# 计算机与网络空间安全学院互联网数据库开发大作业 实现文档

姓名：万泽生 黄韵竹 刘扬

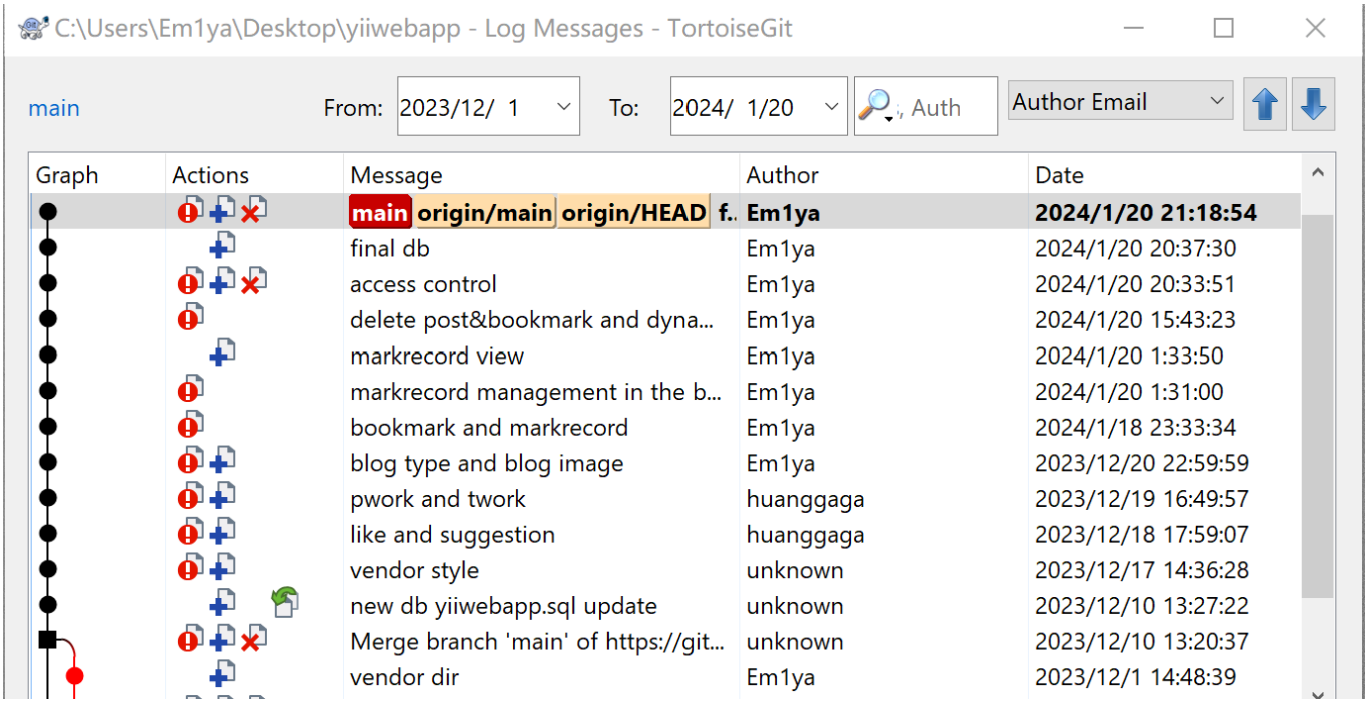
学号：2014081 2011064 2011839

专业：信息安全、法学双学位

## 一、任务分工

- 万泽生：参与前期讨论、参与数据库设计、完成前后台模板的收集、完成后台管理功能的实现、完成文章功能除点赞外的开发与调试；完成登陆的实现；设计访问控制。完成部署文档, 需求文档。
- 黄韵竹: 参与前期讨论，参与数据库设计。完成个人作业下载、团队作业下载开发与调试；完成点赞，取消点赞功能的开发与调试。完成用户手册, 参与制作项目展示 PPT。
- 刘扬：参与前期讨论，参与数据库设计。完成首页展示的开发与调试；团队介绍界面的开发与调试。参与制作项目展示ppt。

## 二、代码管理工具git




说明：其中Em1ya为万泽生，huanggaga为黄韵竹，而unknown为刘扬。

## 三、主要代码以及目录展示

### （一）整体结构展示

```
assets/             静态资源访问方式
commands/           一个示例controller
```

| | |
|---|---|
| config/ | yii应用的设置文件 |
| controllers/ | 控制器的php文件 |
| mail/ | 电子邮件的layout |
| models/ | 模型的php文件 |
| runtime/ | yii运行时生成文件 |
| tests/ | 一些示例 |
| vendor/ | yii第三方php文件 |
| views/ | 视图的php文件 |
| web/ | 入口脚本以及用户请求的网络资源 |

