

# 实现文档

## 前期规划

### 互联网数据库

🕒 12月17日 (周日) 09:55 - 10:20 | 24 分 45 秒

① 会议 ID: 150 376 860

👤    

☰ [在日历中查看详情](#)

#### 参会人统计

[导出](#)

12月17日

10:20 离开会议

09:55 加入会议

## 会议记录

👤 梁晓储 | 12月17日创建

### 议程 1：讨论分工和后续任务 2023/12/17

#### 讨论过程：

- 将大作业分割成了不同模块
- 每个人对模块进行了认领
- 确定了分工，具体如下
  - 前端 
  - 后端 
  - 数据库 
  - 爬虫 

#### 后续安排：

- ☒ 确定每个人的任务进度，调整任务分工
  -  梁晓储  方奕 本周一 21:00

在项目实现之前进行了一次会议讨论，会议确认了网站是由 `vue` 和 `yii2` 组成前后端分离的形式，降低了部分组员的学习成本

同时由组长将整个项目的实现进行模块化拆分，为每位组员分配任务

## 任务分工

### 2110951 梁晓储

- 全权负责Vue前端代码
- 前后端连接的实现
- 后端api控制器的框架编写
- 部分后端api函数的调整

### 2112106 方奕

- Yii2后端的构建
- 后端与Mysql数据库的连接
- 独自完成Yii2中models的编写，实现表项的传入
- 独自数据库在后端的CRUD功能
- 完成了大部分后端api函数
- 后端主页的设计

### 2113419 张昊星

- 独自完成数据库的设计与构建
- 为每个表项进行主键设置与外键连接
- 实现表与表之间联动的触发器
- 数据验证与处理

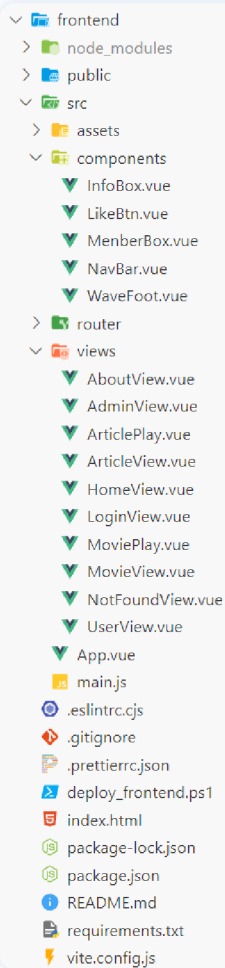
- 部分后端api函数的调整

## 2112414 王思宇

- 完成了爬虫模块的编写，为数据库提供了大量数据
- 负责网页数据内容的查找
- 完成了每个文档的大部分内容
- 负责PPT制作

### 文件目录展示

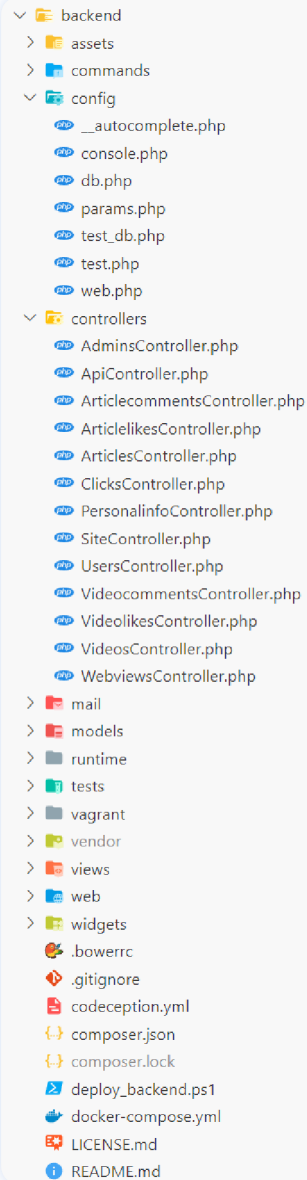
### 前端目录



```
frontend
├── node_modules
├── public
├── src
│   ├── assets
│   ├── components
│   │   ├── InfoBox.vue
│   │   ├── LikeBtn.vue
│   │   ├── MemberBox.vue
│   │   ├── NavBar.vue
│   │   └── WaveFoot.vue
│   ├── router
│   └── views
│       ├── AboutView.vue
│       ├── AdminView.vue
│       ├── ArticlePlay.vue
│       ├── ArticleView.vue
│       ├── HomeView.vue
│       ├── LoginView.vue
│       ├── MoviePlay.vue
│       ├── MovieView.vue
│       ├── NotFoundView.vue
│       └── UserView.vue
├── App.vue
├── main.js
├── .eslintrc.cjs
├── .gitignore
├── .prettierrc.json
├── deploy_frontend.ps1
├── index.html
├── package-lock.json
├── package.json
├── README.md
├── requirements.txt
└── vite.config.js
```

