# Code Repository

All source code and War file are included in Github Repository.

Github: <https://github.com/EricXiaxl/TodoApplication.git>

# Main Functions Introduction

1. User Service:

* Register
* Login
* Logout
* Authentication and Authorization

1. Todo-List Service

* Query todo list (In order)
* Add todo
* Delete todo
* Update todo

# Tech Chosen

1. **Back-end**:

* **java 8**
* **Spring/SpringBoot**
* **RESTful API**
* **H2 Database:** the biggest advantage of H2 is that H2 provides a very convenient web console (visit localhost:port/h2-console) for operating and managing database content, which is much easier to use than the swing and awt consoles of HSQLDB.
* **Spring Security:** Compared with the traditional security framework Shiro, Spring Security is easier to integrate with other members of the Spring family. Furthermore, in terms of security, in addition to providing the most basic authentication and authorization, Spring Security also provides other security management functions such as session fixation attack defense, CSRF defense and Http firewall, etc.
* **JWT:** In the case of distributed application deployment in the future, JWT has better scalability than Session, because Session needs to do multi-service data sharing, while JWT does not. In addition, one of the principles of RESTful API is stateless, jwt does not store any state on the server side.
* **Cache (Redis):** Using Redis to store user information can reduce the number of times the server queries the database.

1. **Front-end**:

* thymeleaf
* JavaScript
* JQuery
* CSS

# Architecture

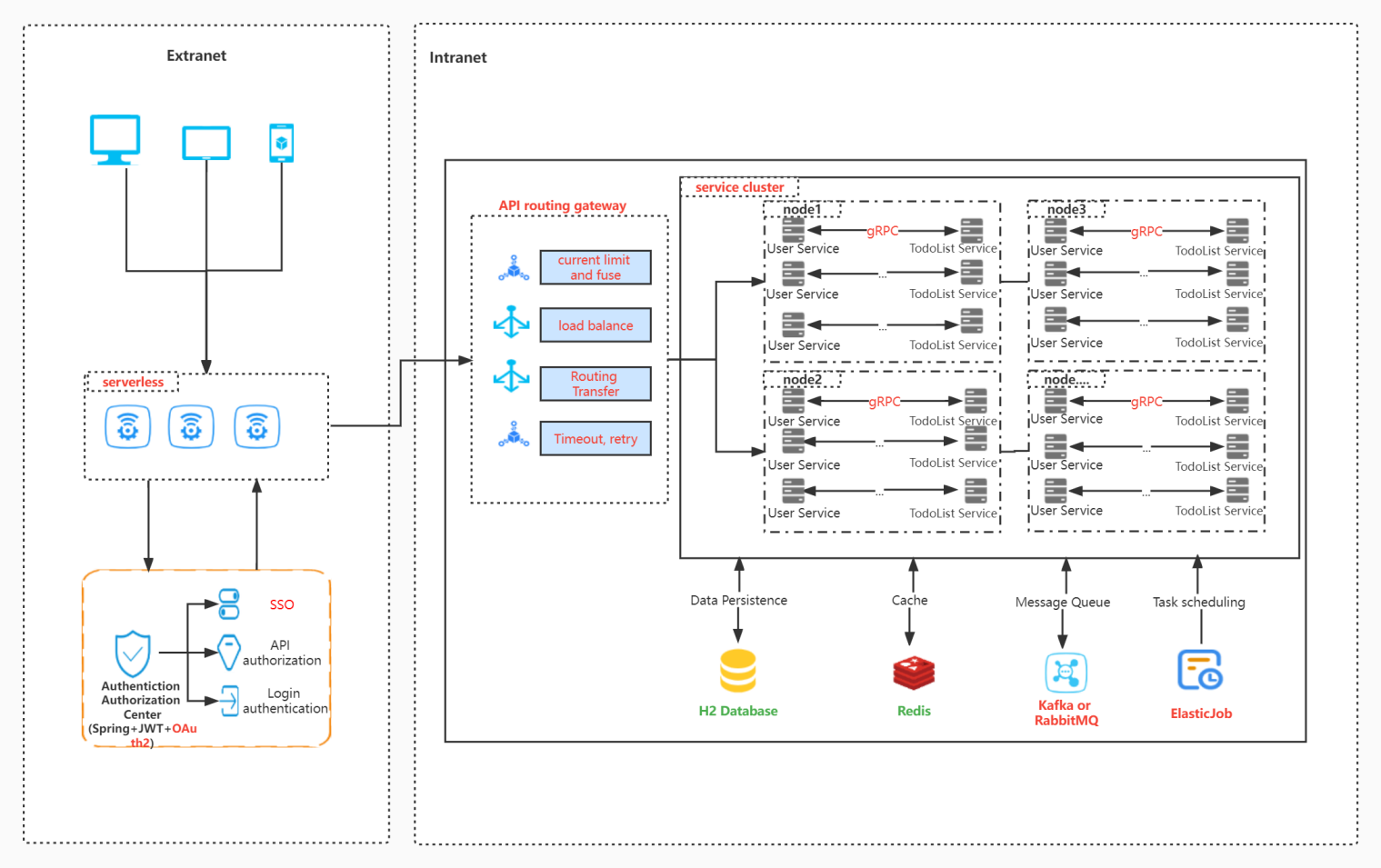


Fig 1. Todo Application Architecture

(Due to time constraints, red sections are future work if given more time)

