互联网数据库开发课程-LLLG组-实现文档

高效规划和团队合作

在最开始会议中，首先将环境进行配置，之后讨论了每个人的分工情况，在本次项目的的构建中，经讨论小组要制作的功能后，决定根据任务量分成前端页面；新闻、研究、联系、登录界面；博客及评论；视频及评论。因此实行了以下分工：

组长：李威远

领队分工、页面设计、前后端对接、数据可视化实现

组员：李秉睿

部分数据处理、新闻、研究、联系、登录界面设计和编写

组员：郭昱杰

博客及评论模块的数据处理、后端设计、前端界面编写

组员：刘国民

视频页面的整体框架、后台对视频的增删改查

之后具体分工实现细节会在成员任务完成情况中具体展示。

小组一起解决难题

在项目实现后期，遇到了一定的瓶颈，小组成员一起开会来解决这些问题。通过许多次开会，所有的困难得以克服，最终成功实现该项目。

文档分工

需求文档：李秉睿

设计文档：李秉睿

实现文档：team

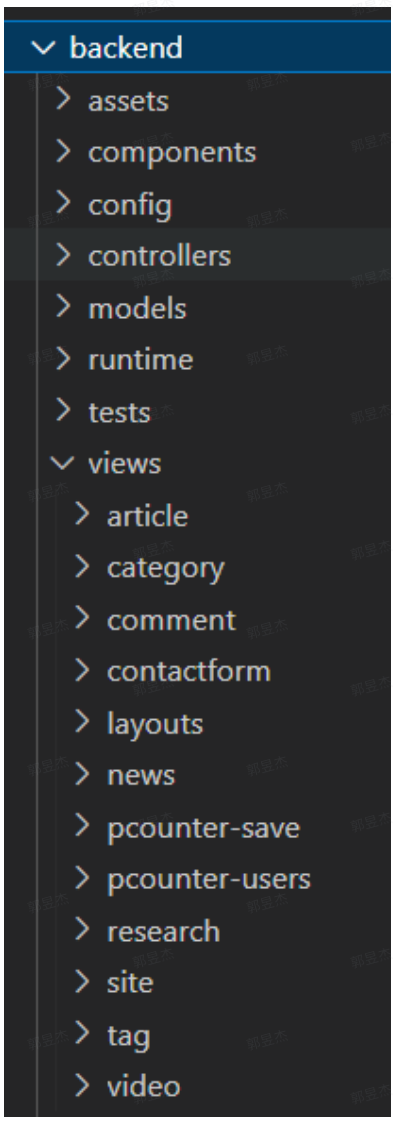
用户手册：郭昱杰

部署文档：刘国民

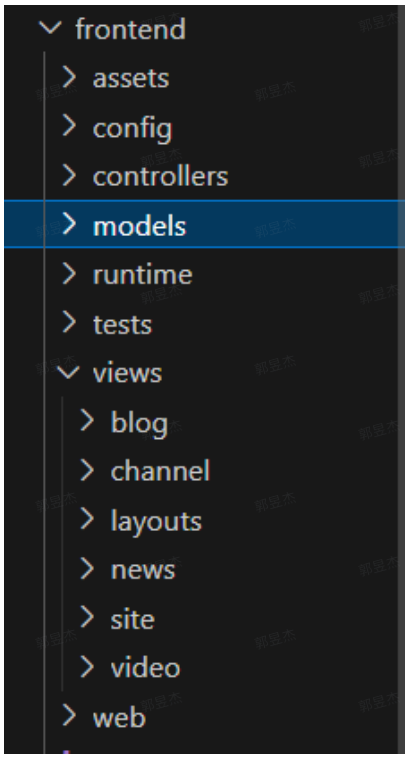
项目展示PPT：team

文件结构展示

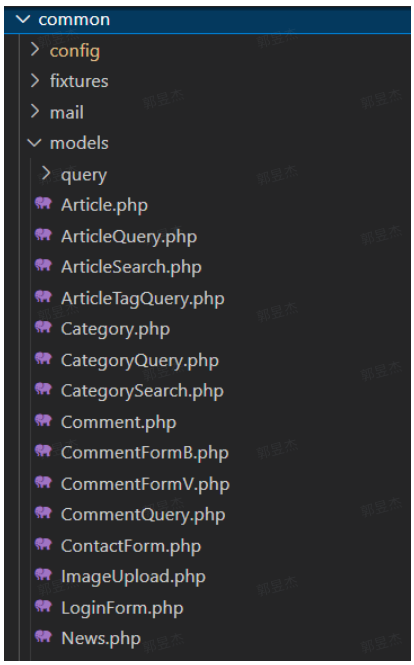
Backend



Frontend



Common



上述三个模块为实现本次项目的主要结构。common为backend和frontend的提供一些必要的API来辅助backend和frontend的构建。

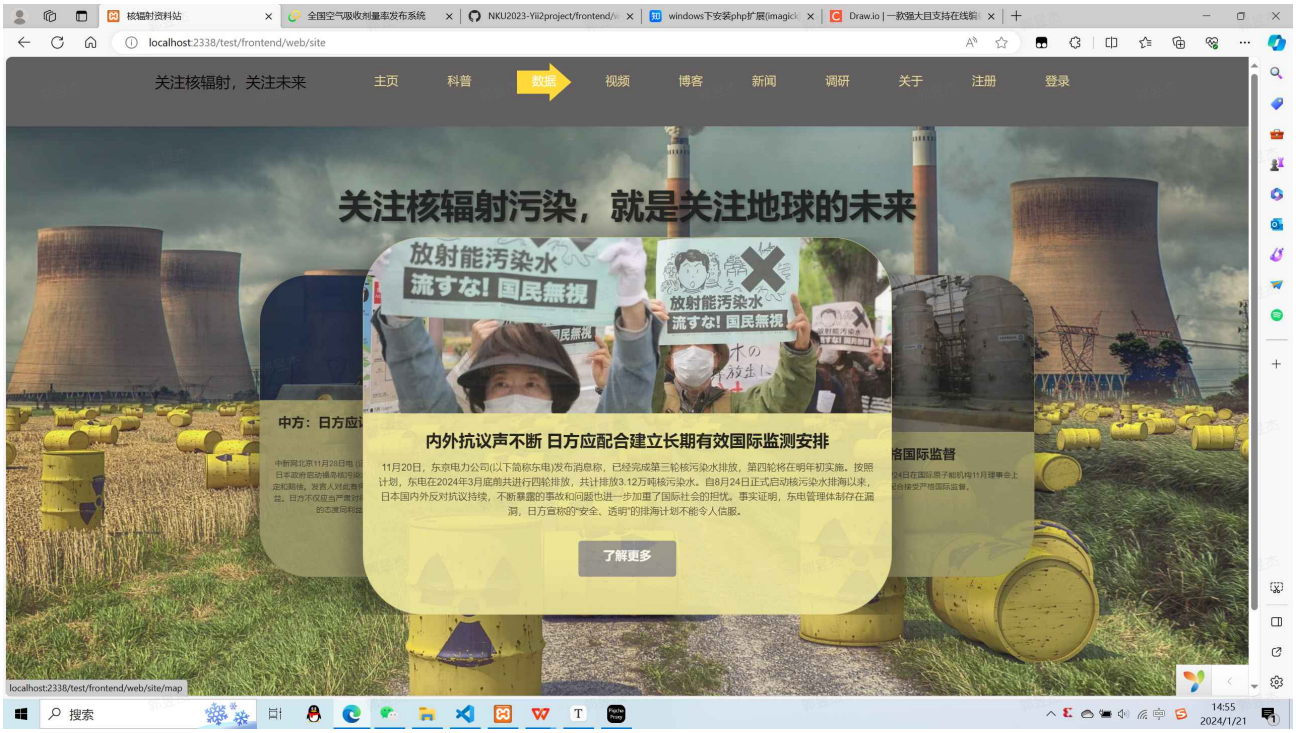
主要代码实现

1.1导航栏与首页设计

（1）功能介绍

本网页提供了一个简洁直观的动态导航栏功能，在选中分页的时候会左移导航栏，显示选择框，呈现一个舒适的动态效果。

此外，本网页借助于前端的跑马灯呈现，为用户提供了一个良好的视觉体验，并借助于MVC三层功能的简单处理，能够从新闻数据库中获取最新的几篇文章作为主页面的展示。



（2）实现与设计（含MVC三层逻辑）

项目的框架是一个基于yii2的advanced模板的框架，因此包含layout等处理，我们这里不再过多赘述模板中的原理，我们大概看一下导航栏的实现：

1. **View**：frontend/views/layouts中的全部内容、frontend/views/site/index.php

首先我们在header.php中设计实现了一个兼顾注册登录功能的动态导航头，其主要使用路由去访问各个页面对应的controller控制函数，并通过应用主体的user属性去协助判断当前是否是已登录状态，如果是就不再显示注册和登录栏目：

代码如下：

```
<?php $this->beginBody() ?>
    <?php
    echo "<table>";
    NavBar::begin([
        'brandLabel' => Yii::$app->name,
        'brandUrl' => Yii::$app->homeUrl,
        'options' => [
            'class' => 'navbar-expand-lg navbar-light',
            'style' => 'background-color: #616060; box-shadow: 0 4px 6px
rgba(0, 0, 0, 0.1);',
        ],
    ]);
    $menuItems = [
        ['label' => '主页', 'url' => ['/site']],
```

```
        ['label' => '科普', 'url' => ['/site/about']],
        ['label'=>'数据','url'=>['/site/map']],
        ['label'=>'视频','url'=>['/video/index']],
        ['label'=>'博客','url'=>['/blog/blog']],
        ['label' => '新闻', 'url' => ['/news/index']],
        ['label' => '调研', 'url' => ['/site/research']],
        ['label'=>'关于','url'=>['/site/contact']]
    ];
    if (Yii::$app->user->isGuest) {
        $menuItems[] = ['label' => '注册', 'url' => ['/site/signup']];
        $menuItems[] = ['label' => '登录', 'url' => ['/site/login']];
    } else {
        $menuItems[] = '<li>'
            . Html::beginForm(['/site/logout'], 'post')
            . Html::submitButton(
                '登出 (' . Yii::$app->user->identity->username . ')',
                ['class' => 'btn btn-link logout']
            )
            . Html::endForm()
            . '</li>';
    }?>
    <?php
    echo Nav::widget([
