YouSheBiaoTiHei: 'YouSheBiaoTiHei';
15 // 字体颜色
16 @text: #3d3d3d;
17 @text-secondary: #6d757a;
18
19 // px/rem
20 @border-radius: 1rem;
21
22 // shadow
23 @shadow-outer: 0px 4px 10px 0px rgba(0, 0, 0, 0.3);
24 @shadow-inner: inset 0px 4px 10px 0px rgba(0, 0, 0, 0.3);
25

```

### 3.2.8 自动导入组件

为了减少组件、常用函数引入带来不必要的麻烦，我们配置了自动导入，将通过标签名自动识别组件。

```

1   AutoImport({
2     // 自动导入Vue等常用函数
3     imports: [
4       'vue',
5       {
6         'naive-ui': [
7           'useDialog',
8           'useMessage',
9           'useNotification',
10          'useLoadingBar'
11        ]
12      }
13    ],
14  }),
15  Components({
16    // 自动导入组件

```

```
17     resolvers: [NaiveUiResolver()]
18   }),
```

### 3.2.8 其他优化配置

```
1 build: {
2   chunkSizeWarningLimit: 2000, // chunk 超过 2000kb 之后进行提示
3   cssCodeSplit: true, //css 拆分
4   sourcemap: false, //不生成sourcemap
5   assetsInlineLimit: 5000, //小于该值 图片将打包成Base64
6 },
```

## 3.3 音视频方案

由于该项目主要采用 video.js 作为视频加载的主要插件，因此主要优化方案是围绕 video.js 的插件生态出发。

### 3.3.1 video.js 提高视频加载体验插件

1. **videojs-contrib-hls** 和 **videojs-contrib-dash**：这两个插件支持HLS和MPEG-DASH自适应流。它们能够动态地根据用户的网络质量选择合适的视频质量，从而提供更流畅的播放体验。

HLS（HTTP Live Streaming）和MPEG-DASH（Dynamic Adaptive Streaming over HTTP）都是自适应比特率流技术，它们的主要目标是为用户提供最佳的视频播放体验，无论网络环境如何。

#### 1. HLS（HTTP Live Streaming）

**定义：**HLS是由Apple公司开发的一种流媒体协议。它会将视频文件切分为多个小的TS（Transport Stream）文件，并且为不同的视频质量（如高、中、低质量）生成多个TS文件系列。

**播放流程：**当用户开始播放视频时，HLS的客户端会首先下载一个包含所有TS文件URL的M3U8文件。根据用户的网络条件，它会选择合适的TS文件进行下载和播放。如果网络条件改变（比如从Wi-Fi切换到移动数据），HLS可以在不同质量的TS文件之间无缝切换，确保视频播放不被中断。

#### 2. MPEG-DASH（Dynamic Adaptive Streaming over HTTP）

**定义：**DASH是一个开放标准的流媒体协议，与HLS的工作原理类似，但它不受任何公司的控制。

**播放流程：**与HLS相似，DASH也使用了一个描述文件（这里是MPD, Media Presentation Description）来列出所有可用的视频片段和质量级别。然后，客户端会基于网络条件动态选择合适的片段进行下载和播放。

**需要前端进行配置吗？** 前端确实需要进行配置，但这主要是为了让视频播放器支持HLS和DASH。大多数现代的视频播放器，如 `video.js`，可以通过插件或扩展来支持这两种格式。一旦你配置

好了播放器，它就可以自动处理后端返回的HLS或DASH视频URL，并为用户提供自适应比特率流体验。