其中views是每个页面的代码， components是网页代码可以嵌入的模块， App.vue是项目的主入口

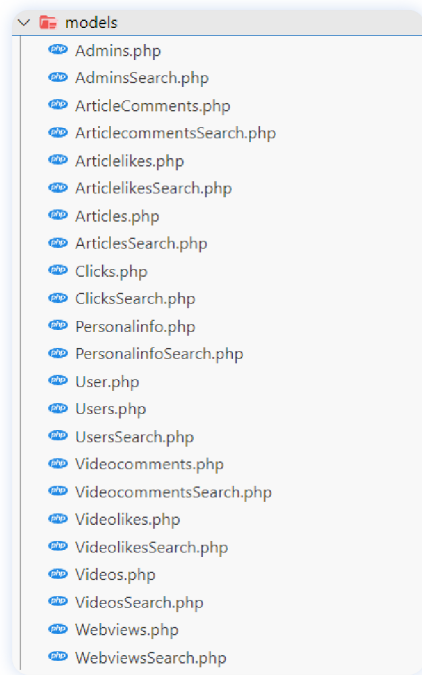
## 后端目录



```

└─ backend
  ├── assets
  ├── commands
  ├── config
  │   ├── __autocomplete.php
  │   ├── console.php
  │   ├── db.php
  │   ├── params.php
  │   ├── test_db.php
  │   ├── test.php
  │   └── web.php
  ├── controllers
  │   ├── AdminsController.php
  │   ├── ApiController.php
  │   ├── ArticlecommentsController.php
  │   ├── ArticlelikesController.php
  │   ├── ArticlesController.php
  │   ├── ClicksController.php
  │   ├── PersonalinfoController.php
  │   ├── SiteController.php
  │   ├── UsersController.php
  │   ├── VideocommentsController.php
  │   ├── VideolikesController.php
  │   ├── VideosController.php
  │   └── WebviewsController.php
  ├── mail
  ├── models
  ├── runtime
  ├── tests
  ├── vagrant
  ├── vendor
  ├── views
  ├── web
  ├── widgets
  ├── .bowerrc
  ├── .gitignore
  ├── codeception.yml
  ├── composer.json
  ├── composer.lock
  ├── deploy_backend.ps1
  ├── docker-compose.yml
  ├── LICENSE.md
  └── README.md

```



config 文件夹包含了应用程序的配置文件，用于配置不同环境下的运行参数。

controllers 文件夹用于存放控制器类文件，控制器负责处理用户请求并响应相应的操作。

models 文件夹主要用于存放模型类文件，模型类是应用程序与数据库表之间的映射

## 主要代码展示

### 前端部分

前端代码最主要的部分就是与后端交互并且将后端传入的数据写入网页中

本项目使用的是axios，它能够实现异步请求管理以及支持并发请求。以其中一个函数为例：

```
1  getUrl() {  
2      const id = this.$route.params.id  
3      axios
```

```

4      .post('http://localhost:8080/api/getarticle?id=' +
      id)
5      .then((response) => {
6          this.title = response.data.Title
7          this.content = response.data.Content
8          this.articleTime = response.data.PublicationDate
9          console.log(this.title)
10     })
11     .catch((error) => {
12         console.error('请求数据失败', error)
13     })
14 },