        'options' => ['class' => 'navbar-nav ml-auto',],
        'items' => $menuItems,
    ]);
    NavBar::end();
```

随后，针对需求不同css格式的不同页面，我们给出了不同的layout模板(默认使用main模板)，在其中启用这个header头，具体实现和advanced模板类似，这里我们不多赘述，如下所示即为index主页面所使用的模板框架及其调用的css属性：

```
    <?php $this->beginBody() ?>
    <div class="wrap h-100 d-flex flex-column">
        <?php echo $this->render('header') ?>
            <div class="content-wrapper-index p-3" >
                <?= Alert::widget() ?>
                <?= $content ?>
            </div>
    </div>
    <?php $this->endBody() ?>
```

我们通过widget的小部件功能封装对index.php的调用，其中index.php实现中，我们核心关注跑马灯功能的实现，我们使用js插件，在其中引用复现了如下的两个类，具体代码比较长，我们不再展示

DraggingEvent 类

- 构造函数：接受一个目标元素（默认为 undefined），用于初始化拖拽事件的目标元素。
- event 方法：接受一个回调函数，用于处理拖拽事件。在鼠标按下或触摸屏开始时，注册相应的事件监听器，包括鼠标移动、离开文档、鼠标抬起、触摸移动、触摸结束等事件。在拖拽过程中，调用回调函数处理事件，并在拖拽结束时移除事件监听器。

CardCarousel 类（继承自 DraggingEvent）

- 构造函数：接受一个容器元素和一个可选的控制器元素（默认为 undefined）。初始化了一些 DOM 元素、卡片数据和事件监听器。
- updateCardWidth 方法：用于更新卡片宽度，在窗口大小改变时调用。
- build 方法：构建初始状态的卡片布局，计算卡片的位置、缩放比例、层级等，并更新卡片样式。
- controller 方法：处理控制器（键盘）事件，根据按键调整卡片的位置。
- calcPos 方法：计算卡片的水平位置。
- updateCards 方法：更新卡片的样式，包括位置、缩放、透明度和层级。
- calcScale2 和 calcScale 方法：计算卡片的缩放比例。
- checkOrdering 方法：检查卡片的顺序，处理卡片位置的变化。
- moveCards 方法：处理卡片的移动，根据拖拽距离更新卡片样式

在index文件中，我们使用传入的参数new进行页面的渲染和新闻图片、信息等内容的填充，这部分的数据源自我们的News数据库表，因此这部分内容的实现则需要Controler和Model的支持

2. **Controller：**frontend/control/siteController

该类起初为模板自带，我们对其进行了修改，增加了网页的切换的实现，对不同的行为跳转到不同的网页路由，并视情况选择对应的layout框架和传输所需的数据，这部分的数据传输则需要model的相关支持，以获取index网页所需要的最新的新闻。

我们通过调用如下代码实现，其中，我们通过访问common/models/News的model来实现对数据访问：

```
    public function actionIndex()
    {
        $data=News::getAll(6);
        return $this->render('index',[
            'news'=>$data['news'],
        ]);
    }
```

3. **Model：**common/models/News

这个Model中，我们主要实现对News数据库的访问和处理，其中涉及到的首页最新新闻访问的调用代码是如下的部分：

```
        public static function getAll($pageSize = 5)
        {
            $query = News::find()->latest();
            $count = $query->count();
            $pagination = new Pagination(['totalCount' => $count,
    'pageSize'=>$pageSize]);
            $news = $query->offset($pagination->offset)
                ->limit($pagination->limit)
                ->all();
            $data['news'] = $news;
            $data['pagination'] = $pagination;
            return $data;
        }
```

这段代码的原理如下：

a. 查询构建： 使用 Yii2 框架的 ActiveRecord 查询构建器（`News::find()`）创建一个查询，并通过 `latest()` 方法将结果按照发布时间降序排序。

b. 分页处理： 获取查询结果的总数，并使用 `Pagination` 类初始化一个分页对象，设置每页显示的条目数（`$pageSize`）。然后，通过 `offset` 和 `limit` 方法应用分页到查询，以获取当前页的新闻数据。