例如，要让 `video.js` 支持HLS，可以使用 `videojs-contrib-hls` 插件；要支持DASH，可以使用 `videojs-contrib-dash` 插件。

2. **videojs-resolution-switcher**：这个插件允许用户手动选择所需的视频质量，比如从高清切换到标清，以应对网络速度的变化。
3. **videojs-thumbnails**：此插件为进度条提供缩略图预览，这样即使在缓冲时，用户也能有个直观的内容预览。
4. **videojs-buffer-wait**：这是一个视频缓冲指示器插件，可以在视频缓冲期间显示一个旋转的加载指示器。
5. **videojs-seek-buttons**：这个插件为播放器提供前进和后退按钮，允许用户跳过某些部分，特别是在视频缓冲时。
6. **videojs-vhs-save**：这个插件可以缓存HLS视频片段，使得在断网或低网速情况下，视频可以继续播放一段时间。
7. **videojs-errors**：此插件提供更详细、友好的错误信息，以帮助用户了解为什么视频无法加载或播放。

### 3.3.2 其他角度优化方案

1. **内容分发网络（CDN）**：使用CDN可以大大加速全球各地用户的访问速度，因为CDN会将视频内容缓存到多个地理位置的服务器上。
2. **自适应比特率流（ABR）**：如HLS或DASH，可以根据用户的网络条件动态调整视频质量。
3. **视频编码优化**：使用更先进的编码格式如H.265（而不是H.264）可以在保持相同的视频质量的同时，降低文件大小。同时，确保你的视频有多个比特率版本，以适应各种网络条件。
4. **懒加载**：在页面加载时只加载视频的元数据，而不是整个视频文件。只有当用户点击播放按钮时才开始加载视频内容。
5. **预加载**：预先加载页面上即将播放的视频或用户可能感兴趣的视频，但要注意不要过度预加载，避免浪费用户的数据。
6. **浏览器缓存策略**：合理配置服务器的HTTP缓存头，确保已加载的视频内容在一段时间内被缓存，从而加快再次访问时的加载速度。
7. **优化数据库查询**：如果你的应用依赖于数据库来获取视频URL或其他元数据，确保查询是高效的，考虑使用缓存来减少数据库的负担。
8. **使用视频封面**：展示一个视频封面图像，直到视频准备好播放，这可以提供一个更流畅的用户体验，因为用户不会看到一个空白的视频播放器。
9. **反馈机制**：提供一个加载指示器或进度条，让用户知道视频正在加载，并在视频无法加载时提供有用的错误信息。

## 3.4 组件库与组件封装方案

### 组件库封装方案

我们借助了 `native-ui` 作为基本组件库，为了使组件库的样式贴近设计风格，我们通过两种方式实现了个性化适配：

1. 通过 `:deep()` 穿透样式，实现局部修改。
2. 通过全局配置，修改 `native-ui` 内部样式，实现全局修改。

### 组件封装方案

我们将 Button、Input、Card 等一系列基础组件进行封装，减少了代码耦合，提高开发效率。

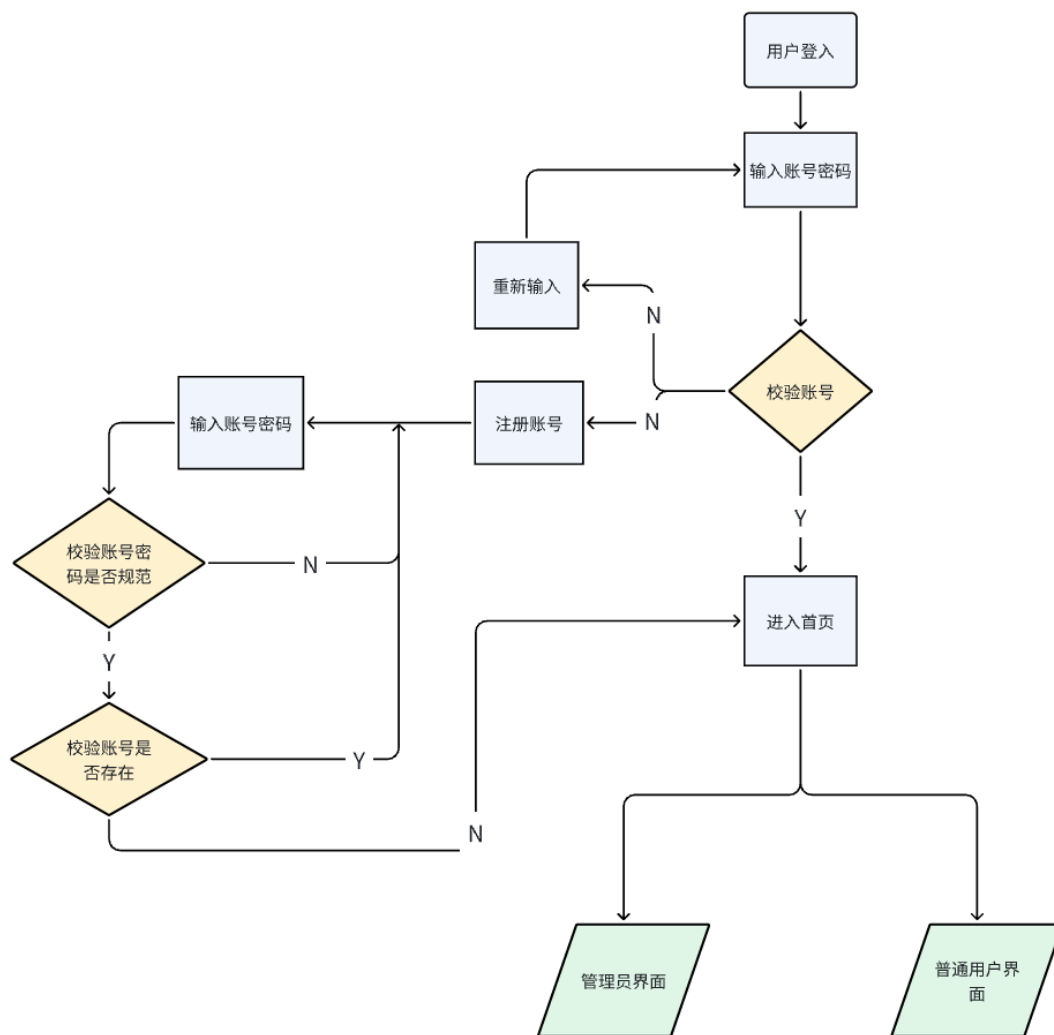
## 四、 后端架构

### 4.1 技术选型

- 开发框架：SpringBoot+SpringMVC。两者结合简化了Web应用开发，可以快速配置、开发和部署。
