PROJECT SECURITY DOCUMENT

Group: M16B-4

Contents

1. Introduction

Our project team implemented a supportive system (Aecl Online Learning System) for undergraduate Computer Science students in the University of Sydney. The website is hosted on a RHEL (Red Hat Enterprise Linux) Virtual Machine including three independent servers, among which there is a web server to host static resources, an API server to handle all the user requests and a MySQL server to serve the database.

For data integrity concerns, we implemented full-site HTTPS encryption. All communications among users, web server and API server are encrypted by an SSL certificate issued by a verified Certification Authority (CA). Specifically, when the user requests a service, the secured conversation will be processed by web server, and be transmitted to API server to insert, obtain, update, or delete data from database.

Moreover, we design a clean RESTful API model to regulate the conversation between API server and web server. Services are divided in very detailed fashion to ensure the atomicity of APIs. Also, we built a logging system and detailed referencing document to ensure the availability and auditing of APIs.

2. System Specifications

Virtual Machine Environment: Red Hat Enterprise Linux 8.0

API Application: Spring Boot 2.2.7

API Application Packaging: Maven

Database Server: MySQL Community Server 8.0

Database Persistence Framework: MyBatis 3.5.4

Web Server: Gunicorn + Bottle.Py

HTTPS Certificate: DV Certificate Issued by TrustedAsia

Web URI Monitor Application: Alibaba Druid

API Document Application: Swagger UI

3. Security Architecture Overview

To simulate the real-world server configuration of some serious website, we built three different applications, listening to three different ports in a stable VM environment. All the applications run independently and through the functional interaction among them, the website can handle the service requests securely and properly.

Firstly, HTTPS works as the only channel in regard of data transmission. As shown in the lower portion of *figure 1*, when web server receives the user's request to some web page, it will find all the related static resources, integrate them and respond to the user, during which the URL of the webpage is going to change into URI format to incorporate with our SSL setting. As a result, web server will answer to prospective users with a public key, which is for users to encrypt a symmetric secret key for further communication.

According to *figure 1*, web server and database server are separated by an API server due to information integrity, and this middleware processes requests from web server, where reduce the overload of the web server, and organizes requests and answers around the database, so that protects and maintains our database be operational.

Secondly, all the user requests go through front-end filtering. The web server will not only pre-process the user request to make it compatible with API server RESTful requirements but run some form validation as well.

Thirdly, when the API server receives requests from Web Server, it will first check the validity of the authorization token and the correctness of RESTful API invocation. Before forwarding the request to database server, it will process the request again to prevent the possible database injection attack and identity forgery.