```

这个函数首先获取当前路由，并且将其作为参数通过 `post` 请求发送给后端提供的api，后端的api接收到这个请求后就会回应给前端一个response，经过后端的处理，这个response会以一个json的形式供前端读取。将json中一些量赋值给前端页面中定义的变量，就完成了前后端的数据交互。

## 后端部分

为了实现与前端的交互，后端首先要解除 cors 令牌，因为yii2框架有一个机制，每个请求都会先传入这个令牌再进行处理，不解除的话前端发送过来的跨域请求都会被yii2拒绝

```

1  // #config/web.php
2  'components' => [
3      'request' => [
4          'enableCsrfValidation' => false
5      ],
6  ]
7
8  'response' => [
9      'charset' => 'UTF-8',

```

```

10     'on beforeSend' => function ($event) {
11         $response = $event->sender;
12         $response->headers->set('Access-Control-Allow-
Origin', '*');//设置跨域请求
13         $response->headers->set('Access-Control-Allow-
Methods', 'GET, POST, PUT, DELETE, OPTIONS');
14         $response->headers->set('Access-Control-Allow-
Headers', 'Origin, X-Requested-With, Content-Type, Accept');
15     },
16 ],

```

开放了跨域请求后，还需要设置对前端获取数据提供api接口

```

1  // #config/web.php
2  'urlManager' => [
3      'enablePrettyUrl' => true,
4      'showScriptName' => false,
5      'rules' => [
6          'api/login' => 'api/login',
7          'api/signup' => 'api/signup',
8          'api/adminlogin' => 'api/adminlogin',
9          'api/getarticle' => 'api/getarticle',
10         'api/getvideo' => 'api/getvideo',
11         'api/getvideocomment' => 'api/getvideocomment',
12         'api/getarticlecomment' => 'api/getarticlecomment',
13         'api/getclick' => 'api/getclick',
14         'api/addvideocomment' => 'api/addvideocomment',
15         'api/addarticlecomment' => 'api/addarticlecomment',
16         'api/addclick' => 'api/addclick',
17         'api/getpersonalinfo' => 'api/getpersonalinfo',
18         'api/addwebviews' => 'api/addwebviews',
19         'api/getvideopagecount' => 'api/getvideopagecount',
20         'api/getarticlepagecount' =>
'api/getarticlepagecount',
21         'api/checkwebviews' => 'api/checkwebviews',
22         'api/getvideolikes' => 'api/getvideolikes',
23         'api/getarticlelikes' => 'api/getarticlelikes',

```

```

24         'api/addvideolikes' => 'api/addvideolikes',
25         'api/addarticlelikes' => 'api/addarticlelikes',
26     ],
27 ],

```

需要在urlManager的rules中配置所需要的api接口，其中左侧的 `api/##` 代表着前端访问的路由为 `/api/##`

，右侧的 `api/##` 代表着Yii2会去ApiController.php中调用 `action##` 函数来处理本次请求。

以前端获取视频为例，本次请求对应着 `'api/getvideo' => 'api/getvideo'` 语句，前端会使用 `axios` 向 `http://localhost:8080/api/getvideo` 发送请求，随后Yii2后端会调用 `actionGetvideo()` 来处理数据并且将处理好的数据传递给前端

```

1  // #controllers/ApiController.php
2  namespace app\controllers;
3  use app\models\Videos;
4
5  public function actionGetvideo()
6  {
7      \Yii::$app->response->format =
        \yii\web\Response::FORMAT_JSON;
8
9      // 获取页数
10     $page = \Yii::$app->request->get('page');
11     $intpage = (int)$page;
12     $id = \Yii::$app->request->get('id');
13
14     if ($id !== null) {
15         // 如果有 id 参数，则查询指定 VideoID 的视频信息
16         $videos = Videos::find()->select(['VideoID',
            'Title', 'Description', 'PictureURL', 'UploadDate',
            'VideoURL'])->where(['VideoID' => $id])->one();
17     } else {
18         // 否则按照原来的逻辑查询分页数据