## （二）编写代码文件展示

对于yii2代码的编写集中在MVC中，其他部分虽然也有配置修改代码，但是不影响主体功能的实现，不做赘述。

### 1、controller

| | |
|---|---|
| /MarkrecordController.php | 对收藏记录进行增删改查的控制器（管理员） |
| /MessageController.php | 对私信消息进行增删改查的控制器（管理员） |
| /PollutionController.php | 对动态地图污染信息进行增删改查的控制器（管理员） |
| /PostController.php | 对文章博客进行增删改查的控制器（管理员） |
| /RegionController.php | 对地区进行增删改查的控制器（管理员） |
| /SiteController.php | 前台首页，登录，前台留言展示博客，文章阅读的控制器 |
| /SuggestionController.php | 对前台留言进行增删改查的控制器（管理员） |
| /UserController.php | 对用户信息进行增删改查的控制器（管理员） |
| /UserspaceController.php | 前台用户的个人空间，对用户自身的文章，收藏，私信进行管理的控制器 |
| /BookmarkController.php | 对收藏夹进行增删改查的控制器（管理员） |
| /CategoryController.php | 对文章类型进行增删改查的控制器（管理员） |
| /CommentController.php | 对文章的评论进行增删改查的控制器（管理员） |

### 2、model

注：大部分的文件都是由Gii自动生成的，所以带有`ModelSearch.php`就是Gii自动生成的用于包装模型数据库操作的类型，只要知道了`Model`所代表的模型含义即可，下面仅仅介绍本项目中起到作用的`Model`文件。

| | |
|---|---|
| /Bookmark.php | 文章收藏夹模型 |
| /Category.php | 文章分类模型 |
| /Comment.php | 文章评论模型 |
| /Likes.php | 用户点赞模型 |
| /LoginForm.php | 登录表单模型 |
| /Markrecord.php | 文章收藏记录模型 |
| /Message.php | 私信消息模型 |
| /Pollution.php | 核污染记录模型 |
| /Post.php | 文章模型 |
| /Region.php | 地区模型 |

```
/Suggestion.php                 前台留言模型
/User.php                       用户模型
```

**3、view**

注：生成的php文件太多，仅仅展示controller对应的目录以及布局目录，不会详细展示所有的php文件。

```
/bookmark                       收藏夹管理页面（后台）
/category                       文章类型管理页面（后台）
/comment                        文章评论管理页面（后台）
/layouts                        页面模板布局文件
/likes                          点赞管理页面（后台）
/markrecord                     收藏记录管理页面（后台）
/message                        私信消息管理页面（后台）
/pollution                      核污染数据管理页面（后台）
/post                           文章管理页面（后台）
/region                         地区管理页面（后台）
/site                           首页、前台留言、作业、团队展示、文章展示等页面
/suggestion                     前台留言管理页面（后台）
/user                           用户管理页面（后台）
/userspace                      用户个人空间页面
```

# 四、实现展示

注：该处仅仅挑选重要的代码进行展示后台代码基本上是Gii生成的CURD代码，在此不作赘述。

## （一）模板布局的实现

本次实验使用的模板为：后台模板tabler-1.0.0-beta16 前台模板Anchor-Bootstrap-UI-Kit-master。首先要将模板要生成的html代码放在`view/layout`目录下，然后修改`AppAsset`并给所有的`controller`类的`action`中都加载。 但是实际上发现，该原理有赖于加载前台模板时的工作目录，如果能将静态资源摆放在layout所访问的目录下，即使不调整`AppAsset`也可以实现静态资源的加载，关于获得工作目录，一个比较tricky的办法是在layout中执行半句话木马（笑）`eval(system('dir'));`，然后观察目录即可。核心代码如下：

```
 public function actionLogin(){
  $this->layout="loginlayout";
  ...
}
```

## （二）登录模板的移植

yii2本身存在一套自己的登陆模板，通过`User.php`中存在的静态成员进行登录，其认证过程稍显复杂，而且不涉及数据库访问，但是该套系统牵扯了yii2本身的很多机制（代码生成等等）。经过取舍后决定直接将yii2本身的用户登录与数据库访问以及我们自己的数据表设计结合。
首先，Gii生成的数据类型为`ActiveRecord`，而原生`User`为`IdentityInterface`，所以先将Gii生成的代码修改声明，使其实现`IdentityInterface`的功能。

```
class User extends \yii\db\ActiveRecord implements \yii\web\IdentityInterface
```