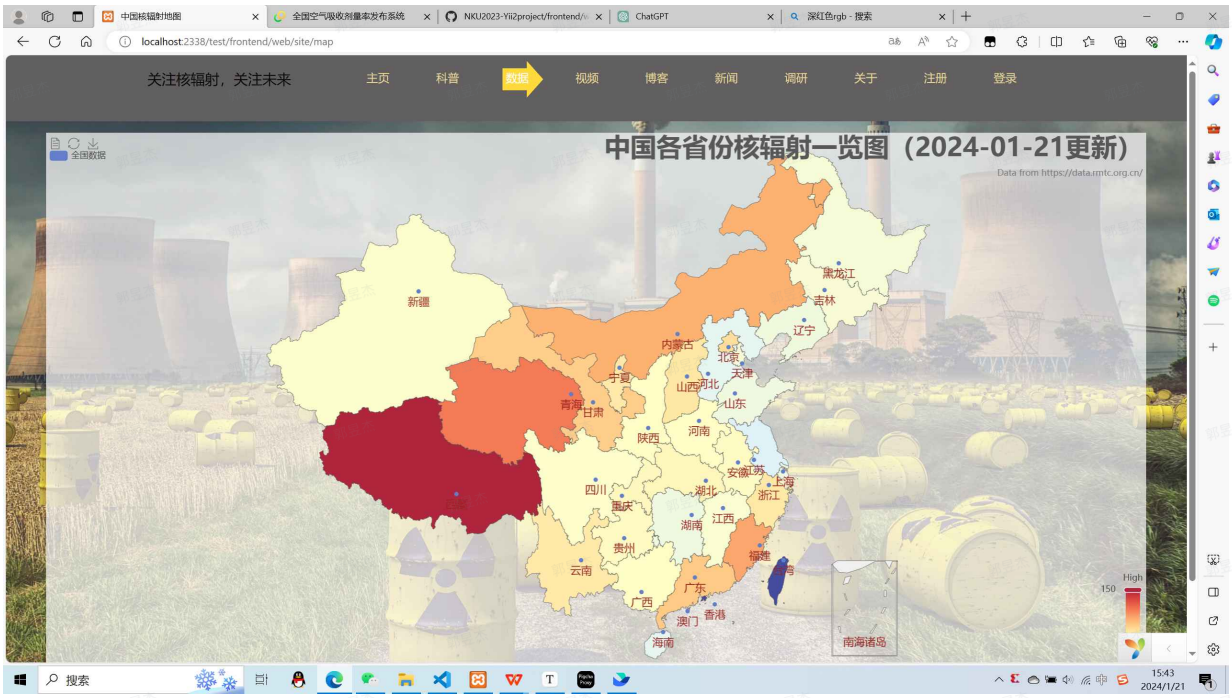
c. 返回结果： 将获取到的新闻数据和分页信息封装到一个关联数组中，然后返回该数组作为结果。这样，调用该 `getAll` 方法的代码可以获取新闻数据和相关的分页信息，用于展示和处理分页功能。

基于如上的一个简单的**MVC三层处理流程**，我们就完成了index页面对最新的几个新闻的数据获取，以此通过先前给出的跑马灯js渲染，实现了一个简洁大方的index首页设计，让用户在第一时间就能感受到核辐射防治的紧迫性和实时性。

1.2数据可视化页面Map的实现

（1）功能介绍

我们还额外实现了基于全球实时最新的核辐射数据，构建相关了相关的动态可视化数据图，方便用户去查询今日的核辐射数据，了解国家核辐射挑战现状



（2）实现与设计（含MVC三层逻辑）

核辐射数据的获取源于我们组**李秉睿**同学对于网址https://data.rmtc.org.cn/gis/listtype0M.html的爬虫，该网站记录了全国各地实时更新的最新核辐射数据，这里我不再赘述

核辐射地图的代码实现基于ecahrt.js和中国地图china.js插件，我们使用echart绘制了这一张地图，通过在文档加载完成后执行的函数$(document).ready(function)完成了地图的渲染和处理，该地图支持拖拽、放大、选中、数据网站点击跳转、热力图数值映射等等功能。

该可视化图像的具体实现同样依赖于MVC三个层级的传输，但部分处理不同，我们来看一看：

1. **View**：frontend/views/site/map.php，frontend\web\js\plague-map.js

map.php中使用了和index类似的框架，但是由于渲染图像是调用js文件实现，map.php中并未直接体现对Controller层级的请求，而是在js中完成。

在js文件中，我们通过ajax请求完成了这一功能：

```
$.ajax({
    url: 'Map/value', // 相关控制器的路由
    type: 'GET',
    dataType: 'json',
    success: function (jsonData) {
        renderMap(chinaMapChart, jsonData);
    },
    error: function (request) {
        alert('获取数据失败');
    }
});
```

2. **Controller**：frontend\controllers\MapController.php

3. **Model：**common\controllers\MapController.php

这里，我们首先检查当前是否是最新的数据，如果是不用对数据库获取，本地保存了相关数据的json文件，直接读取传递即可，如果不是，我们则需要通过Map的Model对map数据库中数据进行获取。这部分代码实现基本上和正常的数据库读入处理相同，区别在于增加了判断和数据类型转化，转换处理如下所示：

```
        // 处理加载的数据
        var convertedData = [];
        for (var key in jsonData) {
            if (jsonData.hasOwnProperty(key)) {
                var item = jsonData[key];
                convertedData.push({
                    name: item.name,
                    value: parseFloat(item.avg_value) // 转换为浮点数
                });
            }
        }
```

通过如上三步处理后，我们即可完成数据的获取。

在数据的获取后，我们通过如下ecahrts代码完成对可视化地图的渲染：

```
function renderMap(chinaMapChart, jsonData) {
        // 处理加载的 JSON 数据
        var convertedData = [];
        for (var key in jsonData) {
            if (jsonData.hasOwnProperty(key)) {
                var item = jsonData[key];
                convertedData.push({
                    name: item.name,
                    value: parseFloat(item.avg_value) // 转换为浮点数
                });
            }
        }

        optionChinaMap = {
            title: {
                text: '中国各省份核辐射一览图（2024-01-21更新）',
                subtext: 'Data from https://data.rmtc.org.cn/',
                sublink: 'https://data.rmtc.org.cn/',
                left: 'right',
                textStyle: {
                    fontSize: 36, // 设置标题的字体大小
```

```
                }
              },
              legend: {
                orient: 'horizontal',//图例的排列方向
                x:'left',//图例的位置
                y:'20',

                data:['全国数据'] // 设置标题的字体大小

            },

            tooltip: {//提示框组件
                formatter:function(params,ticket, callback){//提示框浮层内容格式器，支持字符串模板和回调函数两种形式。
                    return '核辐射数值'+'<br />'+params.name+': '+params.value + 'nGy/h'
                }//数据格式化
            },
        backgroundColor:'rgba(255,255,255,0.7)',//背景色
        visualMap: {
            left: 'right',
            min: 0,
            max: 150,
            inRange: {
              color: [
                '#313695',
                '#4575b4',
                '#74add1',
                '#abd9e9',
                '#e0f3f8',
                '#ffffbf',
                '#fee090',
                '#fdae61',
                '#f46d43',
                '#d73027',
                '#a50026'
              ]
            },
            text: ['High', 'Low'],
            calculable: true
          },
          toolbox: {
            show: true,
            //orient: 'vertical',
            left: 'left',
            top: 'top',
            feature: {
```

```
                dataView: { readOnly: false },
                restore: {},
                saveAsImage: {}
            }
        },
             series : [
                 {
                     name: '全国数据',
                     type: 'map',
                     mapType: 'china',
                     zoom: 1.1,
                     roam: true,//是否开启鼠标缩放和平移漫游
                     itemStyle:{//地图区域的多边形 图形样式
                         normal:{//是图形在默认状态下的样式
                             label:{
                                 show: true,
                                 textStyle: {color: "#8B0000",fontSize : 15}

                             }
                         },
                         emphasis:{//是图形在高亮状态下的样式,比如在鼠标悬浮或者图例联动高亮时

                             label:{show:true},
                         }
                     },
                     top:"100",//组件距离容器的距离
                     data: convertedData
                 }
             ]
        };
    chinaMapChart.setOption(optionChinaMap, true);
}
```