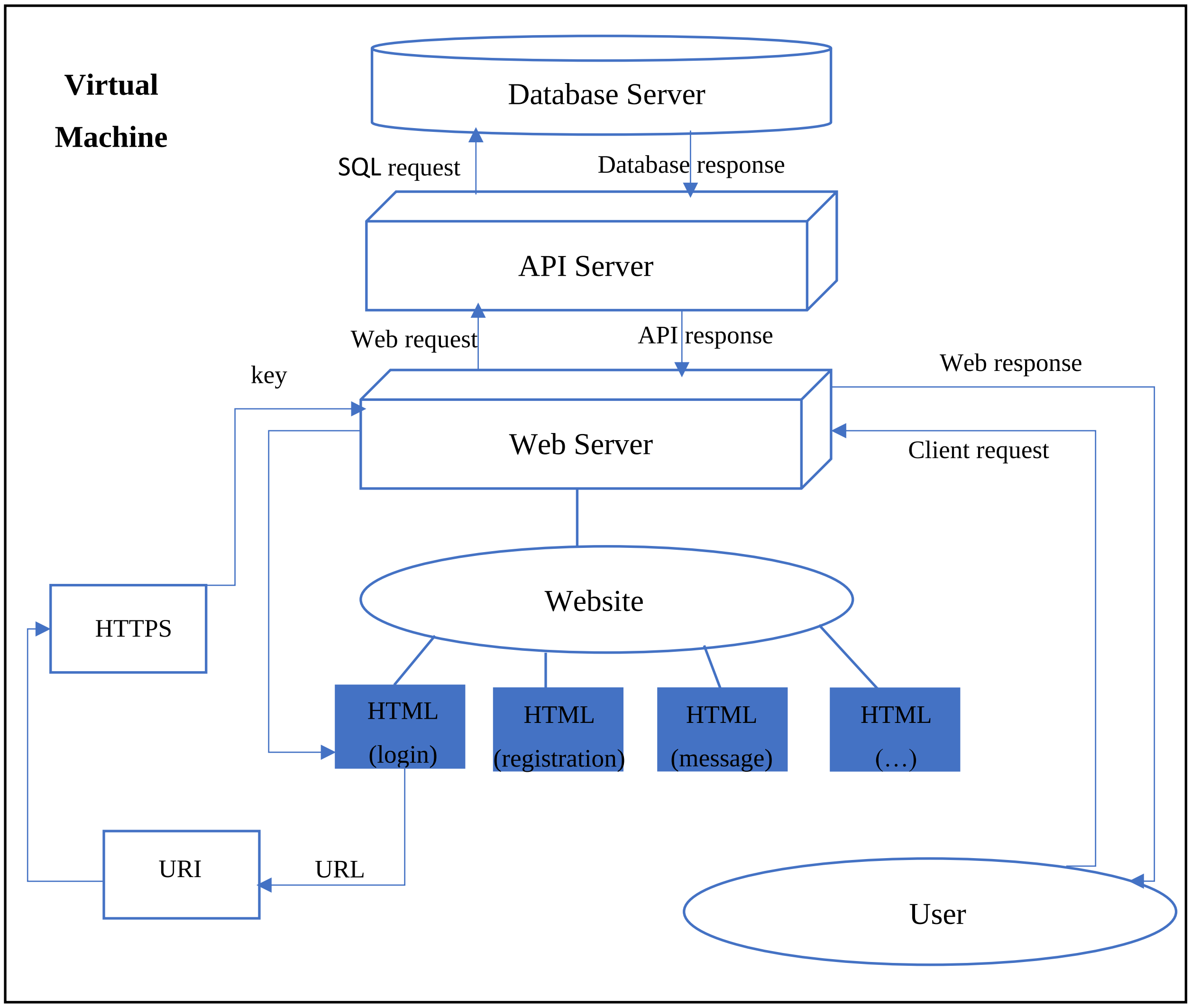
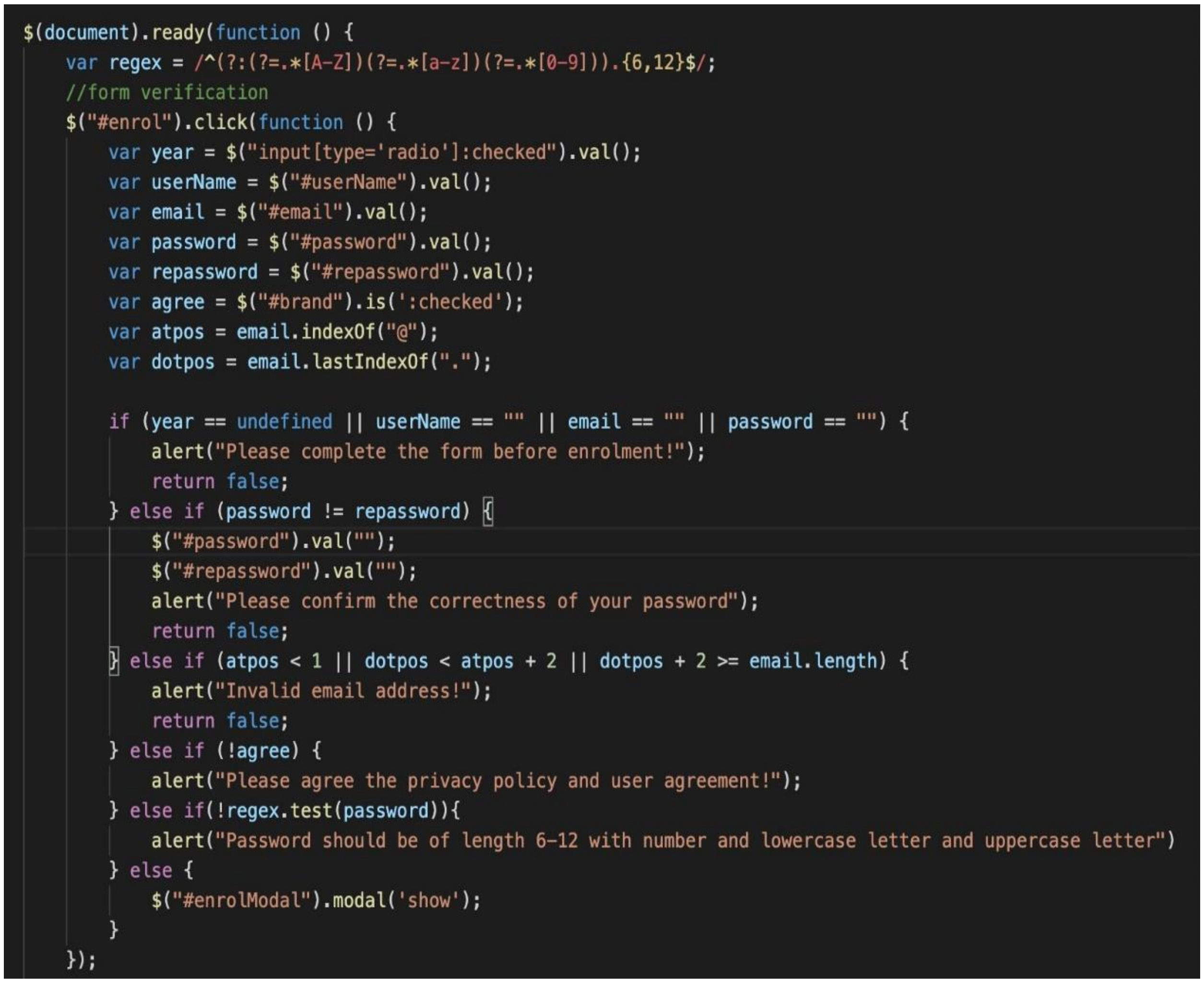