- 数据库：Mysql。轻量级数据库，在小规模开发的情况下可以降低成本。
- ORM框架：MyBatis-plus，Mybatis。结合使用可以简化数据库操作、减少 SQL 编写工作、提高开发效率。
- 缓存：Redis。支持多种数据结构的内存数据库和缓存系统
- 视频处理：FFmpeg。提供多种视频处理方式，可以高效操作。
- 代码风格：RestFulApi。使接口更加清晰和易于理解，提高了可读性和可维护性。

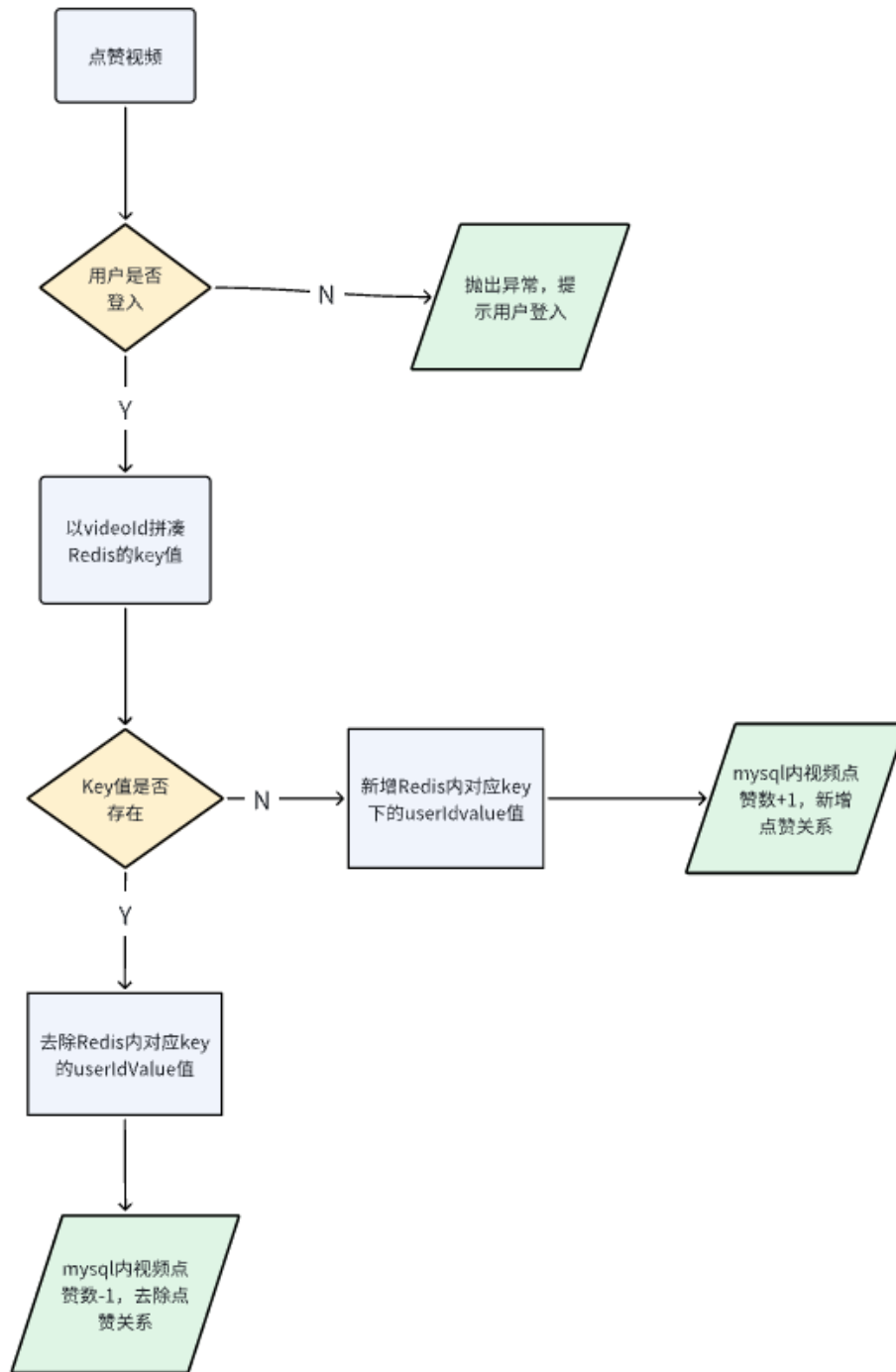
### 4.2 功能流程图

#### 4.2.1 登入注册功能流程图



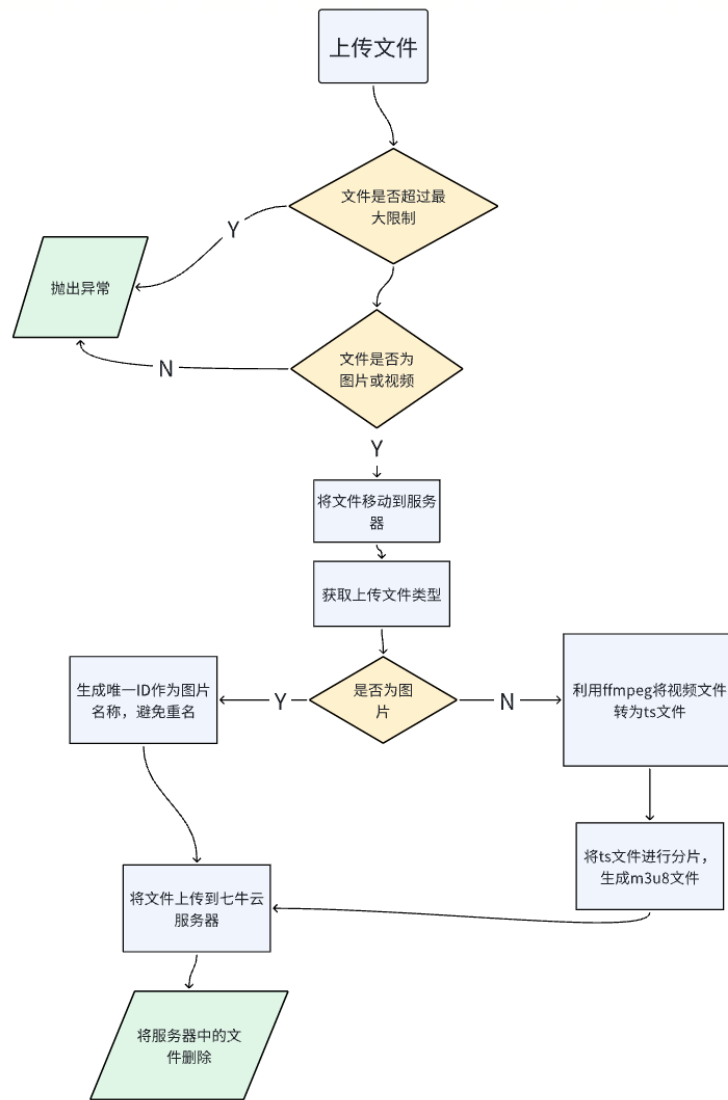
- 用户在注册时会检验账号长度和密码是否在指定范围内
- 对用户密码采用MD5盐值加密，加强安全性
- 登入后会返回userRole，来判断属于哪种角色

#### 4.2.2 点赞功能流程图



- 用户点赞时会对登入状态进行判断，未登入会提示状态异常

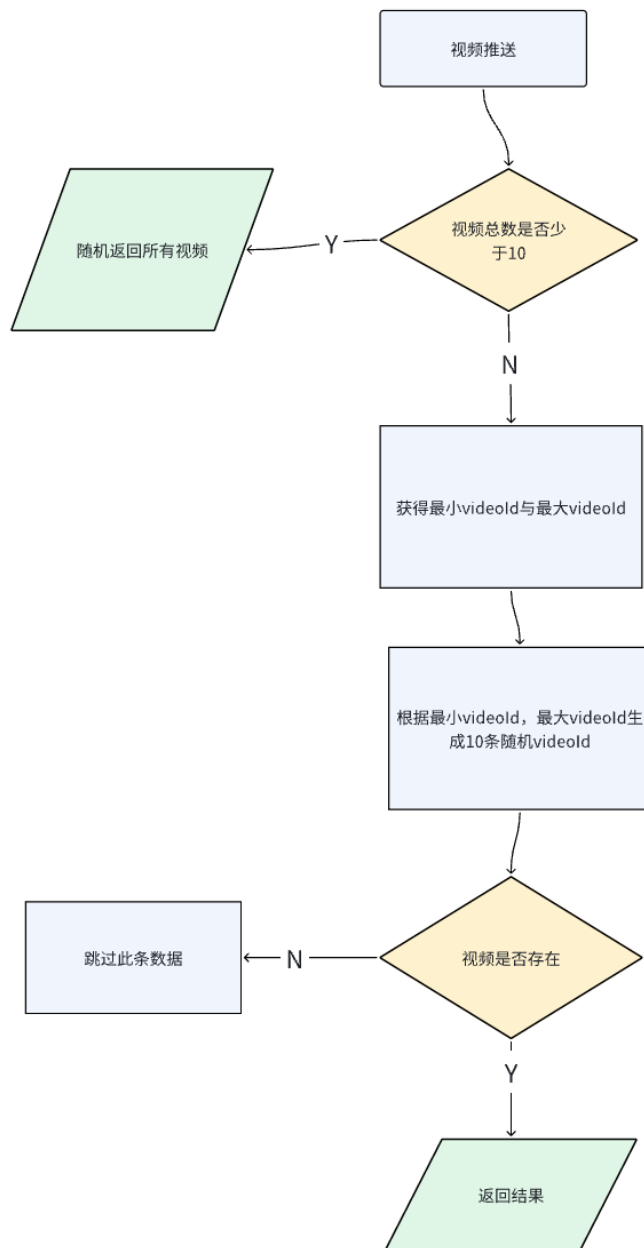
### 4.2.3 文件上传功能流程图



- 在对视频进行分片前先将其转为ts类型，可以加快切片速度
- 上传文件前对文件类型进行判断，避免后续文件错误

#### 4.2.4 主页视频推送功能流程图





- 通过获得随机videoid来取随机视频，可以提高查询效率

## 4.3 数据库设计

### 4.3.1 用户表

字段名	描述	数据类型	数据长度	主键
id	用户id	bigint		Y
user_account	用户账号	varchar	20	N
user_password	用户密码	varchar	255	N
user_name	用户昵称	varchar	255	N

email	用户邮箱	varchar	40	N
user_role	用户角色	tinyint		N
create_time	创建时间	datetime		N
update_time	修改时间	datetime		N
is_delete	逻辑删除	tinyint		N

4.3.2 用户信息表

字段名	描述	数据类型	数据长度	主键
id	用户信息id	bigint		Y
user_id	用户id	bigint		N
phone	用户手机	varchar	15	N
gender	用户性别	tinyint		N
age	用户年龄	int		N
user_avatar	用户头像url	text		N
user_profile	用户简介	varchar	1024	N
follower	粉丝数	int		N
following	关注数	int		N
create_time	创建时间	datetime		N
update_time	修改时间	datetime		N

4.3.3用户关注表

字段名	描述	数据类型	数据长度	主键
id	关注表id	bigint		Y
follower_id	关注者用户id	bigint		N
following_id	被关注者用户id	bigint		N
create_time	创建时间	datetime		N

update_time	修改时间	datetime		N
-------------	------	----------	--	---

### 4.3.4 用户视频文件夹表

字段名	描述	数据类型	数据长度	主键
id	视频文件夹id	bigint		Y
user_id	用户id	bigint		N
video_folder_name	文件夹名称	varchar	50	N
video_folder_role	文件夹权限	tinyint		N
is_main	是否为主文件夹	tinyint		N
create_time	创建时间	datetime		N
update_time	修改时间	datetime		N