在这之后需要给身份验证接口以新的实现，下面是原有yii2的认证接口的实现：

```
    private static $users = [
          '100' => [
              'id' => '100',
              'username' => 'admin',
              ...
      ];
      ...
      public static function findIdentityByAccessToken($token, $type = null){
          foreach (self::$users as $user) {
              if ($user['accessToken'] === $token) {
                  return new static($user);
              }
          }
          return null;
      }
      ...
```

可以发现，原有的User实现中使用的$users的部分都需要我们去替代成从数据库中取数据而不是从静态成员中取得，所以直接使用find方法即可。修改代码如下：

```
      ...
      public static function findIdentityByAccessToken($token, $type = null){
          $user=User::findOne(['accessToken'=>$token]);
          if($user==null){
              return null;
          }
          return new static($user);
      }
      ...
```

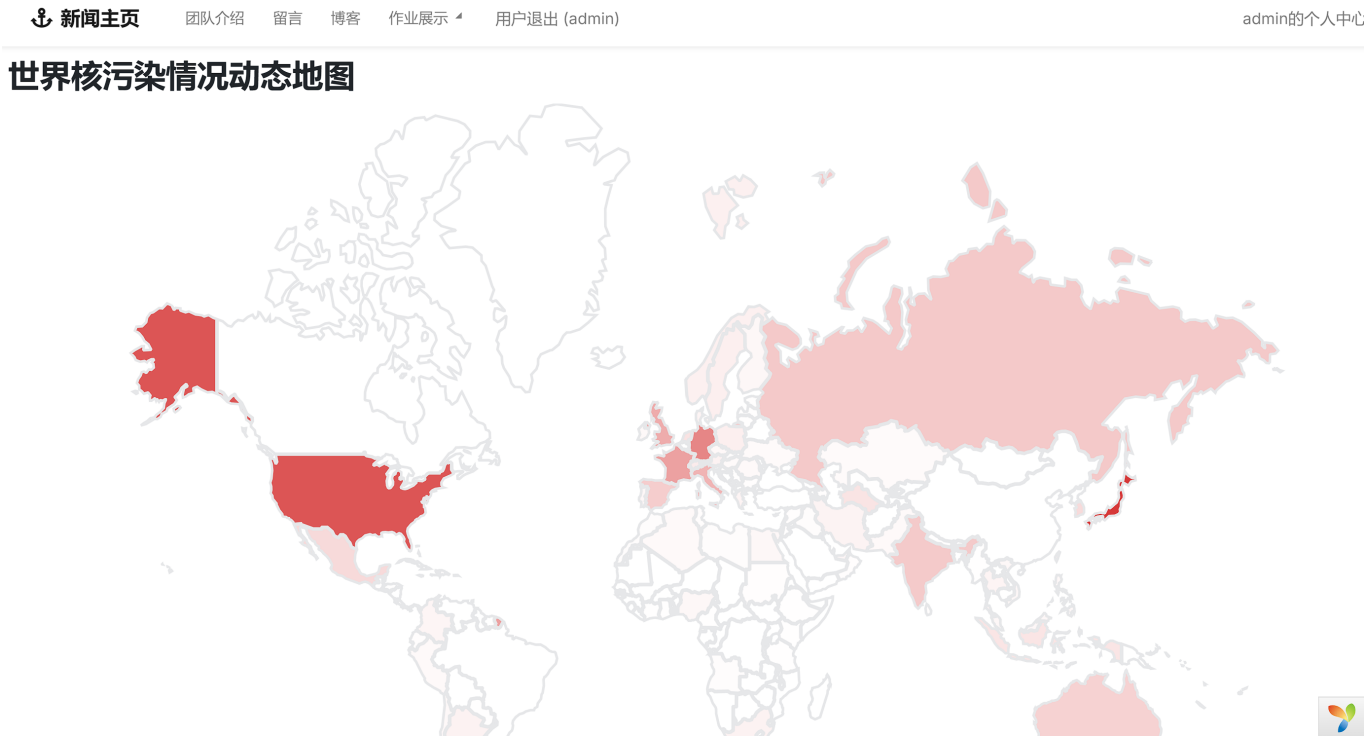
这样，经过登陆后，我们就可以从Yii::$app->user->identity->user_id中取得用户的实时用户id，而不用再让用户传递自己的身份而造成横向越权的情况。