Figure1. Architecture of Aecl Security Solution

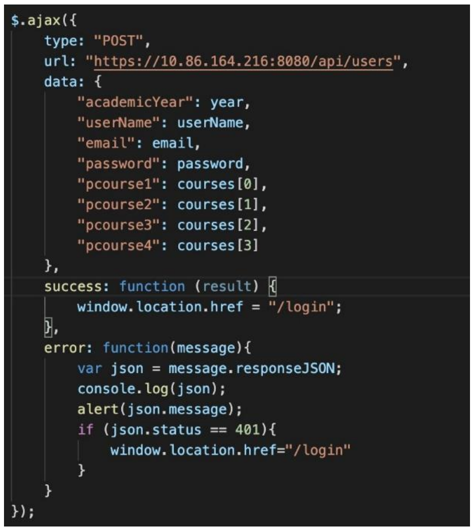
In details, a special case is put forward here to further illustrate our security strategy. The first part is the authentication process about registration and login for a new user while the second part entails with malicious account auditing and respective administration solution.

Assume a user requests service from Aecl, such that the user make commands about registration and web server receive the command and HTTPS tunnel provide a digital certificate to the web server, then it returns certificate with public key to “tell” user that they can have interaction secretly which the web server can serve each user privately. As the user registering, the personal information will be sent to web server, where Web server uses JavaScript to collect all required information such as year, username, email, password, then confirms the completeness of the data, and then packs the entire information as *figure 2* and *figure 3* to request API server to process the second check.



```
$(document).ready(function () {
    var regex = /^(?:(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])).{6,12}$/;
    //form verification
    $("#enrol").click(function () {
        var year = $("input[type='radio']:checked").val();
        var userName = $("#userName").val();
        var email = $("#email").val();
        var password = $("#password").val();
        var repassword = $("#repassword").val();
        var agree = $("#brand").is(':checked');
        var atpos = email.indexOf("@");
        var dotpos = email.lastIndexOf(".");

        if (year == undefined || userName == "" || email == "" || password == "") {
            alert("Please complete the form before enrolment!");
            return false;
        } else if (password != repassword) {
            $("#password").val("");
            $("#repassword").val("");
            alert("Please confirm the correctness of your password");
            return false;
        } else if (atpos < 1 || dotpos < atpos + 2 || dotpos + 2 >= email.length) {
            alert("Invalid email address!");
            return false;
        } else if (!agree) {
            alert("Please agree the privacy policy and user agreement!");
        } else if(!regex.test(password)){
            alert("Password should be of length 6-12 with number and lowercase letter and uppercase letter")
        } else {
            $("#enrolModal").modal('show');
        }
    });
```