```



```

19         $videos = Videos::find()→select(['VideoID',
        'Title', 'Description', 'PictureURL', 'UploadDate',
        'VideoURL'])→offset(18 * ($intpage - 1))→limit(18)-
        >all();
20     }
21
22     // 格式化为 JSON 并返回
23     return $videos;
24 }

```

## 爬虫部分

本组选取的是央视视频网对视频进行抓取，中国新闻网对文章进行抓取。以最复杂的视频抓取为例

首先我们获取到了样式视频网对搜索请求的api，其中page参数为页数，qtext\_str参数为‘核污染’关键字

```

1     "https://search.cctv.com/ifsearch.php?page=
    {page}&qtext=%E6%A0%B8%E6%B1%A1%E6%9F%93&sort=relevance&pageS
    ize=20&type=web&vtime=-1&datepid=1&channel=&pageflag=1&qtext_
    str=%E6%A0%B8%E6%B1%A1%E6%9F%93"

```

随后使用requests和BeautifulSoup对返回内容进行获取，再使用json.loads对网页内容格式化

```

1     response = requests.get(url)
2     soup = BeautifulSoup(response.text, "html.parser")
3     json1 = json.loads(soup.text)

```

随后到了数据处理部分

```

1  items = json1["list"]
2  for item in items:
3      item["all_title"] = re.sub(r"^\[[]*\]", "",
4      item["all_title"])
5      a = {
6          "id": item["id"],
7          "title": item["channel"],
8          "content": item["all_title"],
9          "updatetime": item["uploadtime"],
10         "picurl": item["imglink"],
11     }
12     parsed_url = urllib.parse.urlparse(item["imglink"])
13     path = parsed_url.path
14     # 获取最后一个斜杠之前的子串
15     last_slash = path.rfind("/") + 1
16     # 获取最后一个斜杠之后的子串
17     last_part = path[last_slash:]
18     videoId = last_part.split("-")[0]

```

到了最重要的部分，我们后续发现搜索请求的api中并没有视频的播放源链接，不过经过组员对网页内容的分析和请求爬取，找到了视频源请求的api

<https://vdn.apps.cntv.cn/api/getHttpVideoInfo.do?pid=>

再次根据这个url进行数据抓取就获取到了所有视频信息

```

1  url = "https://vdn.apps.cntv.cn/api/getHttpVideoInfo.do?pid="
2  + videoId
3  response1 = requests.get(url)
4  json2 = json.loads(response1.text)
5  b = json2["video"]["chapters4"][0]["url"]
6  a["videourl"] = b

```

获取到视频信息后，还需要将信息写入数据库中，我们使用了pymysql库进行操作

```
1  # 创建数据库连接
2  connection = pymysql.connect(
3      host="localhost",
4      user="root",
5      password="root",
6      db="internetdatabasedevelopment",
7  )
8
9  # 创建游标对象
10 cursor = connection.cursor()
11
12 for movie in movieList:
13     query = f"INSERT INTO videos (VideoID, Title,
14     Description, PictureURL, VideoURL, UploadDate) VALUES
15     ('{movie['id']}', '{movie['title']}', '{movie['content']}',
16     '{movie['picurl']}',
17     '{movie['videourl']}', '{movie['updatetime']}')"
18     cursor.execute(query)
19
20 # 事务提交
21 connection.commit()
22
23 #关闭连接
24 connection.close()
```

首先进行Mysql的连接和数据库的选择，然后创建游标对象，再根据这个游标对象完成查询语句，最后提交到数据库中并关闭连接

飞书任务分工截图

✓ 任务已完成



## 数据爬取

创建于会话: [互联网数据库开发小组](#)

👤 王思宇

📅 12月20日 截止

📁 互联网数据库开发 默认分组

+ 添加至任务清单

☰ 利用一些爬虫库对核污染相关的资料进行爬取，以便填充进web中

📌 5 / 5

- ☒ 选取方便爬取的网站
- ☒ 编写能够正常连接到网站的爬虫程序
- ☒ 爬取相关内容
- ☒ 设计爬取文件的存储路径
- ☒ 编写能够将爬取内容存储进mysql数据库的程序