### 4.3.5 视频表

字段名	描述	数据类型	数据长度	主键
id	视频id	bigint		Y
user_id	用户id	bigint		N
video_cover_url	视频封面url	varchar	1024	N
video_url	视频url	varchar	1024	N
video_description	视频描述	varchar	1024	N
video_type	视频类型	int		N
video_favour_num	视频收藏数	int		N
video_share_num	视频分享数	int		N
video_thumb_num	视频点赞数	int		N
video_play_num	视频播放量	bigint		N
video_status	视频状态	tinyint		N
video_role	视频权限	tinyint		N

create_time	创建时间	datetime		N
update_time	修改时间	datetime		N

4.3.6 视频评论表

字段名	描述	数据类型	数据长度	主键
id	视频评论id	bigint		Y
video_id	视频id	bigint		N
user_id	用户id	bigint		N
parent_id	一级评论的id	bigint		N
answer_id	要回答的评论id	bigint		N
video_comment_cotent	评论内容	varchar	255	N
video_comment_status	评论状态	tinyint		N
video_comment_thumb	评论点赞数	int		N
create_time	创建时间	datetime		N
update_time	修改时间	datetime		N

4.3.7 视频收藏表

字段名	描述	数据类型	数据长度	主键
id	视频收藏id	bigint		Y
video_folder_id	收藏文件夹id	bigint		N
video_id	视频id	bigint		N
user_id	用户id	bigint		N
create_time	创建时间	datetime		N
update_time	修改时间	datetime		N

4.3.8 视频点赞表

字段名	描述	数据类型	数据长度	主键
id	视频点赞id	bigint		Y
video_id	视频id	bigint		N
user_id	用户id	bigint		N
create_time	创建时间	datetime		N
update_time	修改时间	datetime		N

4.3.9 视频标签表

字段名	描述	数据类型	数据长度	主键
id	视频标签id	bigint		Y
video_tag_name	视频标签名	varchar	255	N
create_time	创建时间	datetime		N
update_time	修改时间	datetime		N

4.3.10视频标签关系表

字段名	描述	数据类型	数据长度	主键
id	视频标签关系id	bigint		Y
video_id	视频id	bigint		N
video_tag_id	视频标签id	bigint		N
create_time	创建时间	datetime		N
update_time	修改时间	datetime		N

4.3.11视频类型表

字段名	描述	数据类型	数据长度	主键
id	视频类型id	bigint		Y
video_type_name	视频类型名称	varchar	50	N

video_topic	视频大类下的主题名称	varchar	100	N
video_topic_description	主题描述	tinyint		N
is_parent	是否为视频大类	tinyint		N
parent_id	视频大类id	bigint		N
create_time	创建时间	datetime		N
