Java2 Final Project Report

Team Members and Contributions

Student Name	Student ID	Department	Contribution	Work
Huihui HUANG	12010336	CSE	50%	Frontend, Frontend/Backend Connection
Yitong WANG	11910104	CSE	50%	Data Collection, Backend

Repositories Selected

We selected 4 repositories in this project, and they all meets the requirement for this project:

- https://github.com/facebookresearch/pytorch3d.git
- https://github.com/spring-projects/spring-boot.git
- https://github.com/itzg/docker-minecraft-server.git
- https://github.com/redis/redis-py.git

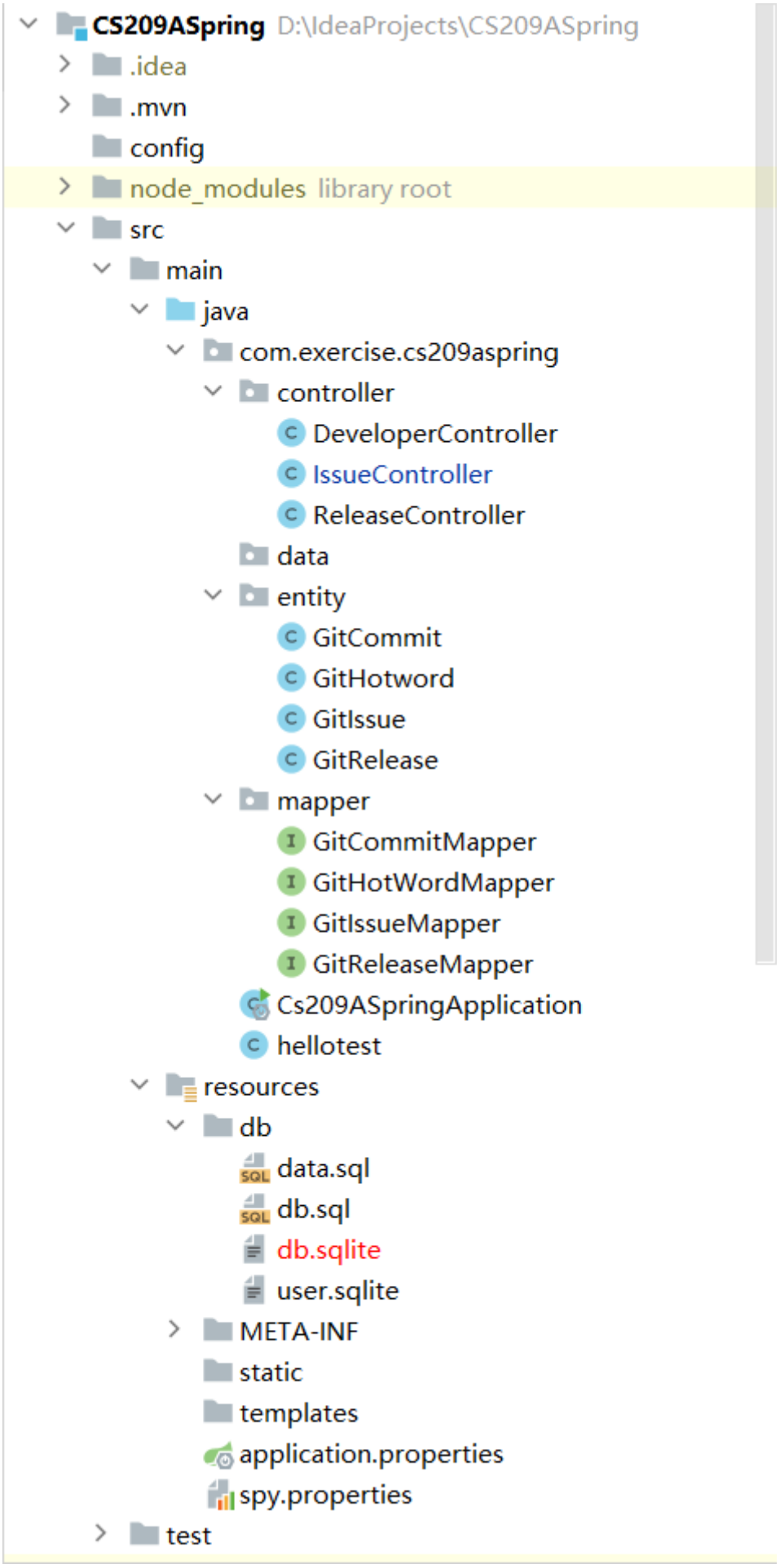
Architecture

Back-End Architecture

Overall Structure: Springboot + Mybatis(Database Interaction) + JPA(Data Persistence) + SQLite(Database)

Springboot

The directory structure of the backend:


CS209ASpring D:\IdeaProjects\CS209ASpring
.idea
.mvn
config
node_modules library root
src
main
java
com.exercise.cs209aspring
controller
DeveloperController
IssueController
ReleaseController
data
entity
GitCommit
GitHotword
GitIssue
GitRelease
mapper
GitCommitMapper
GitHotWordMapper
GitIssueMapper
GitReleaseMapper
Cs209ASpringApplication
hellotest
resources
db
data.sql
db.sql
db.sqlite
user.sqlite
META-INF
static
templates
application.properties
spy.properties
test