2.1blog和comment:

实现功能

对于blog和comment模块，设计有如下的功能：

- 搜索浏览博客：设置目录和进行搜索
- 发表与核污染内容有关的博客，并且可以进行修改
- 在博客里面发表相应的评论
- 在后台可以对博客和评论进行管理：修改或者删除博客的内容，审核评论是否可以发表

针对上述功能，我们对blog和comment进行如下的实现：

实现方法

Model

该部分主要是位于common/models目录下的Article、Category、comment等与之有关的文件。下面重点介绍Article，Category，Comment和ImageUpload。

首先介绍Article，article中的函数包括（这里展示的代码只取比较有代表性的部分）：

- 获取article的某些属性：比如由谁创建，当前是否是公开发表的状态等等。如下面代码的例子，分别是获取由谁创建，获取图像的地址（其它部分代码类似）。

```
    public function getCreatedBy()
    {
        return $this->hasOne(User::className(), ['id' => 'created_by']);
    }

    public function getImage()
    {
        $imgurl = ($this->image) ? '../../../frontend/web/uploads/' . $this->image
    : '../../../frontend/web/img/no-image.png';
        return $imgurl;
    }
```

- 修改某些属性，修改图像需要用的ImageUpload的model，在之后会介绍，下面是修改图像的例子。

```
    public function saveImage($filename)
    {
        $this->image = $filename;
        return $this->save(false);
    }
```

- 修改某些其它属性，按照某些属性进行排序（通过查询之后进行排序实现）。例如下面的代码，是按照受欢迎的程度或者时间来进行排序。

```
    // 获取最受欢迎的三篇博客
    public static function Popular()
    {
        return Article::find()->published()->orderBy('viewed desc')->limit(3)-
    >all();
```

```
    }

    // 获取最新的四篇博客
    public static function Recent()
    {
        return Article::find()->published()->orderBy('date asc')->limit(4)->all();
    }
```

对于article，我们概要添加一个article的搜索函数，将该功能的实现放在ArticleSearch类中，代码如下：

```
    public function search($params)
    {
        $query = Article::find();

        $dataProvider = new ActiveDataProvider([
            'query' = > $query,
        ]);

        $this->load($params);

        if (!$this->validate()) {
            return $dataProvider;
        }

        // 这些属性需要精确查找
        $query->andFilterWhere([
            'id' = > $this->id,
                'date' = > $this->date,
                'viewed' = > $this->viewed,
                'created_by' = > $this->created_by,
                'status' = > $this->status,
                'category_id' = > $this->category_id,
        ]);

        // 这些属性需要模糊查找
        $query->andFilterWhere(['like', 'title', $this->title])
            ->andFilterWhere(['like', 'description', $this->description])
            ->andFilterWhere(['like', 'content', $this->content])
            ->andFilterWhere(['like', 'image', $this->image]);

        return $dataProvider;
    }
```

对于Category，与article类似，但有一个函数的功能是获取该目录下的article，代码实现如下：

```
    public static function getArticlesByCategory($id)
    {
        $query = Article::find()->where(['category_id' = > $id])->published();
        $count = $query->count();
        $pagination = new Pagination(['totalCount' = > $count, 'pageSize' = > 6]);
        $articles = $query->offset($pagination->offset)
            ->limit($pagination->limit)
            ->all();
        $data['articles'] = $articles;
        $data['pagination'] = $pagination;

        return $data;
    }
```

对于Comment的实现，在common/models/Comment.php中，该类中，主要提供的方法为：

- 获取一些属性：该功能的实现与上面Article类似，不再做赘述。
- 实现评论审核的通过与不通过，该功能是为后台提供服务的，前台加载评论的时候会过滤掉DISALLOW的评论。

```
    /*
     实现评论是否通过，在后台进行该操作。
     在前台，加载评论只会加载statue为ALLOW的，DISALLOW的进行过滤
    */
    public function allow()
    {
        $this->status = self::STATUS_ALLOW;
        return $this->save(false);
    }

    public function disallow()
    {
        $this->status = self::STATUS_DISALLOW;
        return $this->save(false);
    }
```

最后一个重要的model是common/modes/ImageUpload.php，在该类里面，实现了文件的传输，实现主要逻辑如下（一些小细节不做赘述）：

- 上传图像，主要是将上传的文件放掉frontend的路径下提供给frontend使用。

```
    public function uploadFile(UploadedFile $file, $currentImage)
    {
        $this->image = $file;
        if ($this->validate())
        {
            // 检查是否存在，如果存在删除文件，确保只有一个文件与对象相关联
            if ($this->fileExists($currentImage))
            {
                unlink($this->getFolder().$currentImage);
            }
        }
        // 生成新的文件名并保存
        $filename = $this->generateFilename();
        $this->image->saveAs('@frontend/web/uploads/'.$filename);
        return $filename;
    }
```

- 删除图像，检测文件是否存在，存在就进行删除

```
        public function deleteCurrentImage($currentImage)
        {
            // 存在该文件就进行删除
            if($this->fileExists($currentImage))
            {
                unlink($this->getFolder() . $currentImage);
            }
        }
```

Controller

该部分分为前台和后台的controller，对应的controller分别位于frontend/controllers中的BlogConteroller.php与位于backend/controllers中的BlogController.php、CommentController.php和CategoryController.php。

这些Controller的内容实际上相差不大，我们挑选一个来进行说明，其它不再做赘述。原理一致。我们以backend中的CommentController.php为例来进行说明。只说明比较关键的部分。

- 响应对应事件：如delete，allow，disallow，当发出post之类的请求时，会做出一些响应。如下面例子所示，虽然下面例子没有判断是什么请求，但思路与其它controller一致，找到对应的元素，然后进行删除，设置allow等操作。

```
    // 响应delete，删除对应的评论
    public function actionDelete($id)
    {
```

```
        $comment = Comment::findOne($id);
        if ($comment->delete())
        {
            return $this->redirect(['comment/index']);
        }
    }
    // 响应allow，将对应action设置为通过（在对应页面可见）
    public function actionAllow($id)
    {
        $comment = Comment::findOne($id);
        if ($comment->allow())
        {
            return $this->redirect(['index']);
        }
    }
    // 响应disallow，将对应action设置为不通过（对应页面会过滤掉）
    public function actionDisallow($id)
    {
        $comment = Comment::findOne($id);
        if ($comment->disallow())
        {
            return $this->redirect(['index']);
        }
    }
```

