**布局**

采用页面。

左侧显示所有文件夹（iview的menu）。可以执行创建目录（通过一个**button**），以及更改目录（通过icon，**打开一个modal**，可以改**名称**和**父目录**）

右侧上方显示目录导航，以及过滤（input输入关键字，select选择类型（所有文档/自己文档/收藏文档/推荐文档））。

右侧下方为table，显示符合条件的文档。每条记录带链接，可以直接在新页面打开文档。

**目录（folders）**

**db:**

Folders:{

Name:string required; 1~255 目录名称

AuthorId: objectId required; 谁创建了目录

ParentFolderId: objectId optional; 简单起见，只支持2层目录，根据是否设置parentFolder，判定是否为顶级目录

cDate: date required

uDate:date optional

dDate:date optional

}

**创建新目录**

**输入参数**

Values:{

“recInfo”:{

name:{“value”:”xxxx”},

AuthorId:{value:objectId}

author:{“value”:yyyy},

ParentFolder:{“value”:zzzz}

}

“curColl”:”floders”

}

**步骤**

1 检查格式和数据

2 对字段name，使用DOMPurify进行XSS检测

3 如果输入中，parentFolder存在

3.1检查parentFolder在db中是否存在

3.2检查parentFolder的author是否为当前用户（用户只能在自己的目录下创建目录）

3.3 检查parentFolder的记录，其中字段parentFolder是否为空，不空，报错，保证只能创建2层目录

4 从当前用户（sessionId）中获得author

**更新目录**

目录可以更改的部分是；1. 名称；2 上级目录

Values:{

“recInfo”:{

name:{“value”:”xxxx”},

ParentFolderId:{“value”:zzzz}

},

“recId”: objectId,

“curColl”:”folders”

}

1 检查格式和数据

2 对字段name，使用DOMPurify进行XSS检测

3 检查recId是否存在

4 如果输入中parentFolder存在

如果值不是null

4.1检查parentFolder在db中是否存在

4.2检查parentFolder的author是否为当前用户（用户只能更新在自己的目录下）

4.3 检查parentFolder的记录，其中字段parentFolder是否为空，不空，报错，保证只能创建2层目录

如果值是null（说明要升级为顶级目录），直接执行

**文档（articles）**

**db:**

5个table，一个用来存储用户创建的文档(articles)，一个用来存储文档的关键字(tags)，一个用来存储文档的评论(articleComments)，一个用来存储文档的内插图片(articleImages)，一个用来存储文档的附件(articleAttachments)，一个用于文档归类（categories，2层，例如LTE-A/MIT）

**articles**:{

Name:string required; 1~50 文档名称

Status: **string|enum** required; editing/finish 文档状态：是否可以被搜索到

AuthorId: objectId required; 谁创建了目录

FolderId: objectId required; 放在哪个目录

pureContent:string optional; 1~10000 文档内容（纯文本）

htmlContent:string optional; 1~50000 文档内容（html）

categoryId: objectId required 文档分类

TagsId:array[objectId] require; 1~5/2~20 关键字：最多5个，每个2～20字符

ArticleImagesId: [objectId] optional;0~5 文档内插图片，最多5幅

ArticleAttachmentsId:[objectId] optional; 0~5 文档附件，最多5个

articleCommentsId: [objectId] optional;0~255 做多255个评论

cDate: date required

uDate:date optional

dDate:date optional

}

**Tag**:{

Name: string required; 2~20

cDate: date, required

}

**article\_comment**:{

articleId:objectId required; 属于哪个文档

authorId: objectId required; 谁写的

content: string required; 1~255 评论内容

~~commentImages:~~

cDate: date required

~~uDate:date optional~~

dDate:date optional

}

**Article\_image**:{

Name:string required;2~255 作为image的alt属性（从上传文件属性读取，而不是根据client的输入）

hashName:string required

pathId:objectId required 图片存储的位置（在coll config中定义）

Size: int required; 0~max in byte（在const/globalConfig/uploadFileDefine定义）

articledId:objectId required; 是哪个文档的图片

authorId:objectId required 是哪个作者上传的（方便统计作者的配额）

cDate: date required

~~uDate:date optional~~

dDate:date optional

}

**articleAttachments**:{

name: string required:1~255 附件名称

hashName:string required:40+4 附件hash名称（防止猜测附件名）

pathId:objectId required 图片存储的位置（在coll config中定义）

size:int required; 1~5,000,000 inbyte

articledId:objectId required; 是哪个文档的附件

authorId:objectId required 是哪个作者上传的（方便统计作者的配额）

cDate: date required

~~uDate:date optional~~

dDate:date optional

}

//记录其他用户对文档的态度

Like\_dislike:{

articleId: article\_objectId required; 1~100 文档

authorId: user\_objectId optional 评论用户

like: Boolean required like dislike 根据不同的URL，在server直接设置。所以无需任何inputRule（client和server）

cDate:date optional

dDate:date optional

}

//处理完Like\_dislike后，自动处理，无需任何inputRule

**Like\_dislike\_static**:{

articleId: article\_objectId required; 1~100 文档

likeTotalNum:int optional

dislikeTotalNum: int

}

记录所有用户的推荐信息

**Recommend**:{

articleId: objectId required 推荐的文章

From: objectId required 谁推荐

ToUser： objectId optioa 推荐给单个用户

ToGoup： objectId optioa 推荐给朋友圈

ToPublicGoup： objectId optioa 推荐给群

cDate

}

记录单个用户的推荐/接收的推荐文档

userRecommend:{

userId: objectId

sendRecommend:

Receiverecommend:

}