Package	Description
controller	Handle request from frontend
entity	Entities to maintain the objects in database
mapper	Interfaces to interact with database

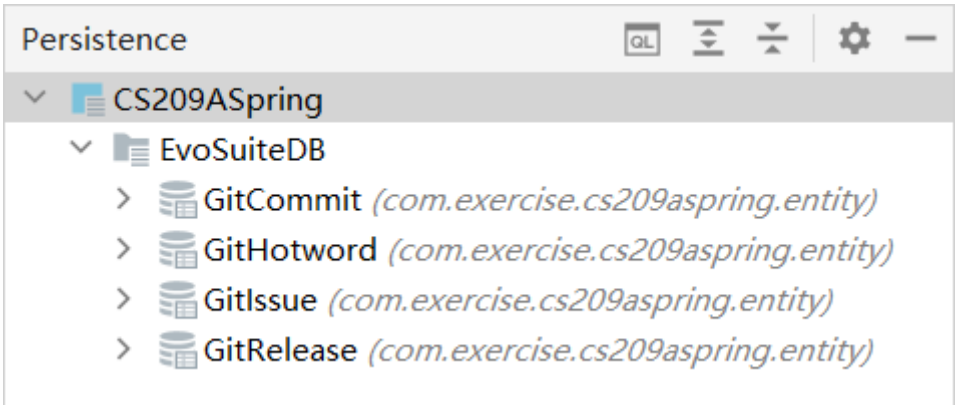
Mybatis

To simplify the configuration, we just import mybatis configuration from pom.xml.