- 在后台会展示一个评论的列表，采用了actionIndex函数进行响应。调用了model中comment部分的查询函数getAll()获得并将数据传递给上层view。我们分成了多个comment进行展示，因此传递时使用$this->render("index", [array])后跟查询并处理后的数据内容。

```
    public function actionIndex()
    {
        $comments = Comment::find()->orderBy('id desc')->all();

        return $this->render('index', ['comments' = > $comments]);
    }
```

View

view分为前台的view和后台的view，前台的view为frontend\views\blog中，后台的view位于backend\views\article、backend\views\category\和backend\views\comment中。下面来介绍比较重要的几个view：

前台

前台页面的实现思路是找到模板并套用，在一些地方会做出一定的修改。

- frontend\views\blog\blog.php：这个是进入文章界面的主要板块，第一个部分为图片的滑动部分，第二个部分为
- frontend\views\blog\create.php：为创建博客页面的php文件
- frontend\views\blog\view.php：点击进入某一个博客具体内容的页面
- frontend\views\blog\mypost.php：管理当前账号所写文章的界面
- frontend\views\blog\update.php：更新当前文章

下面是对于页面的展示：

后台

该部分的形式其实都类似，仅以blog的为例，在主要的界面的php文件中，大部分都是如下面的代码所示，利用yii框架中的部件生成，填入的数据有：序列号，image，标题，描述，日期，创作者，是否公开发表。

```
    <!-- 使用 Yii 框架中的GridView小部件来生成一个数据表格 -->
    <!-- 显示数据库模型的列表 -->
    <?= GridView::widget([
        'dataProvider' => $dataProvider,
        'filterModel' => $searchModel,
        'columns' => [
            ['class' => 'yii\grid\SerialColumn'],
```

```
            [
                'attribute' => 'Image',
                'content' => function ($model) {
                    return $this->render('article_item', ['model' => $model]);
                }

            ],
            'title',
            'description:ntext',
            'date',
            [
                'attribute' => 'created_by',
                'content' => function ($model) {
                    return $model->createdBy->username;
                }
            ],
            'status',
            // 定义更新操作
            [
                'class' => 'yii\grid\ActionColumn',
                'buttons' => [
                        'update' => function ($url)
                        {
                            return Html::a('Update', $url, [
                                'data-method' => 'post',
                            ]);
                        }
                    ]
            ],
            // 定义删除操作
            [
                'class' => 'yii\grid\ActionColumn',
                'buttons' => [
                        'delete' => function ($url) {
                            return Html::a('Delete', $url, [
                                'data-method' => 'post',
                                'data-confirm' => 'Are you sure?'
                            ]);
                        }
                    ]
             ],
        ],
]); ?>
```

后台展示与之前my articles类似，不再做重复展示。

3.1news和research

（1）功能介绍

对于news和research模块，由于这部分信息具有一定的严肃性，不允许用户进行发布，只允许管理员从后台添加，设计有如下的功能：

- 浏览视频新闻和文字新闻：文字新闻可跳转到对应的页面、视频新闻跳转到对应的视频页面。
- 访问对应的研究页面：通过研究页面得到的链接进入外部论文网站进行观看。

由于这两部分的实现也相对类似，以news举例。

（2）实现与设计（含MVC三层逻辑）

Model

我们用gii生成对应的news的model文件，其相应的字段在rules()和attributeLabels()中设置，使用newsQuery的model进行查询工作。

```
class News extends \yii\db\ActiveRecord
{
    /**
     * {@inheritdoc}
     */
    public static function tableName()
    {
        return '{{%news}}';
    }

    /**
     * {@inheritdoc}
     */
    public function rules()
    {
        return [
            [['image'], 'string'],
            [['pubDate', 'title', 'infoSource', 'sourceUrl'], 'string', 'max'
=> 255],
            [['summary'], 'string', 'max' => 2000],
        ];
    }

    /**
     * {@inheritdoc}
     */
    public function attributeLabels()
    {
```

```
            return [
                'id' => 'ID',
                'pubDate' => 'Pub Date',
                'title' => 'Title',
                'summary' => 'Summary',
                'infoSource' => 'Info Source',
                'sourceUrl' => 'Source Url',
                'image' => 'Image',
            ];
        }

        /**
         * {@inheritdoc}
         * @return NewsQuery the active query used by this AR class.
         */
        public static function find()
        {
            return new NewsQuery(get_called_class());
        }

        public static function getAll($pageSize = 5)
        {

            $query = News::find()->latest();

            $count = $query->count();

            $pagination = new Pagination(['totalCount' => $count,
    'pageSize'=>$pageSize]);
            // 用yii2自带的分页类，进行分页展示新闻。在view中有用到。

            $news = $query->offset($pagination->offset)
                ->limit($pagination->limit)
                ->all();

            $data['news'] = $news;
            $data['pagination'] = $pagination;

            return $data;
        }

    }
```

而NewsQuery下的方法较为简单，从数据库获得接口后分别设置返回所有、一条或者按时间顺序排列最新的即可。

```
class NewsQuery extends \yii\db\ActiveQuery
{
    /**
     * {@inheritdoc}
     * @return News[]|array
     */
    public function all($db = null)
    {
        return parent::all($db);
    }

    /**
     * {@inheritdoc}
     * @return CovNews|array|null
     */
    public function one($db = null)
    {
        return parent::one($db);
    }

    public function latest(){
        return $this->orderBy(['pubDate'=>SORT_DESC]);
    }
}
```

Controller

前台：controller决定获取请求和传递响应的过程。以我们的view页面中分的多个新闻展示样式为例：采用了actionIndex函数进行响应。调用了model中video和news部分的查询函数getAll()获得并将新闻的数据传递给上层view。我们分成了多个news部分以及一个video部分进行展示，因此传递时使用$this->render("index", [array])后跟查询并处理后的数据内容。

```
class NewsController extends Controller
{

    public function actionIndex()
    {
        $this->layout = 'blog';
        $data = CovNews::getAll(16);

        $data1 = $data['news'][0];
        for ($i = 1, $j = 0; $i < 5;) {
            $data2[$j++] = $data['news'][$i++];
        }
```

```
            for ($i = 5, $j = 0; $i < 10;) {
                $data3[$j++] = $data['news'][$i++];
            }
            for ($i = 10, $j = 0; $i < 16;) {
                $data4[$j++] = $data['news'][$i++];
            }
            $videos = Video::getAll(5);
            return $this->render("index", [
                'news1' => $data1,
                'news2' => $data2,
                'news3' => $data3,
                'news4' => $data4,
                'videos' => $videos['videos'],
                'pagination' => $data['pagination'],
            ]);
        }

    }
```