+ 添加子任务

📎 添加附件

## 评论

○ [梁晓倩](#) 创建了任务 12月17日 10:36

输入评论

Aa 😊 @ 📎 ✎ | ➡

👤 2人关注 +

✓ 任务已完成



## Yii2后端设计

创建于会话: [互联网数据库开发小组](#)

👤 方奕

📅 12月20日 截止 🔔

📋 添加至任务清单

☰ 参考[Yii 2.0 权威指南 - 文档 - Yii Framework 中文网 \(yiichina.com\)](#), 了解yii2框架, 实现能够对mysql进行crud并且与前端对接的后端

📌 7 / 7

☰ ☒ 部署yii2并且上传到git (正确使用.gitignore, 避免上传多余的文件) 📄 👤 >

☒ 实现对mysql的写入

☒ 实现对mysql的查询

☒ 实现对mysql的修改

☒ 实现对mysql的删除

☒ 能够将查询到的数据返回封装成json返回给前端 (通过api路由)

☒ 对代码进行注释编写

+ 添加子任务

📎 添加附件

### 评论

○ [梁晓倩](#) 创建了任务 12月17日 10:32

输入评论

Aa 😊 @ 📷 🖋️ | ➡️

👤 3 人关注 +

✓ 任务已完成



## Mysql数据库设计

创建于会话: 互联网数据库开发小组

张昊星

今天 明天 其他时间

互联网数据库开发 默认分组

+ 添加至任务清单

与前端负责人和后端负责人沟通，设计合理的数据库并且构建索引，建立一个能够完成网页需求的数据库，可选择构建索引加速查询