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.5.1</version>
</dependency>
```

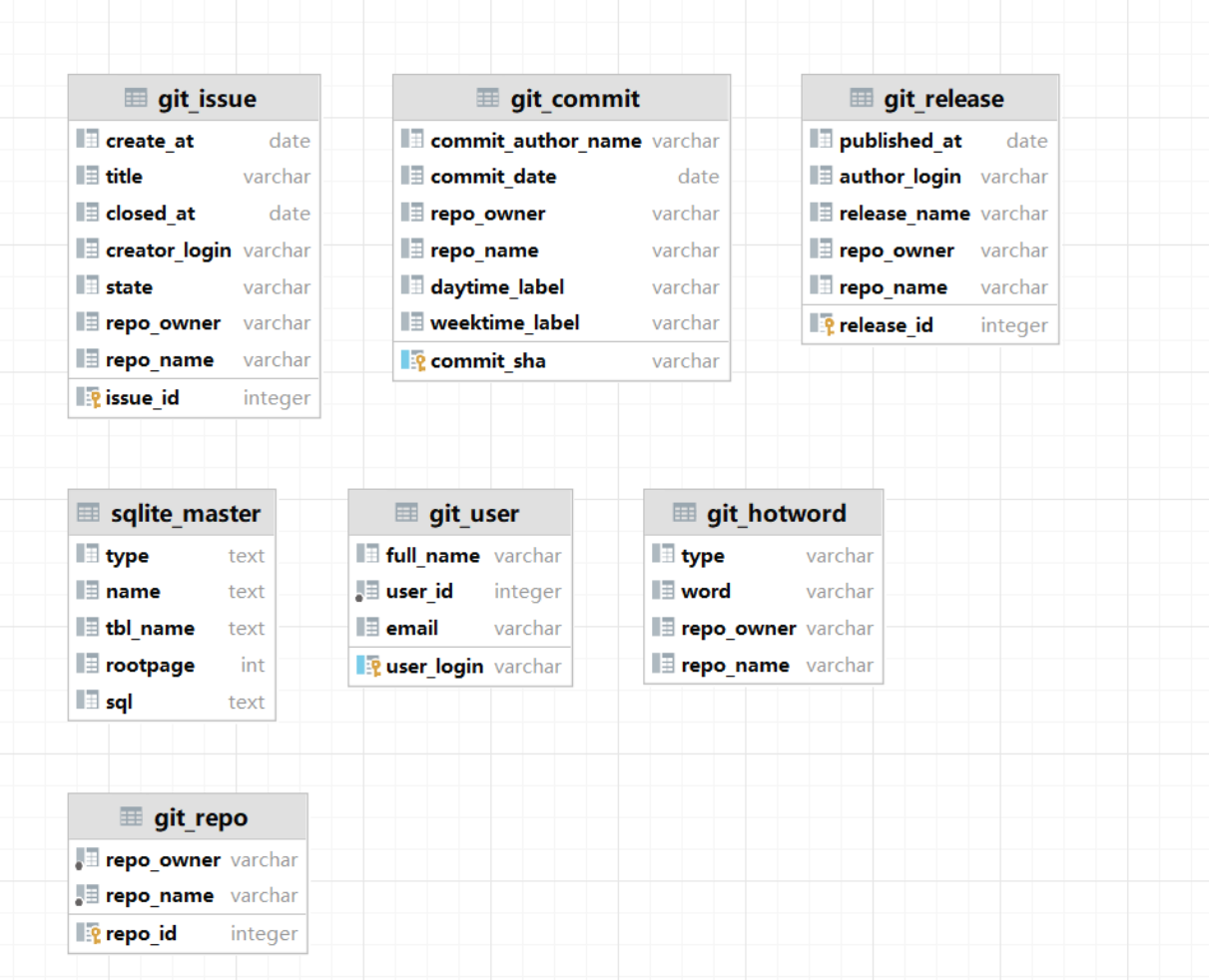
JPA

We use the JPA functionality provided in IDEA. Since we only need to provide the interfaces to answer the questions in the document, we only select 4 tables to generate the persistence entity.



SQLite

There are 7 tables in the SQLite database. We do not design database foreign keys since we have ensured that the data can be queried and stored in a correct form.



APIs

For the communication between the frontend and backend, we developed several restful APIs:

Restful API	Corresponding Question
/{owner}/{repo}/developerCount	How many developers have committed to this repo?
/{owner}/{repo}/mostActive	Which developers are the most active (i.e., who committed the most)?
/{owner}/{repo}/mostActiveTop?top=	
/{owner}/{repo}/issueCount?state=	How many issues are open and how many are closed?
/{owner}/{repo}/issueCountTop?top=	
/{owner}/{repo}/varIssueResolutionTime	What is the typical issue resolution time (i.e., the duration between issue open
/{owner}/{repo}/rangeIssueResolutionTi	

me	time and issue close time) for this repo?
/{owner}/{repo}/avgIssueResolutionTime	
/{owner}/{repo}/releaseCount	How many releases are there in this repo?
/{owner}/{repo}/releaseTimeStampTop?top=	How many commits are made between each release?
/{owner}/{repo}/commitBetween?lower=&uppper=	
/{owner}/{repo}/dayTimeLabel?label=	At which time (e.g., weekday, weekend, morning, evening, etc.) do developers made commits?