update_time	修改时间	datetime		N

## 4.4 API设计

### 4.4.1 用户层接口设计

	接口名	作用	传入参数	传出参数
UserController	userRegiser	用户注册	用户账号密码，确认密码	用户登入状态
	userLoginByAccount	用户登入，获取token	用户账号密码	token
	getUserInfoByToken	获取用户信息	用户id	获取用户基础信息状态
	userLogout	用户登出	空	登出状态
	userUpdateInfo	修改用户信息	用户id，姓名等需要更改的信息	用户信息修改状态
	userFollow	关注用户	用户id，要关注的用户id	用户关注状态
	showFollowVO	查看用户关注和粉丝	用户id，查询选项	用户查询关注和粉丝状态
	userRemoveFollower	移除粉丝	用户id，要移除粉丝的id	用户移除粉丝状态

### 4.4.2 视频层接口设计

	接口名	作用	传入参数	传出参数
--	-----	----	------	------

VideoController	addVideo	用户发布视频	与视频信息有关参数	发布视频状态
	getVideoShowOnPage	获取主页视图下的视频	用户id与创作者id	查询主页视频状态
	getFavourOrThumbVideo	获取点赞或收藏的视频	用户id，查询选项	查询点赞收藏视频状态
	getVideoShowOnMine	获取自己管理视频下的视频	用户id，开始页数	查询个人视频状态
	updateVideoInfo	修改视频信息	与视频描述等有关参数	视频信息修改状态
	getVideoContent	点击进入某个具体视频	视频id，用户id	查看视频具体内容状态
	removeVideo	删除视频	用户id，视频id	删除视频状态
	getVideoTag	获取视频标签	搜索tag关键词	查询视频标签状态
	addVideoTag	添加视频标签	视频标签内容	添加视频标签状态
	getVideoType	获取视频类型	空	查询视频类型状态
	userThumbVideo	点赞视频	用户id，视频id	点赞视频状态
	favourVideo	收藏视频	用户id，视频id	收藏视频状态
	commentVideo	添加评论	与评论有关参数	添加评论状态
	getVideoComment	查看评论	视频id，用户id	查看评论状态
	thumbComment	点赞评论	评论id，用户id	点赞评论状态
	removeVideoComment	删除评论	评论id，用户id	删除评论状态

	getRandomVideo	随机获取分类视频用于首页播放	视频类型id，用户id	查看视频状态
	shareVideo	分享视频	视频id和用户id	分享视频状态

### 4.4.3 文件处理接口设计

	接口名	作用	传入参数	传出参数
FileController	fileUpload	上传文件	文件	文件上传状态

### 4.4.4 管理员接口设计

	接口名	作用	传入参数	传出参数
AdminController	getAuditVideo	查看待审核的视频	用户id，开始页数	查看审核视频状态
	adminUpdateVideoStatus	修改视频状态	用户id，视频id，审核状态	视频审核状态

### 4.4.5 搜索接口设计

	接口名	作用	传入参数	传出参数
SearchController	ReversoSearch	综合视频	搜索内容，搜索选项	视频搜索状态
	searchSpecialVideo	查看特定类型的视频	视频类型id	视频搜索状态

## 4.5 音视频优化处理

### 4.5.1 优化原因

如果直接存储用户上传的mp4文件，往往会存在以下问题。