### （三）动态地图

本项目选择使用jVectorMap来实现动态世界地图，所谓动态地图，不是指地图本身有动效，而是数据来源动态化。动态地图的代码如下（省略了js文件的引入）,通过设置regionStyle中的格式来设置默认的染色，边界颜色及宽度，visualizeData中的values是键值数组，以{'Region Code':'Pollution data'}的形式进行显示，通过遍历从数据库中传来的污染数据即可动态填充该地图。

```
<script>
  // @formatter:on
  document.addEventListener("DOMContentLoaded", function() {
    const map = new jsVectorMap({
        selector: '#map-world-merc',
        map: 'world_merc',
        backgroundColor: 'transparent',
        regionStyle: {
            initial: {
                fill: tabler.getColor('bg-surface'),
                stroke: tabler.getColor('border-color'),
                strokeWidth: 2,
            }
        },
        zoomOnScroll: false,
        zoomButtons: false,
        // -------- Series --------
        visualizeData: {
            scale: [tabler.getColor('bg-surface'), tabler.getColor('danger')],
            values: {
                <?php foreach($pollution_data as $key=>$record):;?>
                    <?= Html::encode($key); ?>:<?= Html::encode($record) ?>,
                <?php endforeach?>
            },
        },
    });
    window.addEventListener("resize", () => {
        map.updateSize();
    });
  });
  // @formatter:off
</script>
```

效果如下：




## （四）文章系统的实现

文章系统是该新闻站的核心功能，在本站具有了基本的外观框架，首页地图，以及登陆系统后，就应该实现最核心的用户功能——文章。围绕新闻站的文章操作，我们将其分为以下几个生命周期：文章发表-文章操作-文章销毁。

**文章发表**：文章发表功能被实现在用户的个人空间内，用户通过在个人空间内填写表单，利用POST方式传递参数，从发表文章操作角度来看，数据的流动方向是V-C-M。首先是表单的设计：

```
    $form = ActiveForm::begin([
        'id' => 'comment-form',
        'action' => ['userspace/postcreate'],
        'options' => ['enctype' => 'multipart/form-data']
        ,]);
    ?>
    <?= $form->field($post, 'post_title')->textInput(['autofocus' => true]) ?>
    <?= $form->field($post, 'post_type')->textInput() ?>
    <?= $form->field($post, 'post_text',['inputOptions' => [
        'style' => 'height: 220px;',
    ]])->textarea() ?>
    <?=$form->field($post, 'post_image')->fileInput();?>
            <?= Html::submitButton('发送', ['class' => 'btn btn-primary', 'name'
=> 'comment-button']) ?>
    <?php ActiveForm::end();
```

用户输入文章的标题，类型，内容，上传封面图片，然后发送请求。在接收端（Controller），大部分的属性已经填写完毕，但是图片我们决定不以blob的方式存储（blob的存和解析还需要额外编码，封面图片不具有机密性，直接存储到文件系统中即可）。

这里比较triky的地方是上传的图片直接存储在了post_image成员中，该成员本来是路径，但是在这之前先让其暂时存储传输实体，不会再额外写上传代码（笑）。

```
    public function actionPostcreate(){
        ...
        if($this->request->isPost){
            if($post->load($this->request->post())){
                $file=UploadedFile::getInstance($post, 'post_image');
                if($file){
                    $filePath = 'frontendassets/img/blogimage/'
.'blog'.count(Post::find()->all()). '.' . $file->extension;
                    $file->saveAs($filePath);
                    $post->post_image = $filePath;
                }
                $post->post_author=Yii::$app->user->identity->user_id;
                $post->post_time=date('Y-m-d H:i:s');
                if($post->save()){
                    return $this->redirect(['userspace/index']);
                }else {
                    Yii::$app->session->setFlash('error', '文章发表失败');
                    return $this->refresh();
                }
            }
        }
        ...
    }
```

可以看难道，对于上传的文件实体，我们进行重命名后转存，并将路径重新写入文章对象的post_image成员，便于将来加载，存储即可。

**文章的处理**：文章的处理主要是指用户浏览文章时对于文章的点赞，收藏，评论这三种方式，三种方式都是用户上传文章信息，表示自己要对哪个文章进行操作（用户信息经过User.php的魔改之后可以直接提取，不用用户手动传输）。
这三种同质的功能存在着一定的不同，不同在于评论需要额外传输评论内容，而点赞需要考虑是否已经点赞，是否需要撤销点赞，收藏不仅需要上传用户指定的收藏夹，也要考虑是否已经收藏的情况，并且收藏行为直接与收藏夹有关，而不直接与用户有关，多以需要多表联合查询。
对比如下：

```
return $this->render('passage', [
            'blogpost' => $blogpost,
            'author'=>$user  ,
            'comment'=>$comment ,
            'old_comments'=>$old_comments,
            'like'=>$like,
            'old_likes'=>$old_likes,
            'if_liked'=>$if_liked,
            'markr'=>$markr,
            'old_marks'=>$bookmarks,
```

```
            'if_marked'=>$if_marked
        ]);
```