/{owner}/{repo}/weekTimeLabel?label=	
/{owner}/{repo}/gitHotWordTop?top=	**(Bonus)** Which topics/keywords/problems are often discussed in this repo?

Data Collection and Analysis

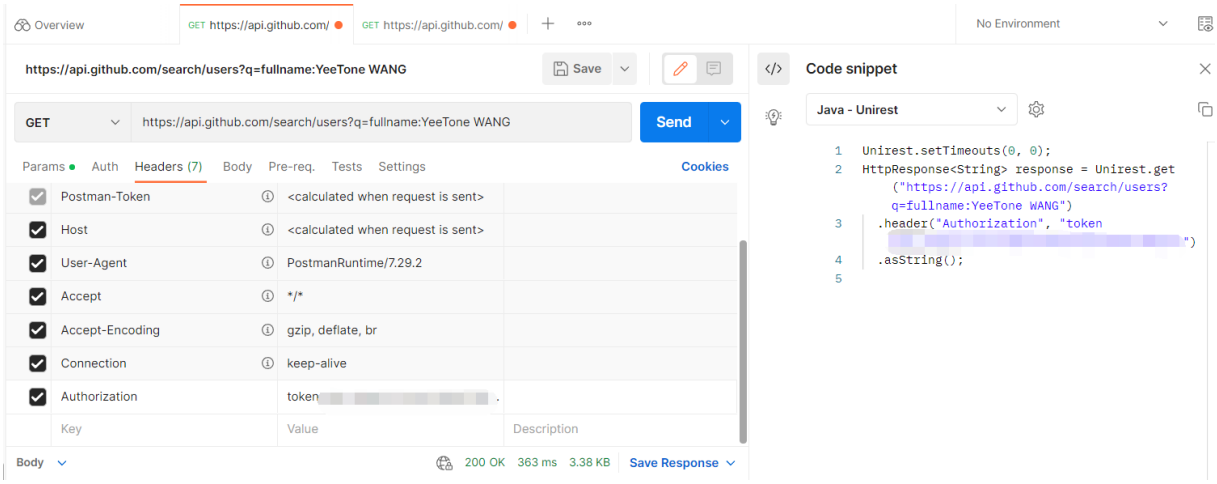
Technique Overview

- Data Query:
  - Postman
  - Unirest
  - OkHttp
- Data Parsing and Analysis:
  - FastJSON
  - NLP stopwords
- Database Importing:
  - SQLite JDBC

Query Github Restful APIs

Data collection is an extremely important part in this project. Github has numbers of public

restful apis to collect data of github repos, and we use unirest and OkHttp to query data and fastjson to parse them. But the code for querying data is automatically generated by postman tool:

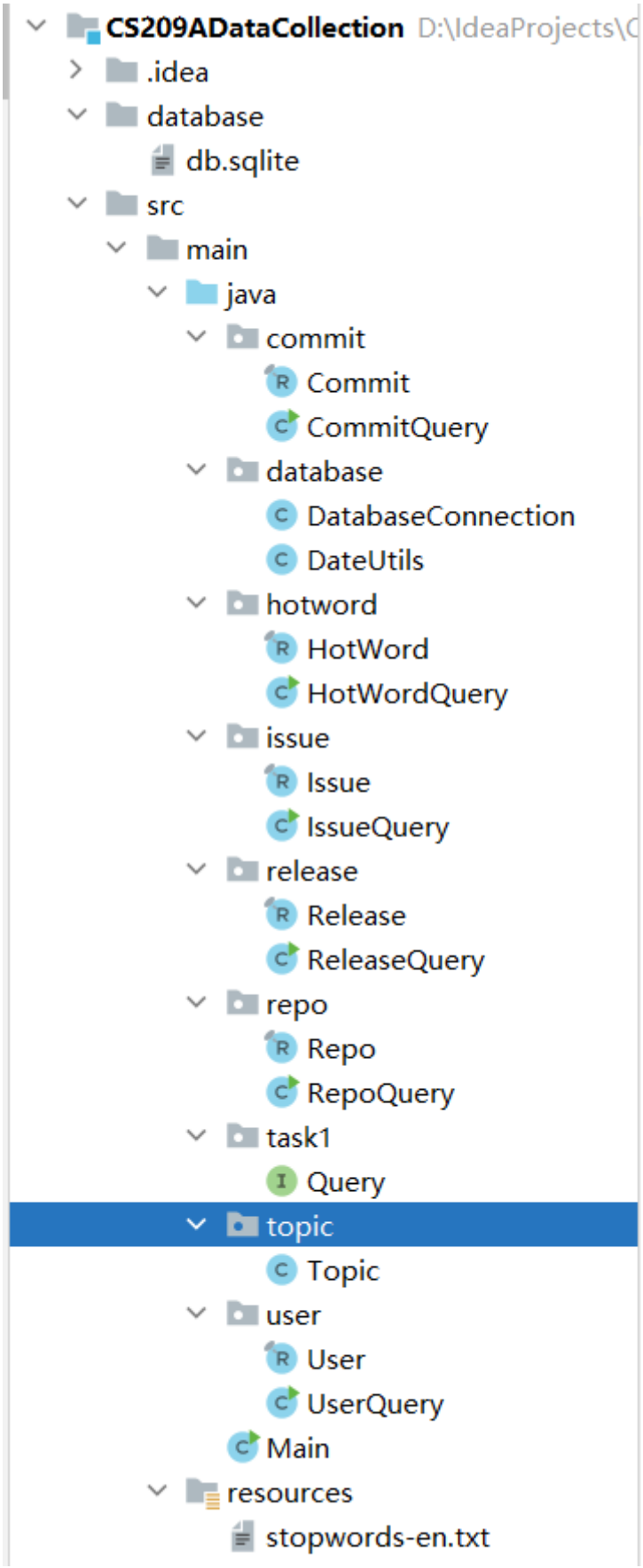


We mainly collect data from these restful apis of github:

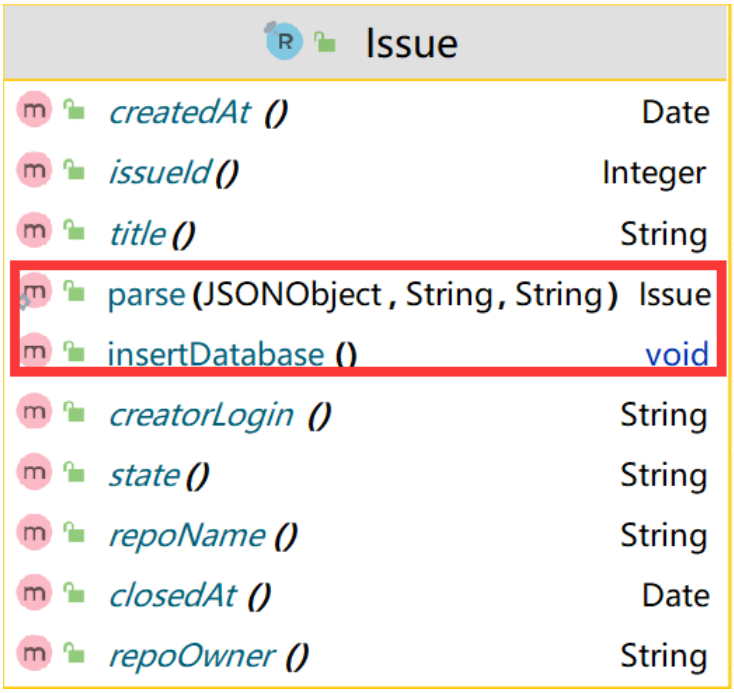
- https://api.github.com/repos/{owner}/{repo}/commits
- https://api.github.com/repos/{owner}/{repo}/issues
- https://api.github.com/repos/{owner}/{repo}/releases
- https://api.github.com/repos/{owner}/{repo}
- https://api.github.com/users/{user}

Interact with Database

For each table of database, we provides a record class and a class for query:



Each of the record class provides a method to parse JSONObject to the record type and a method to interact with the database:



As introduced in CS307 Database Principle course, we have learnt that the open/close JDBC database connection is quite expensive. So we

(1) maintain the database connection via the singleton pattern(单例模式).