- **不适合流式传输。**mp4通常需要下载整个文件才能开始播放，这在低带宽的条件下会导致等待时间较长。



- **无法逐段加载。**这也导致如果用户希望跳到视频的某一部分。仍然需要等待视频加载完成。
- **无法自适应比特率。**mp4通常包含固定比特率视频，这在不同网络速度下可能导致质量问题。
- **有限的安全性。**虽然mp4也支持加密和数字版权管理，但相对有限。

因此我们选择表现更为优秀的m3u8进行存储。相较mp4，m3u8在以下几个方面具有明显的优势

- **自适应比特率和分辨率。**m3u8格式支持自适应比特率和分辨率，这意味着根据用户的网络速度和设备性能，可以在不同质量级别之间动态调整视频的播放质量。这提供了更好的用户体验，确保用户可以顺畅观看视频而无需等待缓冲。
- **流式传输和逐段加载。**m3u8格式将视频分为多个小段，这允许视频按照需求逐段加载和播放。
- **安全性。**m3u8格式支持数字版权管理（DRM）和加密，可以更好地保护视频内容的安全，防止非授权复制和传播。
- **缓存与CDN加速。**m3u8格式易于与CDN集成，从而可以更好地分发视频内容并提高加载速度。

### 4.5.2 优化思路

- 在接收到用户传递mp4文件后，我们先将mp4文件转化为ts文件。这一步处理是为了在对文件进行切片时可以处理的更快，减少用户等待时间
- 再利用ffmpeg对ts文件进行切片，生成m3u8文件。在前期服务器的性能有限的情况下，默认的编码参数会让切片时间过长，对于部分视频，在请求时间内，不能完成切片。而这不是我们预期的结果。在服务器性能有限，但已有存储空间很大时，我们选择用空间换时间，调整ffmpeg中的preset参数，来加快切块时间。
- 最后在处理标准流输出和标准错误输出流时，选择使用两个线程各自处理，减少处理时间。

## 五、测试部署

### 5.1 测试方案

在时间紧张且要求没有线上 bug 的情况下，我们需要的是一种较为**高效且能够覆盖主要功能点**的测试方法，因此我们采用**探索性测试**结合**冒烟测试**的方式来确保关键功能的稳定性。

这里我们将样式问题独立出来，先将产品逻辑和交互列出表格进行测试：

测试用例ID	主类别	子类别	测试用例标题	前置条件	测试步骤	预期结果	实际结果	状态
TC_U_L_001	用户功能	登录	正确的用户信息登录	用户已注册	输入正确的用户名和密码进行登录	用户登录成功		通过
TC_U_L_002	用户功能	登录	记住账号、密码	用户已注册	勾选后登录，退出后重新观	记住账号、密码成功		通过

					察登录框			
TC_U_R_001	用户功能	注册	使用有效信息注册新用户	访问注册页面	输入有效的注册信息并提交	用户注册成功并直接登录		通过
TC_U_L_R_002	用户功能	登录、注册	表单验证	访问登录、注册页面	输入无效的信息或空信息	显示错误信息并阻止按钮操作		通过
TC_V_U_001	视频功能	上传	上传有效格式视频	用户已登录	选择有效格式视频文件上传	视频上传成功并出现在个人页面		通过
TC_V_U_002	视频功能	上传	上传无效格式视频	用户已登录	尝试上传无效格式文件	显示错误信息并拒绝上传		通过
TC_V_P_001	视频功能	播放	播放视频	视频已上传	点击视频进行播放	视频顺畅播放无任何错误提示		通过
TC_V_C_001	视频功能	评论	发表视频评论	用户已登录且视频已播放	输入评论并提交	评论成功显示		通过
TC_V_S_001	视频功能	搜索	通过标题搜索视频	至少一条视频数据存在	在搜索框中输入存在视频的标题	相关视频在搜索结果中显示		通过
TC_V_F_001	视频功能	收藏夹	收藏视频至收藏夹	视频已上传且用户已登录	将视频添加到收藏夹	视频成功出现在用户指定的收藏夹中		通过
TC_V_D_001	视频功能	删除视频	用户删除自己的视频	视频已上传且用户已登录	用户选择已上传的视频进行删除	视频删除成功，并从用户视频列表中移除		通过