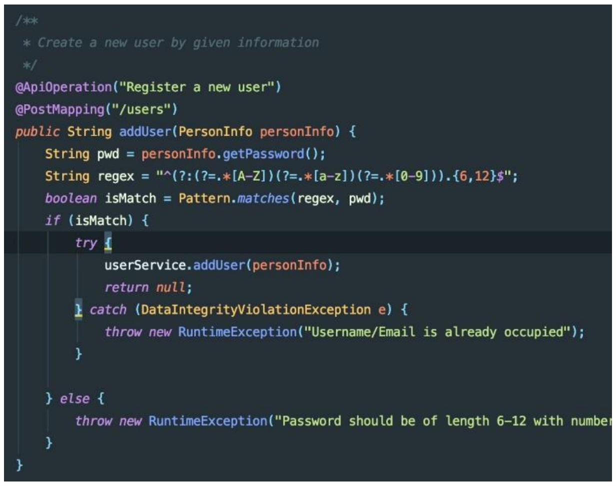
Figure 2. Data Collection by JavaScript



```
$.ajax({
    type: "POST",
    url: "https://10.86.164.216:8080/api/users",
    data: {
        "academicYear": year,
        "userName": userName,
        "email": email,
        "password": password,
        "pcourse1": courses[0],
        "pcourse2": courses[1],
        "pcourse3": courses[2],
        "pcourse4": courses[3]
    },
    success: function (result) {
        window.location.href = "/login";
    },
    error: function(message){
        var json = message.responseJSON;
        console.log(json);
        alert(json.message);
        if (json.status == 401){
            window.location.href="/login"
        }
    }
});
```

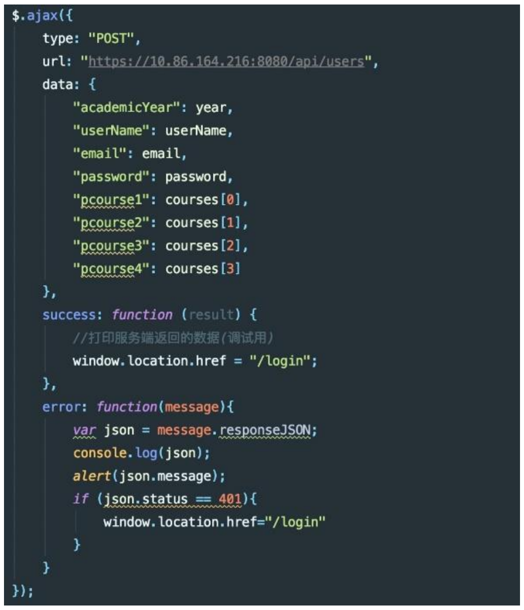
Figure 3. Ajax to API

Subsequently, API server ensures the registering user is not overlapped with others by comparing username and emails by database query and ensures that the password is following the rule by “regex”. This two-step verification is to filter the fake Ajax request made by malicious user. Then the API server will make a request to database server, ask it to add a new record for the new user as shown in figure 4. Whether the insertion is success or not, API will catch the exception message (shown in bottom of *figure 4*) from database and send back the information to web server to tell the user is the registration finished or not. In figure 5, it states the web server returns success response and will redirect to the login page. If the registration fails, 401 error message will be displayed as the response form web server to user.



```
/**
 * Create a new user by given information
 */
@ApiOperation("Register a new user")
@PostMapping("/users")
public String addUser(PersonInfo personInfo) {
    String pwd = personInfo.getPassword();
    String regex = "^(?:(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])).{6,12}$";
    boolean isMatch = Pattern.matches(regex, pwd);
    if (isMatch) {
        try {
            userService.addUser(personInfo);
            return null;
        } catch (DataIntegrityViolationException e) {
            throw new RuntimeException("Username/Email is already occupied");
        }

    } else {
        throw new RuntimeException("Password should be of length 6-12 with numbe
    }
}
```