```
private static Connection connection;

9 usages
public static Connection getInstance() {
    if(connection == null){
        try{
            Class.forName( className: "org.sqlite.JDBC");
            connection = DriverManager.getConnection(Query.JDBC);
            connection.prepareStatement( sql: "PRAGMA foreign_keys = ON").execute();
        }catch (Exception e){
            e.printStackTrace();
        }
    }

    return connection;
}
```

(2) use batches to insert multiple rows in a single queries:

```
Connection connection = DatabaseConnection.getInstance();
connection.setAutoCommit(false);
```

```
for (int i = 0; i < commitList.size(); i++) {
    {
        ps.setString( parameterIndex: 1, commitList.get(i).commitSHA());
        /*...*/
    }
    ps.addBatch();
    if (i % 20000 == 0) {
        ps.executeBatch();
        connection.commit();
    }
}
ps.executeBatch();
connection.commit();
connection.setAutoCommit(true);
```

Data Manipulation and Analysis

Considering the restful apis of github have provided most answers in the documentation, we select some important data manipulation in this part.

Time Labels for Commits

The original date data from github is given as a timestamp, so we use java.util.Date to label its weektime and daytime.

```
private static String weekTimeLabel(Date date){
    long time = date.getTime();
    java.util.Date d = new java.util.Date(time);
    int day = d.getDay();
    return switch (day) {...};
}
```

NLP(自然语言处理)

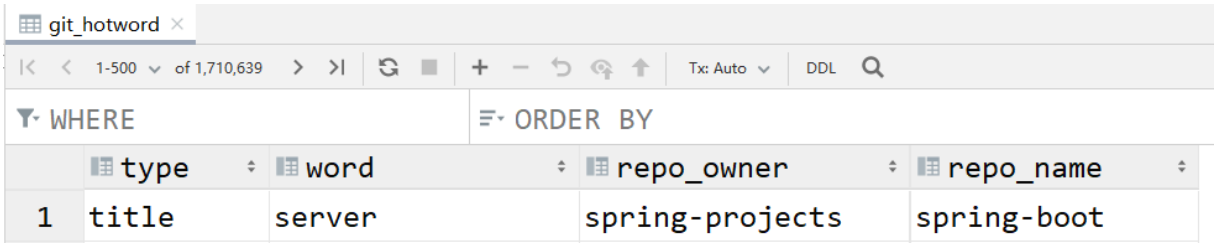
To answer the bonus question, we needs NLP to analyze the texual contents in the issue. We apply the stopwords(停词) from github to remove these words of none sense. Here are the links of stopwords repositories we used in github:

- https://github.com/stopwords-iso/stopwords-en
- https://github.com/6/stopwords-json
- https://github.com/stopwords-iso/stopwords-iso

After the textual words have been queried from github issues api, we could split it and remove those textual words in stopwords or with **punctuations. The following pesudocode shows the progress of analyzing hotwords:**