从传送给View的数据我们就可以看出，Comment评论只需要传送一个待填充的空对象，以及显示老的评论即可，而Likes点赞还要多传一个是否点赞的成员来标识，收藏Markr传送的空对象是收藏记录，也需要传送是否被收藏，而传送的是已经存在的收藏夹而不是收藏记录，因为用户在收藏时会看到的选项是收藏夹而不是记录。

**文章的删除**：前文所述，文章的处理与评论、点赞、收藏都有关，那么在删除文章时，就不能只删除文章本身，否则用户的个人空间内将不能够正确管理自己的评论点赞对象（也可以直接搞成删除后显示not set，但是要写太多的条件判断，不如删干净便捷）。 在这里没有使用大量的多表查询，在Post文章类中实现了getter后，getter的返回结果会作为该类的**虚拟成员**，使用hasOne或者hasMany直接获得有关表的成员，然后删除有关的点赞、评论、收藏记录即可。

```
    public static function deletePoatWithOther($post_id){
        $post=Post::findOne(['post_id'=>$post_id]);
        if(Post::findOne(['post_id'=>$post_id])->delete()){
            foreach($post->comments as $comment){
                Comment::findOne(['comment_id'=>$comment->comment_id])->delete();
            }
            foreach($post->likes as $like){
                Likes::findOne(['like_post'=>$like->like_post,'like_user'=>$like-
>like_user])->delete();
            }
            foreach($post->markrecords as $markrecord){
                Markrecord::findOne(['mark_id'=>$markrecord-
>mark_id,'post_id'=>$markrecord->post_id])->delete();
            }
        }
    }
```

## （五）访问控制实现

yii2本身具有基于RBAC的访问控制机制，但是这种访问控制机制涉及新的数据表，而在之前组内讨论时就在User数据表中添加了user_type的角色字段，所以在这里放弃使用yii2自己的访问控制，由自己实现RBAC的效果。

**纵向越权防护**：纵向越权指低权限用户无法访问高权限功能，在这里将权限分为未登录用户、已登录普通用户、已登录管理员，权限由低到高。未登录用户仅可以查看首页以及博客列表，已登录普通用户可以访问除后台功能外的一切功能，而管理员可以访问所有的功能。

为了限制访问权限，我们需要在相关的Controller开头加上权限检查函数，在SiteController和UserspaceController中需要加入的是对前两种用户的验证区分，实现如下：

```
    public static function checkuser(){
        if(Yii::$app->user->identity==null){
            return false;
        }
```

```
            return true;
        }
```

而其它的后台功能都需要管理员权限，检查如下：

```
    public function checkadmin(){
        if(Yii::$app->user->identity==null){
            return false;
        }
        $user=User::findOne(['user_id'=>Yii::$app->user->identity->user_id]);
        if($user->user_type==="admin"){
            return true;
        }
        return false;
    }
```

**横向越权防护**：横向越权指用户可以通过抓包-修改-重放的方式使自己的权限影响到其它用户，对系统进行分析，用户在大多数功能实现时并不需要上传自己的身份，后台会根据`Yii::$app->user->identity->user_id`自动填写用户真正的身份，而仅仅当存在两个时候有横向越权：

- 用户删除文章时，仅仅传递要删除的文章id
- 用户删除收藏夹和收藏记录时，仅仅指定收藏夹或者收藏记录

所以在实现时，在这几个时候需要检查文章/收藏夹/收藏记录是否是该用户的，以删除文章为例代码如下：

```
        if($this->request->isPost){
            $post_data = $this->request->post();
            $post=Post::find(['post_id'=>$post_data['post_id']])->one();
            //检查横向越权，检查删掉的文章id是否为作者的
            if($post==null || $post->post_author!=Yii::$app->user->identity->user_id){
                $this->redirect(['userspace/post']);
            }
            if(array_key_exists('post_id',$post_data)){
                Post::deletePoatWithOther($post_data['post_id']);
            }
        }
```

**其它安全issue**：

- debug界面和Gii为了后期开发方便暂时没有关闭，上线之前应该关闭。
- 虽然修改了上传文件名，但是没有限制上传的文件类型，会造成严重的文件上传漏洞。