后台的controller略有不同，主要是接收到相应的增删改请求后，对数据库进行操作，成功后会返回后台的news/index页面。但主体还是gii生成的，因而我们不需要进行太多操作

```
    class NewsController extends Controller
    {

        public function behaviors()
        {
            return [
                'access'=>[
                  'class'=>AccessControl::class,
                  'rules'=>[
                      [
                          'allow'=>true,
                          'roles'=>['@']
                      ]
                  ]
                ],
                'verbs' => [
                    'class' => VerbFilter::className(),
                    'actions' => [
                        'delete' => ['POST'],
                    ],
                ],
            ];
```

```
    }


    public function actionIndex()
    {
        $searchModel = new NewsSearch();
        $dataProvider = $searchModel->search(Yii::$app->request->queryParams);

        return $this->render('index', [
            'searchModel' => $searchModel,
            'dataProvider' => $dataProvider,
        ]);
    }

    /**
     * Displays a single CovNews model.
     * @param integer $id
     * @return mixed
     * @throws NotFoundHttpException if the model cannot be found
     */
    public function actionView($id)
    {
        return $this->render('view', [
            'model' => $this->findModel($id),
        ]);
    }

    /**
     * Creates a new CovNews model.
     * If creation is successful, the browser will be redirected to the 'view'
page.
     * @return mixed
     */
    public function actionCreate()
    {
        $model = new CovNews();

        if ($model->load(Yii::$app->request->post()) && $model->save()) {
            return $this->redirect(['view', 'id' => $model->id]);
        }

        return $this->render('create', [
            'model' => $model,
        ]);
    }
```

```
    public function actionUpdate($id)
    {
        $model = $this->findModel($id);

        if ($model->load(Yii::$app->request->post()) && $model->save()) {
            return $this->redirect(['view', 'id' => $model->id]);
        }

        return $this->render('update', [
            'model' => $model,
        ]);
    }

    public function actionDelete($id)
    {
        $this->findModel($id)->delete();

        return $this->redirect(['index']);
    }


    protected function findModel($id)
    {
        if (($model = CovNews::findOne($id)) !== null) {
            return $model;
        }

        throw new NotFoundHttpException('The requested page does not exist.');
    }
}
```

View

前台：数据传递到前台view，我们用php编写的view采用了多个css样式展示对应数据。由于篇幅限制，以其中一个news对应的div块来说明相应的数据块情况。

```
<div class="col-md-6">
    <div class="row">
        <?php foreach ($news2 as $news) : ?>
          <!--对应下层controller传入的news2，用php的foreach来进行循环展示对应数据 -->
            <div class="col-md-6 col-6 paddding animate-box" data-animate-effect="fadeIn">
                <div class="fh5co_suceefh5co_height_2"><img src="<?= $news->image ?>" alt="News Image">
```

```
                    <div class="fh5co_suceefh5co_height_position_absolute">
</div>
                    <div
class="fh5co_suceefh5co_height_position_absolute_font_2">
                        <div class=""><a href="#" class="color_fff"> <i
class="fa fa-clock-o"></i>  <?= $news->pubDate ?></a></div>
                        <div class=""><a href="<?= $news->sourceUrl ?>"
class="fh5co_good_font_2"> <?= $news->title ?> </a></div>
                    </div>
          <!--据不同的块，使用news的图片、发布日期、标题、链接、来源、内容等相应的数据，
都大体类似 -->
                </div>
            </div>
        <?php endforeach; ?>
    </div>
</div>
```

由此我们可以在news和research页面从数据库获取到对应的信息。

上：新闻块news1，下：新闻块news2

后台的view界面相对较简单，由gii生成，在后台也主要也是进行增删的操作，调用了**gridview**部件用于在网格中显示数据，以及controller提供的参数。

```
$this->title = 'News';
$this->params['breadcrumbs'][] = $this->title;
```

```
?>
<div class="news-index">

    <h1><?= Html::encode($this->title) ?></h1>

    <p>
       <?= Html::a('Create News', ['create'], ['class' => 'btn btn-success'])
?>
    </p>

    <?php Pjax::begin(); ?>

    <?= GridView::widget([
        'dataProvider' => $dataProvider,
        'filterModel' => $searchModel,
        'columns' => [
            ['class' => 'yii\grid\SerialColumn'],

            'id',
            'pubDate',
            'title',
            'summary',
            'infoSource',
            //'sourceUrl',
            //'image',

            ['class' => 'yii\grid\ActionColumn',
          'buttons' => [
                        'delete' => function ($url) {
                            return Html::a('Delete', $url, [
                                'data-method' => 'post',
                                'data-confirm' => 'Are you sure?'
                            ]);
                        }
                    ]],
        ],
    ]); ?>

    <?php Pjax::end(); ?>

</div>
```

4.1Video

对于Video 模块，有如下功能：

对于浏览型用户：

① 浏览由其他用户发表的与核污染有关的视频。

② 搜索与某个关键词相关的视频。

③ 对某个视频进行评论（需要注册登录网站）。

④ 喜欢或不喜欢某个视频。

⑤ 关注某个用户。

对于创作型用户：

④ 发表与核污染相关的视频（需要注册登录网站）。

⑤ 上传对应的封面图，选择相应的类别。

⑥ 管理用户自己的视频，可以进行更新、删除。

对于网站管理者：

④ 审核用户上传的视频内容，决定是否发表。（需要管理员登录网站后台）

⑤ 对所有用户上传的视频有更新、删除的权限。

我们采用MVC三层架构来完成前后端的交互。

Model

该部分位于common/models目录下的Video、VideoLike、VideoView有关的文件。首先介绍Video内容：Yii采用ORM（对象关系模型），即一个类对应数据库中的一张表。Video类在gii生成的ActiveRecord基础上扩展出来，以下成员表示视频是否可见。

```
        // 代表视频状态
        const STATUS_UNLISTED=0;
        const STATUS_PUBLISHED=1;
```

以下成员为上传的视频和缩略图文件的实例。

```
public $video;
public $thumbnail;
```

类中方法较多，这里给出一些比较重要的方法。

```
    // 定义数据验证规则。这些规则在保存模型之前应用，以确保数据的有效性。
    /**
     * {@inheritdoc}
     */
    public function rules()
    {
        return [
            [['video_id', 'title'], 'required'],
            [['description'], 'string'],
            [['status', 'has_thumbnail', 'created_at', 'updated_at',
'created_by'], 'integer'],
            [['video_id'], 'string', 'max' => 16],
            [['title', 'video_name'], 'string', 'max' => 255],
            [['tags'], 'string', 'max' => 512],
            [['video_id'], 'unique'],
            ['has_thumbnail','default','value'=>0],
            ['status','default','value'=>self::STATUS_UNLISTED],
            ['thumbnail','image','minWidth'=>1280],
            ['video','file','extensions'=>['mp4']],
            [['created_by'], 'exist', 'skipOnError' => true, 'targetClass' =>
User::className(), 'targetAttribute' => ['created_by' => 'id']],
        ];
    }
```

以上代码主要对模型中的数据进行验证，具体来说，即 `video_id` 和 `title` 字段是必填的。如果这些字段为空，模型将不会通过验证。 `description` 字段必须是字符串类型。之后的每个字段都必须满足对应的要求，实际上该方法是对数据表中的各个字段进行了约束，如果不满足约束则不会通过验证。

```
public function attributeLabels()
    {
        // 对应数据库中的每列信息
        return [
```

```
            'video_id' => 'Video ID',
            'title' => 'Title',
            'description' => 'Description',
            'tags' => 'Tags',
            'status' => 'Status',
            'has_thumbnail' => 'Has Thumbnail',
            'video_name' => 'Video Name',
            'created_at' => 'Created At',
            'updated_at' => 'Updated At',
            'created_by' => 'Created By',
            'thumbnail'=>'Thumbnail'
        ];
    }
```