```
func analyze_hotwords(issue_text: string) return List:
    if STOPWORD is not prepared:
        prepare_stopword();
    if issue_text is NULL:
        return EMPTY_LIST;
    s <- nlp(issue_text)
    result <- EMPTY_LIST;
    for each word in s:
        if word is all English letters:
            if word not in STOPWORD:
                result.add(word)
    return result
```

For each word, we will store one row in the database. There are more than one million data rows in the table.



Although it is a little costy in memory, the query for backend is quite convenient. To select the words often discussed, we could just select them by the frequency in database.

Front-End Visualization

General View

The whole data is shown in this view clearly.

Charts

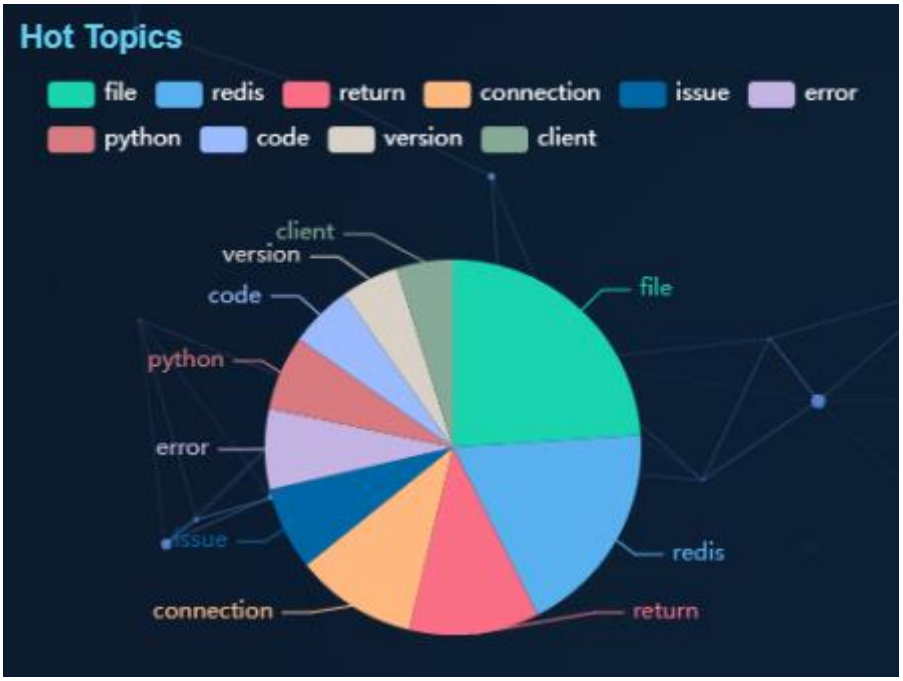
We uses many kinds of charts and each of them has different functions.

1. Wordcloud chart (made with Vue-Worldcloud)

This wordcloud chart shows the hot issue topics and the topic which mentioned more will be bigger in the view.

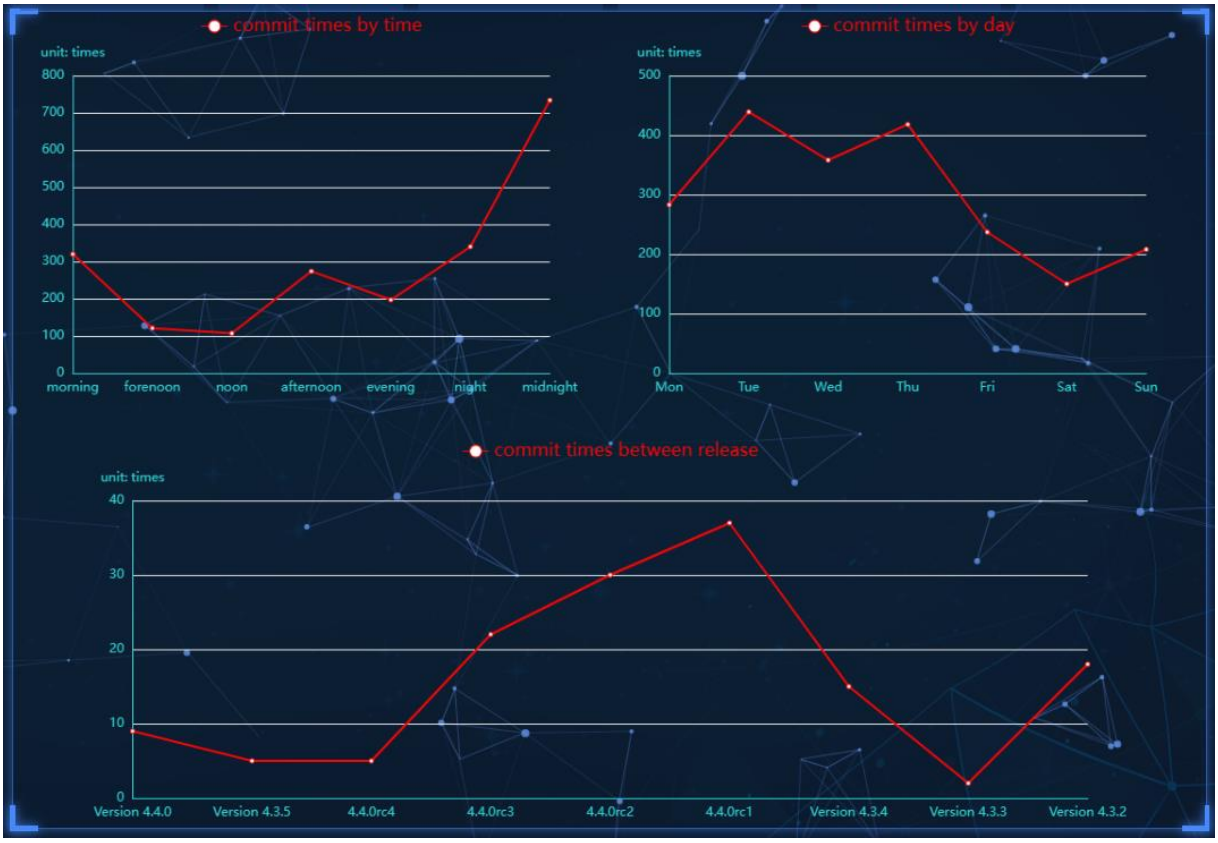
2. Pie chart (made with ECharts)

This pie chart shows the detail proportion of the top ten hot topics.



3. Line chart (made with ECharts)

There are three line sharts. The top-left chart shows the commit times by time, like morning, forenoon and so on. The top-right chart shows the commit times by each day in a week. The bottom chart shows the commit times between two releases.