TC_U_F_001	用户互动	关注	关注用户	用户已登录	查找另一用户并点击关注按钮	被关注用户显示在关注列表中		通过
TC_U_F_002	用户互动	粉丝	查看粉丝列表	用户有粉丝	访问个人中心查看粉丝列表	所有粉丝用户显示在粉丝列表中		通过
				用户已登录				通过

TC_U_U_001	用户设置	信息更新	更新个人信息		修改个人资料并保存	个人信息更新成功提示并反映新信息		
TC_MR_001	移动端兼容性	响应式设计	在手机上查看视频列表	视频列表存在数据	使用手机访问视频列表	视频列表适应手机屏幕尺寸显示		通过
TC_MR_002	移动端兼容性	触控操作	在手机上进行视频搜索	至少一条视频数据存在	使用手机的触屏功能输入搜索关键词	搜索结果正确显示且可通过触控操作选择		通过

## 5.2 部署方案

### 5.2.1 前端部署

我们选择 `Vercel` 作为前端的部署工具。

#### Why Vercel

1. 无服务器自动扩展：Vercel 提供了无服务器自动扩展的平台，无需关心服务器的配置和扩展问题。Vercel 会自动处理流量的分发和应用程序的扩展，确保应用始终处于高可用状态。
2. 部署速度：Vercel 提供了快速的部署速度，它支持快速的持续集成和部署（CI/CD），因此可以轻松将最新的代码变更部署到生产环境中，确保快速迭代和交付。
3. 简单的域名管理：Vercel 提供了简单的域名管理工具，可以轻松配置自定义域名，并管理 DNS 设置，使应用能够通过自定义域名访问。
4. 集成与生态系统：Vercel 与现代前端工具和框架紧密集成，包括 Vite、Next.js、React 等。

#### Vite 和 Vercel 的完美搭配

1. 构建速度：Vite 的特点是快速的构建速度，它可以在开发过程中实时更新，并且在生产构建中也能够快速生成优化的静态资源。这与 Vercel 的快速部署速度相得益彰。
2. 静态托管：Vercel 提供强大的静态文件托管功能，这与 Vite 的输出静态文件的能力非常契合。可以轻松地将 Vite 构建后的静态文件部署到 Vercel，同时利用 Vercel 的缓存和CDN功能提高应用性能。
3. 自动部署：结合 Vercel 的自动部署功能和 Vite 的持续集成工具，我们可以实现无缝的自动化部署流程，确保每次代码提交都能触发自动构建和部署，减少人为错误、提高效率。

### 5.2.2 后端部署

我们选择Jenkins+Docker作为部署工具，jenkins与docker相结合带来了许多好处

- 1. 自动化构建和部署：**Jenkins是一个强大的持续集成和持续部署工具，它可以自动化整个软件开发生命周期中的构建、测试和部署流程。Docker是一种容器化技术，它可以将应用程序和其依赖项封装到一个独立的容器中。结合使用时，Jenkins可以自动构建Docker镜像，并在各个环境中部署这些容器。
- 2. 环境隔离：**Docker容器提供了环境隔离的好处，确保开发、测试和生产环境之间的一致性。Jenkins可以使用Docker来确保在不同环境中部署相同的容器，从而减少了"在我的机器上可以工作"这种问题。
- 3. 快速部署：**Docker容器是轻量级的，启动和停止速度很快，使得快速部署成为可能。Jenkins可以触发自动化的构建和部署流程，使新版本的应用程序能够快速部署到生产环境。

## 六、项目总结与反思

### 6.1 项目自评

在开发期间，了解到群内有朋友说做到了分布式、微服务等技术，但我们并没有选择去使用更高级的技术架构，是考虑到了以下因素：

- 1. 适用性：**这些架构更多是面向体积更为庞大的项目，相对于本次的比赛项目很难体现架构的优势。
- 2. 时间压力：**更高级的技术架构意味着更多的搭建时间，对于紧张的排期并不适合。