以上代码将数据库中的每个字段映射到模型中。每个属性对应模型中的一个变量。

```
    public function save($runValidation = true, $attributeNames = null)
    {
        $isInsert = $this->isNewRecord;
        if ($isInsert) {
                if($this->video) {
                    try {
                        $this->video_id = Yii::$app->security-
>generateRandomString(8);
                    } catch (Exception $e) {
                    }
                    $this->title = $this->video->name;
                    $this->video_name = $this->video->name;
                }
        }
        if ($this->thumbnail) {
            $this->has_thumbnail = 1;
        }
        $saved = parent::save($runValidation, $attributeNames);
        if (!$saved) {
            return false;
        }
        if ($isInsert) {
            $videoPath = Yii::getAlias('@frontend/web/storage/videos/' . $this-
>video_id . '.mp4');
            if (!is_dir(dirname($videoPath))) {
                try {
                    FileHelper::createDirectory(dirname($videoPath));
                } catch (Exception $e) {
                }
```

```
            }
            $this->video->saveAs($videoPath);
        }
        if ($this->thumbnail) {
            $thumbnailPath = Yii::getAlias('@frontend/web/storage/thumbs/' .
$this->video_id . '.jpg');
            if (!is_dir(dirname($thumbnailPath))) {
                FileHelper::createDirectory(dirname($thumbnailPath));
            }
            $this->thumbnail->saveAs($thumbnailPath);
            Image::getImagine()
                ->open($thumbnailPath)
                ->thumbnail(new Box(1280, 1280))
                ->save();
        }

        return true;
    }
```

以上代码是该部分的核心代码，该方法用于处理视频数据的保存逻辑，包括视频文件和缩略图的上传。这个方法重写了 Yii2 中 ActiveRecord 类的 `save` 方法。具体逻辑则是：首先检查当前 `Video` 对象是否为新记录，如果是新记录并且视频文件存在，则进行如下操作：

- 使用 Yii 的安全组件生成一个随机的 8 字符长的字符串作为 `video_id`。
- 将视频文件的名称赋值给 `title` 和 `video_name` 属性。

之后处理缩略图，也是通过 $this->thumbnail 来检查是否存在缩略图，并设置相应的标志位。之后调用父类的保存方法，然后把视频和缩略图保存到指定的文件目录下。

```
    public static function getAll($pageSize = 5)
    {

        $query = Video::find()->published()->latest();

        $videos = $query
            ->limit($pageSize)
            ->all();

        $data['videos'] = $videos;

        return $data;
    }
```

以上方法用于获取所有已经发布的video视频，主要用find方法来找到所有视频，即构建一个数据库查询，并且需要是已发表的内容。之后用 `limit($pageSize)` 方法限制查询结果的数量（默认为5）。最后把查询结果（视频记录）存储在 `$data` 数组中的 `'videos'` 键下，并且返回该数据。

Controller

Controller分为前后台两个部分，分别位于 frontend/controllers/VideoController 和 backend/controllers/VideoController 中。这里主要以前台的VideoController为例进行代码分析和说明。

```
    public function actionCreate()
        {
            $this->layout='video';
            $model = new Video();

            $model->video = UploadedFile::getInstanceByName('video');

            if (Yii::$app->request->isPost && $model->save()) {
                return $this->redirect(['update', 'id' => $model->video_id]);
            }

            return $this->render('create', [
                'model' => $model,
            ]);
        }
```

以上代码主要用于创建新的 `Video` 实例。这里的 Video 即是在Model中所定义的类。该方法处理文件上传，保存视频数据，如果成功则重定向到编辑页面，否则显示创建表单。

- 首先使用 `UploadedFile` 类获取上传的文件。`getInstanceByName('video')` 方法尝试获取通过请求发送的名为 `'video'` 的文件。
- 然后检查是否是 POST 请求并且模型已经保存下相关信息。如果保存成功则重定向到 `update` 动作，同时传递新创建的视频的 `video_id` 作为参数。
- 如果不是 POST 请求或者保存失败，将渲染并返回 `create` 视图。同时将模型实例传递给视图，这样视图可以访问模型的数据和属性。

通过这一方法，可以看到Controller负责把数据保存在Model中，然后调用Model中的方法与数据库实际交互。之后选择一个视图进行显示。

```
    public function actionView($id)
        {
            $this->layout = 'main';
            $video = $this->findVideo($id);
```

```
            $videoView = new VideoView();
            $videoView->video_id = $id;
            $videoView->user_id = \Yii::$app->user->id;
            $videoView->created_at = time();
            $videoView->save();

            $comments=$video->getVideoComments();
            $commentForm= new CommentFormV();

            $similarVideos = Video::find()
                ->published()
                ->byKeyword($video->title)
                ->andWhere(['NOT', ['video_id' => $id]])
                ->limit(10)
                ->all();

            return $this->render('view', [
                'model' => $video,
                'similarVideos' => $similarVideos,
                'comments'=>$comments,
                'commentForm'=>$commentForm
            ]);
        }
```

以上代码用于处理用户查看视频详情的逻辑。首先通过调用 `findVideo` 方法来查找特定 ID 的视频。之后实例化一个videoview对象，并对成员变量进行赋值。最后通过save方法保存到数据库中。然后调用 `Video` 模型的 `getVideoComments` 方法获取当前视频的评论，创建一个新的评论表单模型实例并查找类似的视频（byKeyword)。最后调用 `render` 方法来渲染视图文件（ `'view'` ），并向视图传递视频模型 (`$video`)、类似视频列表 (`$similarVideos`)、评论列表 (`$comments`) 和评论表单模型 (`$commentForm`)，供试图层使用。后台部分的方法与前台有较多类似，在此不再赘述。

View

视图层文件较多，即一个网页页面对应一个php文件。这里以前台的video/home页面为例，页面显示如下图：


```
<?php


use yii\data\ActiveDataProvider;
use yii\helpers\Url;
$this->title = 'Video';
$this->params['breadcrumbs'][] = $this->title;

?>
    <form action="<?php echo Url::to(['/video/search']) ?>"
          class="form-inline my-2 my-lg-0">
        <input class="form-control mr-sm-2" type="search" placeholder="Search"
               name="keyword"
               value="<?php echo Yii::$app->request->get('keyword') ?>">
        <button class="btn btn-outline-success my-2 my-sm-0">Search</button>
    </form>
    <div>
        跟随新闻报道，让我们一起关注核污染，守护地球家园!
    </div>


<?php
echo \yii\widgets\ListView::widget([
    'dataProvider'=>$dataProvider,
    'pager' => [
        'class' => \yii\bootstrap4\LinkPager::class,
    ],
    'itemView'=>'video_item',
    'layout'=>'<div class="d-flex flex-wrap">{items}</div>{pager}',
    'itemOptions'=>[
```

```
            'tag'=>false
        ]
    ])?>
```

将代码与页面显示相对应。视图代码将PHP与HTML混合，首先是一个 HTML 表单，用于视频搜索。表单提交到 `/video/search` 路径（由 `Url::to` 方法生成）。包含一个搜索框，其中 `value` 被设置为当前请求的 `keyword` 参数值，在搜索后保留用户输入的关键词。有一个Search按钮来执行搜索操作。之后插入一个div来显示标语：跟随新闻报道，让我们一起关注核污染，守护地球家园！然后使用 Yii 的 `ListView` 小部件来显示视频列表，即搜索结果。