4. Form chart (made with ECharts)

This form chart shows the typical issue resolution time including average time, range time and variance time.

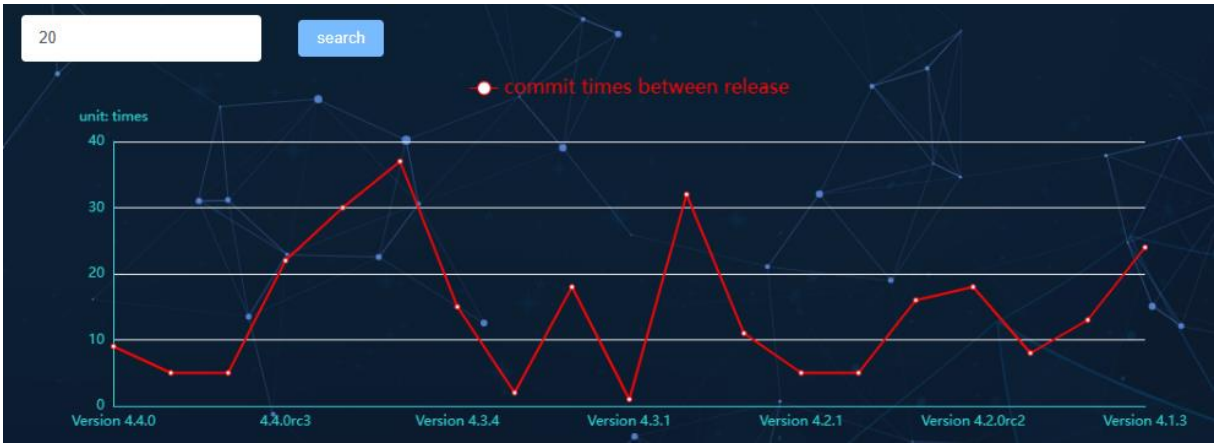
5. Ranking (made with ECharts)

This ranking form shows the commit authors' s name and their commit times which also sorted by the commit times.

User Interact

1. Search

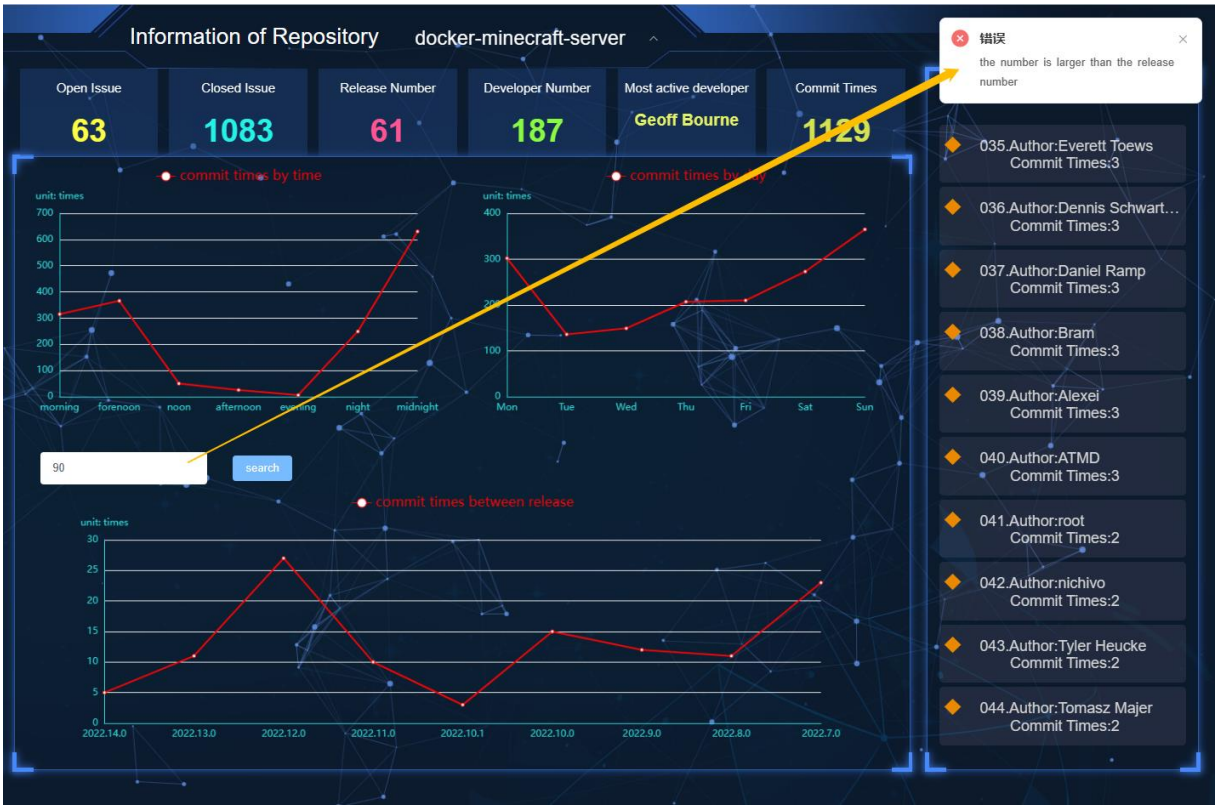
When user view hom many commit times between twe releases, he can choose how many releases he want to see.



2. Check

range check

Since the releases number is limited, when searching, if the search release number is larger then the true release number, it will give a warning.

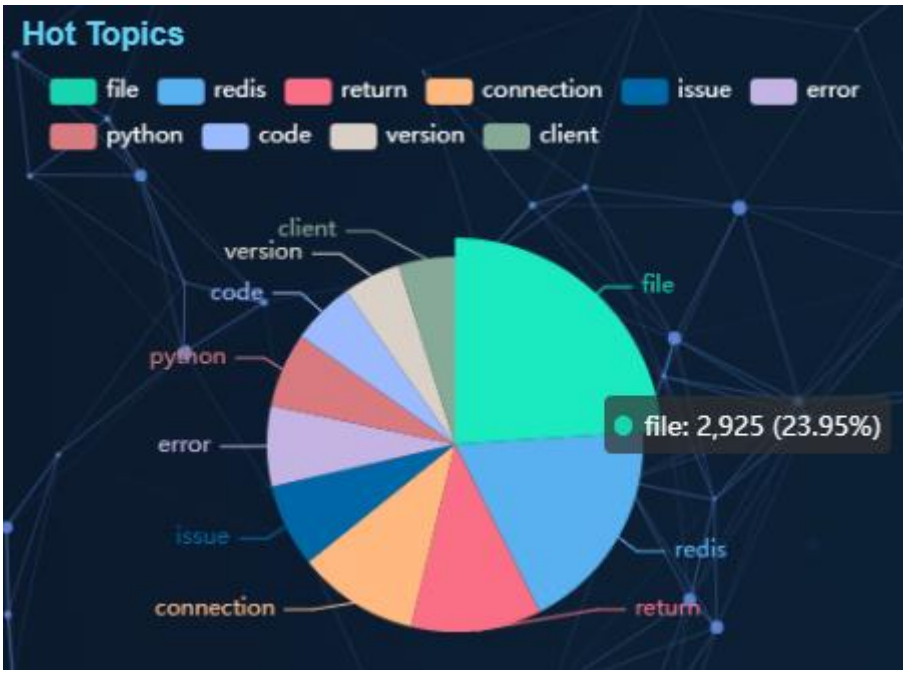


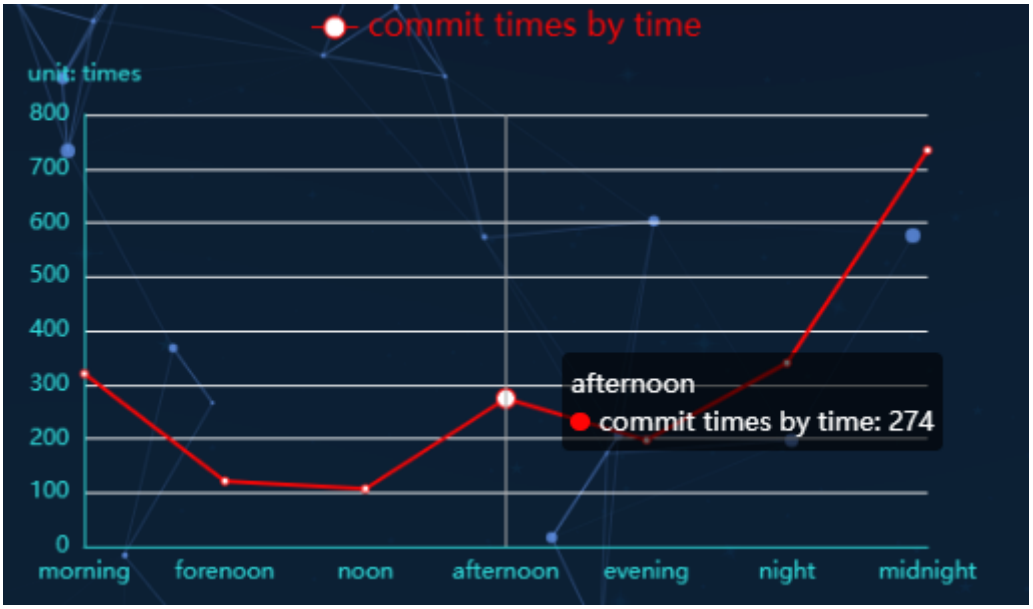
number check

Besides, if the user put some invalid number like alphabet, it will also give a warning.

3. Highlight

The echarts can auto focus on one series of data and highlight them, just like this.





4. Change repository

There are four repositories, and if the user want to change the repository, he can change it commodiously, just like the following shows.

Techniques

Frontend Techniques

The Frontend is based on GitDataV framwork using Vue-cli and Vue 2 to build and also use the following 3-th party library:

- ECharts
- Element-Ui
- Vue-Worldcloud
- Animate.css
- Axios

Besides, the animation are with the help of setInterval function and the watch function whitch can fetch the updating data in real-time.

Frontend and Backend Communication

The frontend uses **Axios** to solve the request and almost all the api we used is get request.

The backend uses **Sping Boot** to solve these request.

Results and Insights

We analyzed different repo and topic, and used their data for corresponding analysis.

We will show our findings by answering the following questions:

(1) What are the answers to the above questions?

Most of answers can be found in the query result of the restful apis provided by Github. Although some parts need data manipulation and analysis, it is not difficult to get the answers and show them in a proper way.

(2) What can we learn about the repo according to your answers?

a. Different repos varies quite a lot, and their interest focuses on quite different areas.
b. For the same repo, the activity of committers are also not identity. Some committers are very active. From another aspects, committers are more likely to submit their commits in weekdays and in midnights.
c. Issues resolution time varies a lot. Some issues could be resolved in 1 hour while some needs more than 3 years.

(3) What can be improved about this repo?

For some adjacent releases of the repos, there are few commit times between them. This may indicate that there are not too many updates for this release. It is suggested distributing commits more evenly.