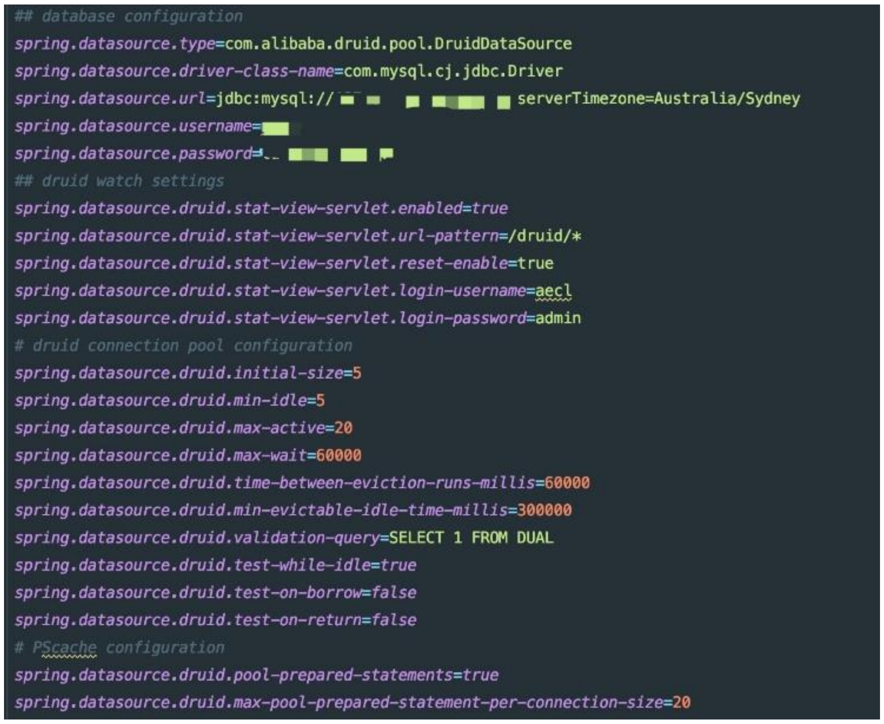
Figure 4. API Request to Database Server



```
$.ajax({
    type: "POST",
    url: "https://10.86.164.216:8080/api/users",
    data: {
        "academicYear": year,
        "userName": userName,
        "email": email,
        "password": password,
        "pcourse1": courses[0],
        "pcourse2": courses[1],
        "pcourse3": courses[2],
        "pcourse4": courses[3]
    },
    success: function (result) {
        //打印服务端返回的数据(调试用)
        window.location.href = "/login";
    },
    error: function(message){
        var json = message.responseJSON;
        console.log(json);
        alert(json.message);
        if (json.status == 401){
            window.location.href="/login"
        }
    }
});
```

Figure 5. Web Server Response

Back to the case, API server also provides service for auditing where it connects all the events into a locally deployed Alibaba Druid Module (shown in *figure 6*), which allows administrators to check the suspect requests.



```
## database configuration
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://[redacted] serverTimezone=Australia/Sydney
spring.datasource.username=[redacted]
spring.datasource.password=[redacted]
## druid watch settings
spring.datasource.druid.stat-view-servlet.enabled=true
spring.datasource.druid.stat-view-servlet.url-pattern=/druid/*
spring.datasource.druid.stat-view-servlet.reset-enable=true
spring.datasource.druid.stat-view-servlet.login-username=aecl
spring.datasource.druid.stat-view-servlet.login-password=admin
# druid connection pool configuration
spring.datasource.druid.initial-size=5
spring.datasource.druid.min-idle=5
spring.datasource.druid.max-active=20
spring.datasource.druid.max-wait=60000
spring.datasource.druid.time-between-eviction-runs-millis=60000
spring.datasource.druid.min-evictable-idle-time-millis=300000
spring.datasource.druid.validation-query=SELECT 1 FROM DUAL
spring.datasource.druid.test-while-idle=true
spring.datasource.druid.test-on-borrow=false
spring.datasource.druid.test-on-return=false
# PScache configuration
spring.datasource.druid.pool-prepared-statements=true
spring.datasource.druid.max-pool-prepared-statement-per-connection-size=20
```