因此，我们将更多的目光放到了比赛的整体评分机制上，按照以下几个方面来自评项目：

#### 功能完成度、丰富度（50%）—— 完成题目中的基础功能，尽可能多的实现挑战性目标

在项目中，我们致力于实现题目中规定的基本功能，并积极探索挑战性的目标。我们的努力包括但不限于：

- **基本功能实现：**我们确保了所有基本功能的顺利实现，以满足比赛的最低要求。
- **挑战性目标：**为了突显我们的技术实力，我们不仅仅停留在满足基本要求上，还尽可能多地实现了具有挑战性的目标，以提升项目的综合价值。

#### 架构设计（30%）—— 清晰表达架构设计意图，以及架构的合理性

我们秉承稳定、安全、契合且高效的理念，在架构设计方面进行了深入思考，并清晰地表达了我们的设计意图以及架构的合理性。

在搭建项目之初，我们对各种必要的配置进行了个性化操作，同时引入了很多的性能优化方案、解耦方案，对我们整个项目的顺利开发埋下了伏笔。

相关的架构图也在第一部分体现。

#### 代码风格（10%）—— 良好的编程风格，注重代码可复用性、可阅读性

我们非常重视良好的编程风格，以确保代码具备高度的可复用性和可阅读性，相关的规范也体现在文档的 1.4.1 中。

## 团队协作（10%）—— 团队分工合理性

我们在项目中实现了高效的团队协作，以确保项目的成功完成。我们的团队协作包括以下方面：

- 任务分工：我们明确了每个成员的任务和职责，避免了任务重叠和冲突。
- 定期会议：我们定期召开会议，分享进展并讨论项目的下一步计划。
- 共享资源：我们建立了共享的项目文档和资源库，以便团队成员之间轻松共享信息和文件。
- 灵活适应：我们的团队能够快速适应项目中的变化和挑战，迅速调整策略和计划，确保项目的顺利推进。

相关的协作方式也体现在了文档的第一板块中。

## 6.2 项目的优化项方向

项目开发过程中，我们意识到细节是产品成功的关键。例如，优化的视频加载速度显著减少了用户的等待时间，提升了用户的整体满意度。我们还特别关注了界面的友好提示设计，减少了用户在操作过程中的困惑和可能的挫败感。

即使功能相对完善，但我们认为还有很多值得提升的点，比如社交互动功能、推荐算法、多端适配……

## 6.3 项目中存在的遗憾

尽管我们的产品多方面都达到了预期目标，但仍有一些遗憾。

- 高级功能缺失：时间和资源的限制导致我们未能在初版产品中集成更多的高级功能，如个性化推荐系统等
- 风险评估：在项目早期，我们对潜在的技术和风险评估不够，未能及时制定有效的应对措施，导致在比赛截止前加急修复了一些重要问题。

## 6.4 自我反思

### 周想

- 首先在决定要做这个项目的时候，一个好的带队人是很重要的。这个项目有且仅有三个人协作完成，我们分为两个前端一个后端，其中UI设计、界面实现、接口对接占了前端大部分的活，而后端则实现数据库设计以及接口设计、接口编写。
- 一个比较成熟的项目开发应该最开始编写设计文档的时候就统统需要将确定好的实现功能都确定好，而不是在后面看缺什么必要的功能再回头去补，这样子使得开发过程混乱无章。
- 这次也是只有两周的时间，并且我们作为开发人员，经验可能有些许的不足，使得我们在协作过程中出现沟通矛盾的问题。

### 童子欣

- 在时间紧张的情况下，有效的团队管理十分重要。

- 测试部分很重要，测试方案也是需要依据项目情况提前确定的。
- 就算时间再紧张，code review 也是必不可少的环节。
- 项目的优秀程度不能就架构而论，产品设计和架构设计都是不可轻视的点。

## 徐小帅

- 确保需求分析清晰和明确是项目顺利进行的关键，以及避免后续集成问题的关键。
- 在讨论和开发过程中进行记录和文档化十分重要。这有助于避免遗忘和混淆，以及提高项目管理的效率。
- 测试是保障软件质量的关键，特别是边界测试。通过仔细的测试，才能捕捉和修复潜在的问题。
- 在开发前就应该做好模块规划，否则后期极易出现代码耦合。