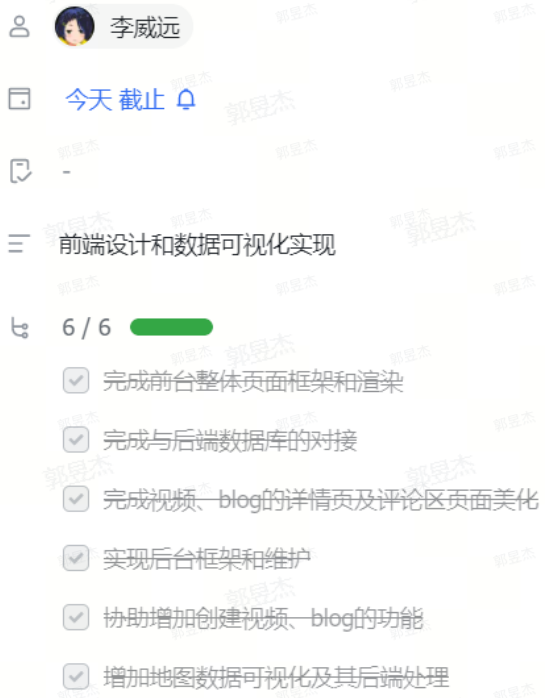
总结

综上，我们可以看到一个完整的MVC架构在网页开发中的应用。

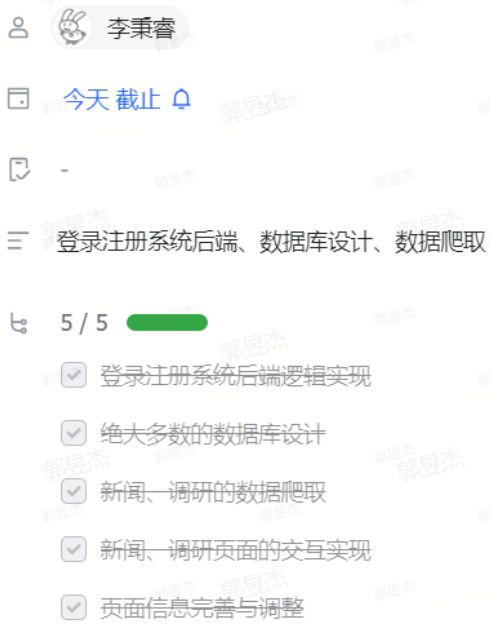
1. 模型（Model）：
   - 负责数据和业务逻辑。
   - 在 Yii2 中，模型通常是 `ActiveRecord` 的扩展，它表示数据库中的一个表，并提供了数据的增删改查（CRUD）操作。
2. 视图（View）：
   - 用于生成用户界面，负责呈现数据。
   - 通常包含 HTML 标记和一些基本的 PHP 代码来显示模型数据。
3. 控制器（Controller）：
   - 是模型和视图之间的中介。
   - 控制器接收用户的输入（例如，通过表单提交），并调用模型去处理这些输入（如保存数据），然后选择一个视图来显示相应的输出（例如，显示一个更新后的页面或者是一个错误消息）。

成员任务完成情况

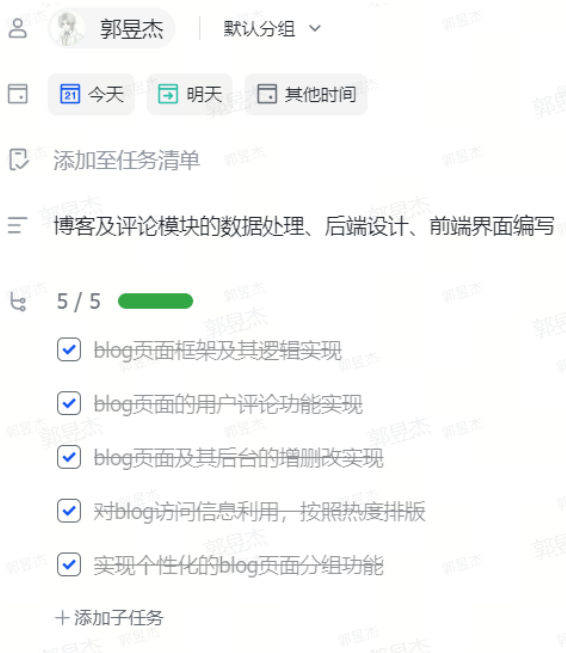
组长：李威远


李威远
今天 截止
-
前端设计和数据可视化实现
6 / 6
完成前台整体页面框架和渲染
完成与后端数据库的对接
完成视频、blog的详情页及评论区页面美化
实现后台框架和维护
协助增加创建视频、blog的功能
增加地图数据可视化及其后端处理

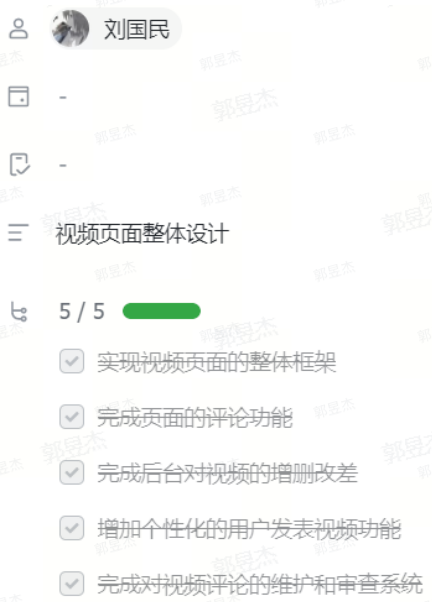
组员：李秉睿


李秉睿
今天 截止
-
登录注册系统后端、数据库设计、数据爬取
5 / 5
登录注册系统后端逻辑实现
绝大多数的数据库设计
新闻、调研的数据爬取
新闻、调研页面的交互实现
页面信息完善与调整

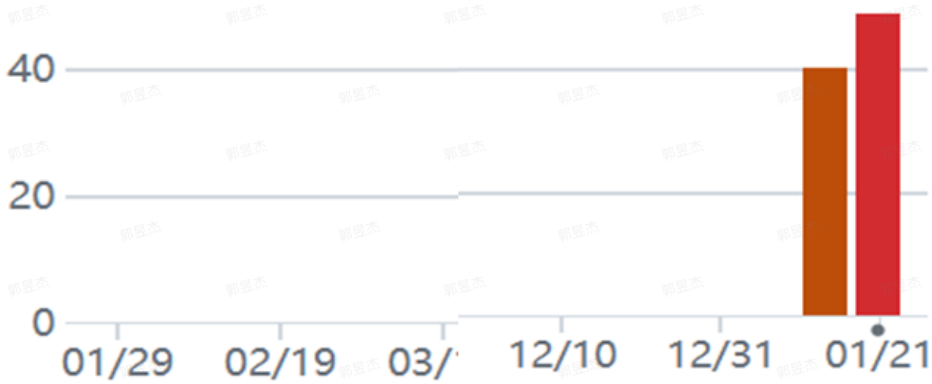
组员：郭昱杰



组员：刘国民



Git提交记录



迫于课业压力而修整：由于小队成员均为大三上学期，大家的课业都很大，在期末月没能开工，一直搁置了。

重振旗鼓后直追猛进：考完试的这段日子以来，小组成员昼夜不息，不是吃饭就是学习yii2框架、研究代码、四个人共同努力，补齐了之前被课业压力夺去的不足，上百次的git提交记录，是9天时间内我们创造的奇迹。