5 / 5

✓ 数据库设计



✓ 完成表与表之间的联系

✓ 对每个表的主键进行规划

✓ 对设计好的数据库进行范式评估

✓ 与前后端沟通，讨论数据库设计是否合理

+ 添加子任务

添加附件

### 评论

梁晓婧 创建了任务 12月17日 10:40

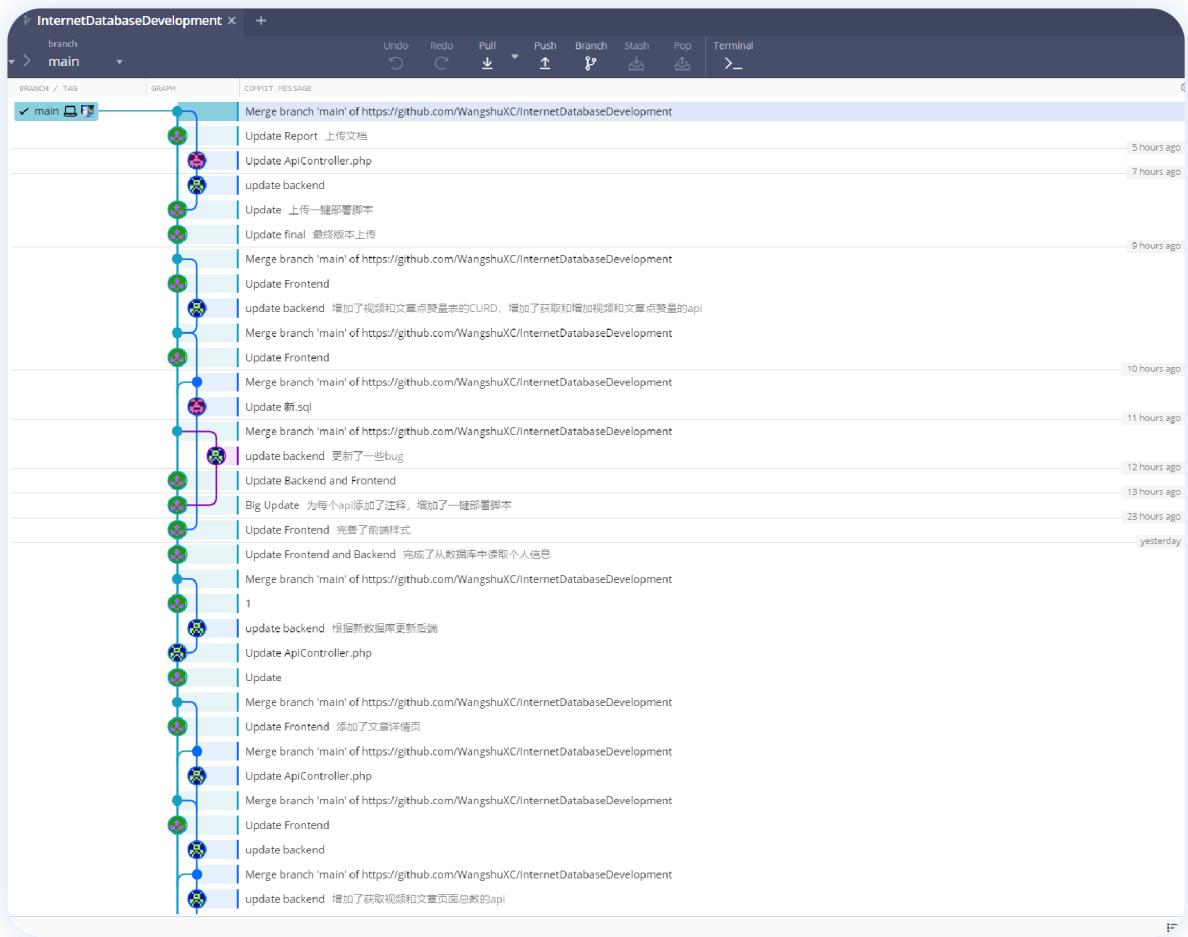
输入评论

Aa 😊 @ 📎 ✎ | ➡

3 人关注 +



Git提交记录



November 23, 2023 – December 23, 2023

Period: 1 month

#### Overview

0 Active pull requests

0 Active issues

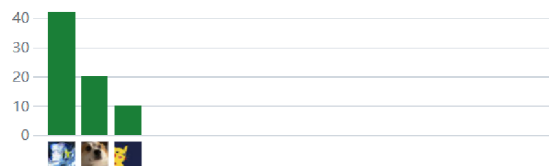
0 Merged pull requests

0 Open pull requests

0 Closed issues

0 New issues

Excluding merges, **3 authors** have pushed **72 commits** to main and **72 commits** to all branches. On main, **0 files** have changed and there have been **0 additions** and **0 deletions**.