# Code Structure

1. Project Structure:

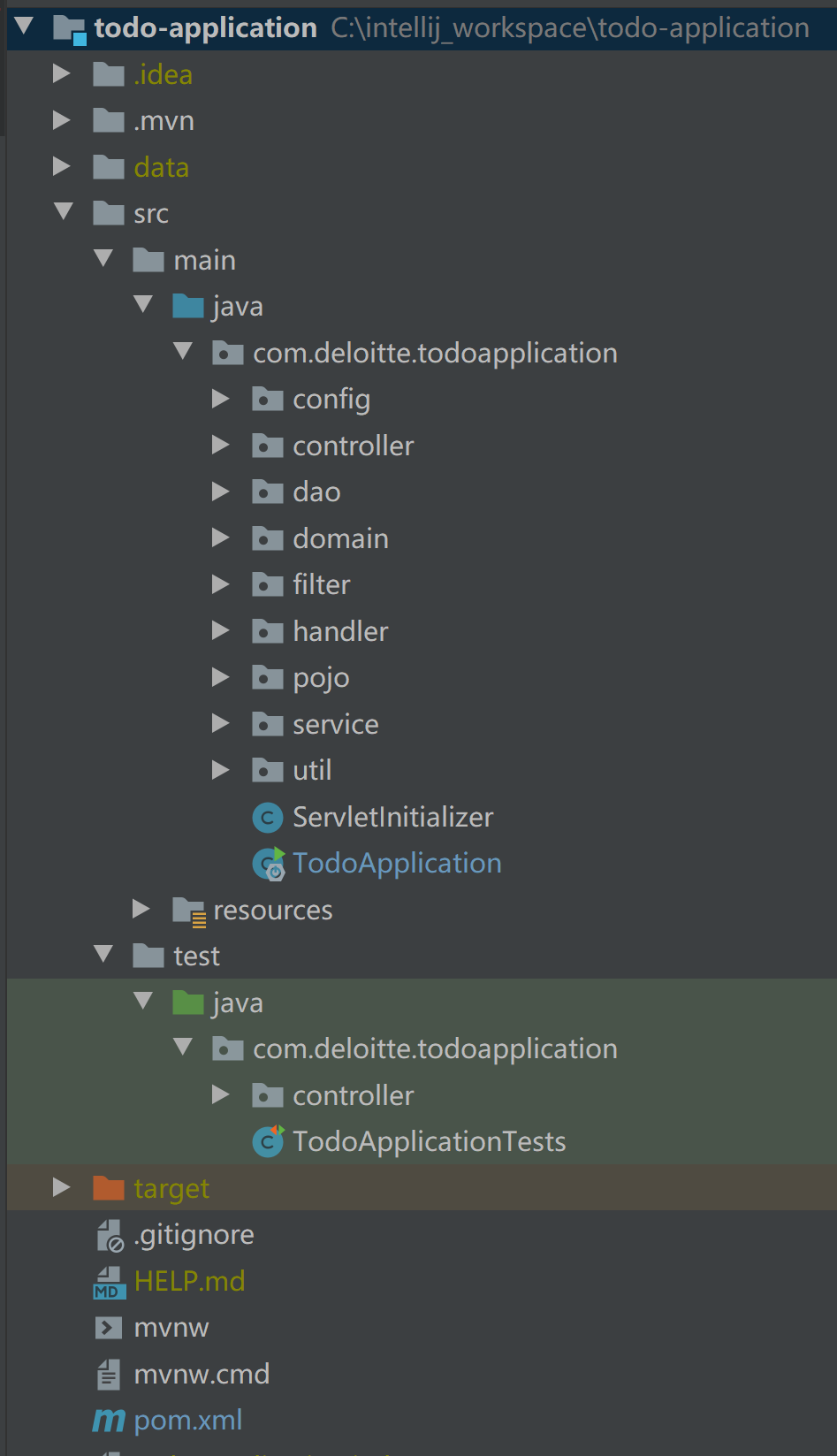


Fig 2. Project Structure

1. Data Structure:

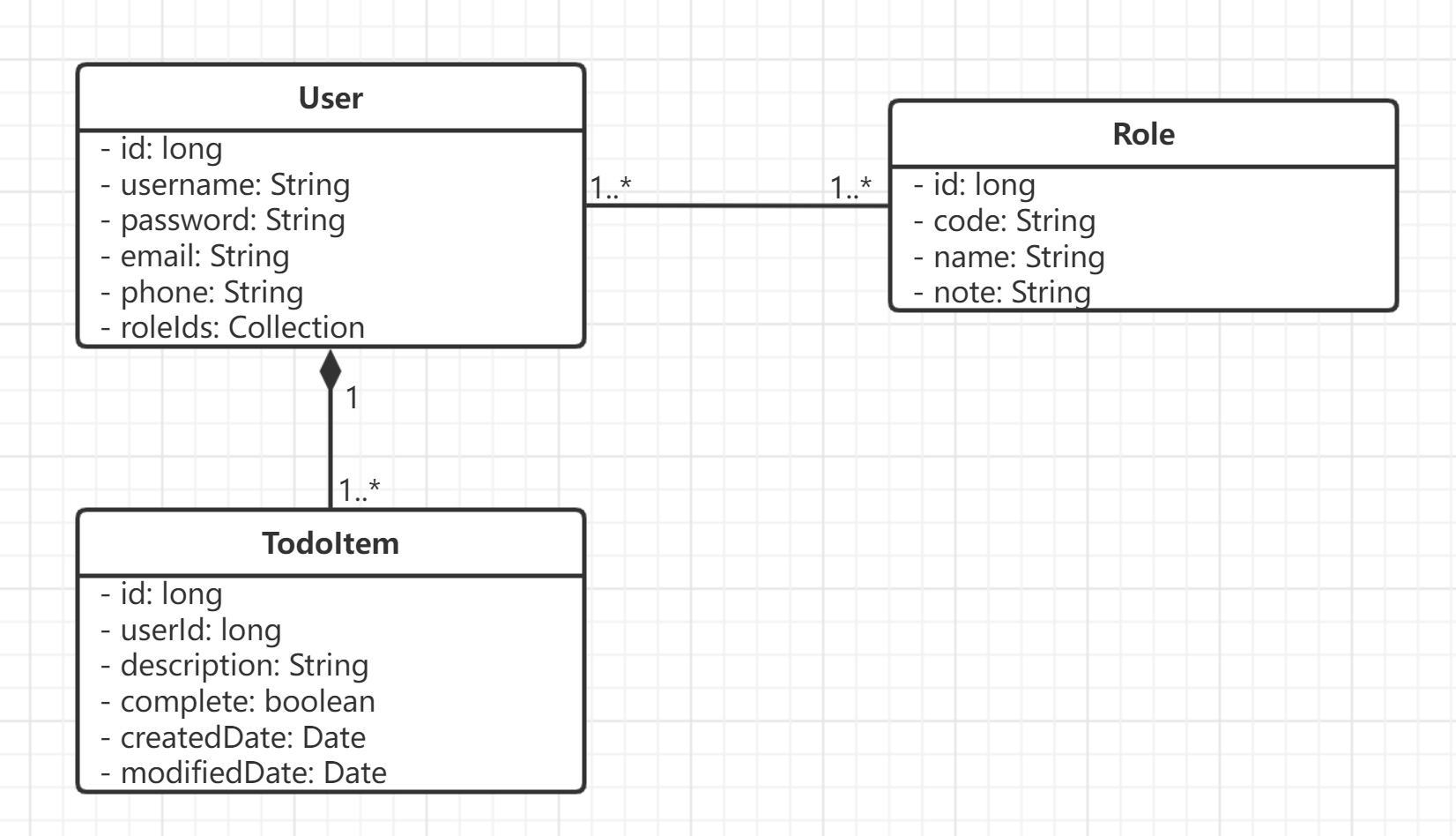


Fig 3. UML - Class Diagram

# Metrics

1. **Security**

* Use Spring Security and JWT to authenticate user identity and authorize API access, and enable CSRF defense at the same time.
* The user password is encrypted and stored in the database.
* Limit the length of relevant fields, such as the description of todo, to avoid the risk of too long request message in malicious requests.
* Data transaction implementation and data rollback.

1. **Performance**

* Storing user information in Redis when user successfully logs in, which means that there is no need to query the database through User Id to obtain user information, thereby reducing database operations.
* Implemented multi-threaded operations on todo to improve processing efficiency.

# Testing

* Testing REST APIs with Postman.
* Using JUnit MockMVC to test Controller and Service to verify that the function is normal and the error message returned is correct.

# Future Work

Due to time constraints, and in order to make the project more convenient to run in the local environment of the code reviewer, the project has been streamlined a lot. So if I will be given more time, I will improve this application in the following ways:

1. **Security**:

* Use HTTPS request to access.
* Login via SMS or email verification code to improve security.
* Perform more text security and legality checks.
* Added forgotten password function.
* Integrate OAuth2 to realize SSO and third-party authorized login.
* Integrate Zipkin to monitor APIs, and use Sentinel to implement API fuse mechanism.
* Add database master and backup mode, so system can switch to the backup node when the master node has a problem.

1. **Performance:**

* Perform paging query on the todo list to improve query efficiency of database.
* Add new gateway/message queue (RabbitMq, Kafka) to achieve load balancing, and reduce the number of service visits and request service waiting time.
* The master-slave configuration is generally combined with read-write separation. The master server is responsible for writing data, and the slave server is responsible for reading data, and ensures that the data of the master server is synchronized to the slave server in time. The master-slave mode can better share the database pressure, separate the insert update operation from the query operation, and improve the overall performance of the system.

1. **Scalability:**

* Implement service cluster multi-node deployment