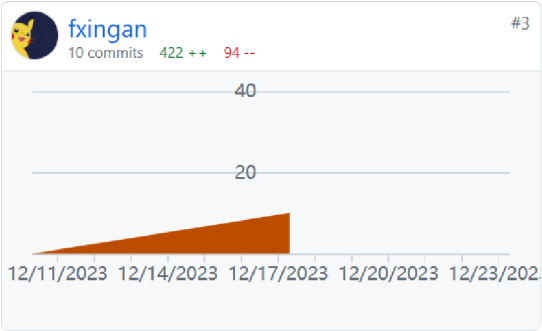
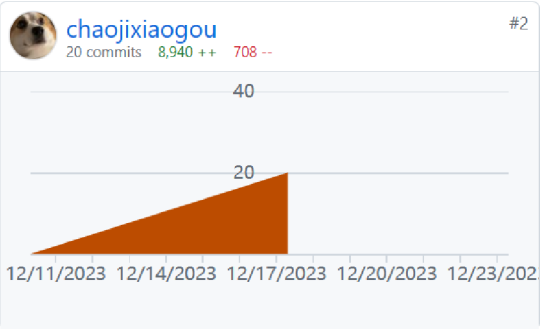
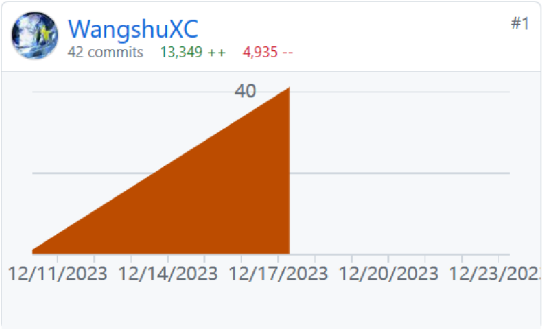
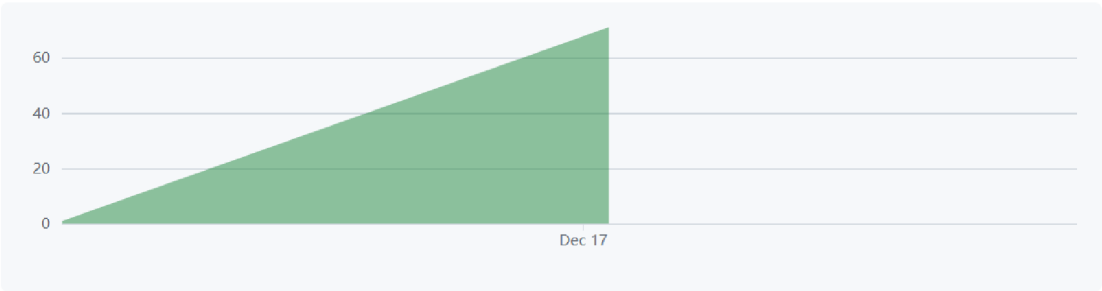


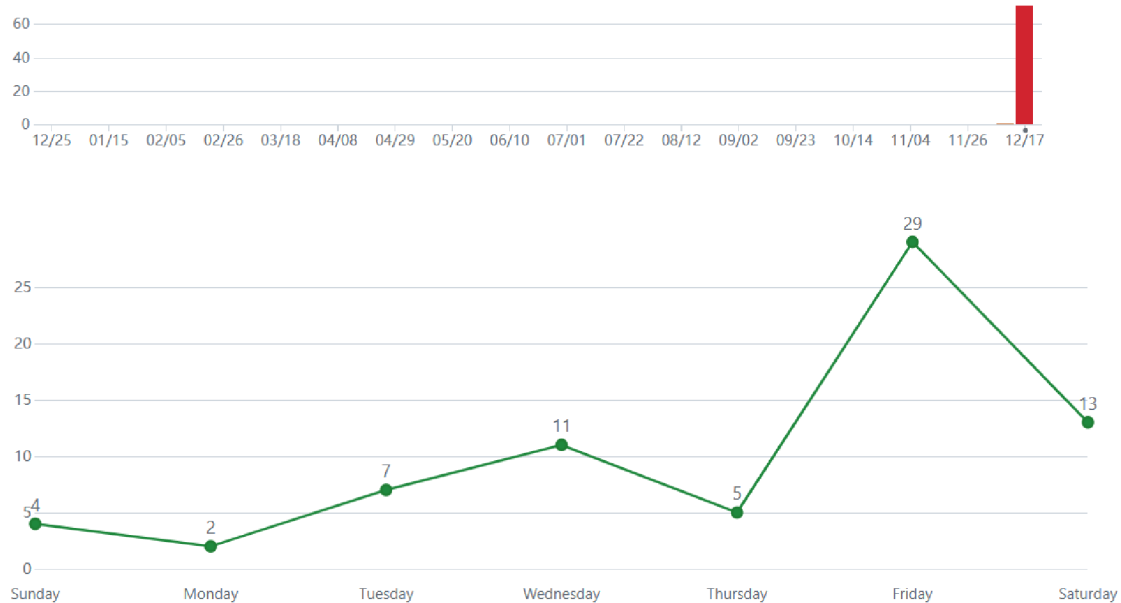


Dec 10, 2023 – Dec 23, 2023

Contributions: Commits ▾

Contributions to main, excluding merge commits





### Code frequency over the history of WangshuXC/InternetDatabaseDevelopment