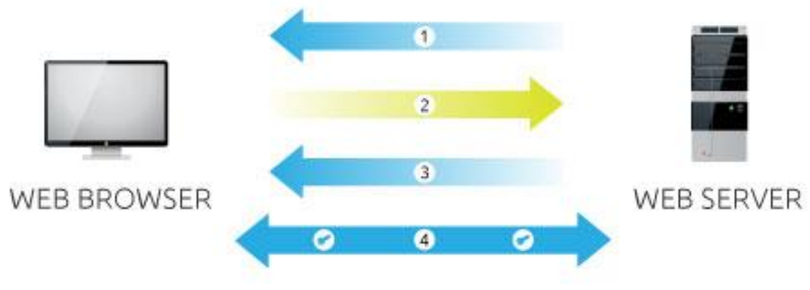
Figure 6. Configuration of Druid Monitoring Service

4. Security Decisions & Security Goals

In order to ensure user privacy that provides anonymity for better discussion, restrict access to administrative functions to authorized personnel, protect Aecl against interruption when supporting study environment for students, and enable to initiate root analysis of incidents and deterred interruption, the corresponding security goals of confidentiality, integrity, availability, access control and accountability are designed to be fulfilled majorly by HTTPS , API and database security.

DATA TRANSMISSION SECURITY: HTTPS

For confidentiality and integrity of the information, we implemented HTTPS for securing communication between different users and the website. To achieve higher performance, SSL was applied with both asymmetric and symmetric encryption (shown in *figure 7*).



1. Server sends public key.
2. Browser uses public key sent by server to encrypt its symmetric key and send back to server.
3. Server decrypts the public key by private key and get the symmetric session key sent by browser.
4. Server and browser then using symmetric key to communicate.

Figure 7. HTTPS

In asymmetric algorithm, there are two choices RSA and ECC, and we need to decide which algorithm is much more satisfier for us. Generally, RSA is very safe and secure. It has been used since 2000 so it is very complete and deserving trust, and RSA algorithm is very hard for normal computer to crack since it involves factorization of big prime numbers. Also, it has side effects, that RSA can be very slow in cases where large data needs to be encrypted. In another way, ECC employs a relatively short encryption key which is faster and requires less computing power. However, the disadvantages lie in the fact that not all services and applications have now implemented ECC algorithm and it is not as common as RSA. Furthermore, though ECC reduces the size of public key, ECC increases the size of the encrypted message greatly more than RSA[1]. After comparison, we finally choose RSA to deal with our SSL settings. Our reasons are as following:

- RSA is more acceptable than ECC. Because RSA is commonly supported by most of websites and application, and it can handle most of situations and may be used by most groups.
- We do not have too large data to encrypt so that even RSA have difficulty handling too much data, we can still use it the project.
- ECC algorithms is very complex, which increases the likelihood of implementation errors, thereby reducing the security of the algorithm.

We applied a domain name first and use this domain name to apply our SSL certificate. *TRUSTED ASIA* is selected since it is reputable, certificated by international and successfully provide SSL Certificate for many years. The level of our SSL is DV that it is unnecessary to purchase OV or EV because our websites are relatively small and are not using for business. After retrieving the certificate (public-private key pair) applied, we encrypted the private key by another long randomised key, which further ensures the certificate security. Lastly, we use gunicorn to deploy our https in frontend and spring boot for our backend

DATA SECURITY WITHIN API SERVER

API server is the most important server in our system, not only does it act as the middleware communicating database and web server, but it is the core entry point to handle the service requested by the user. Within the server, the data management architecture is divided to three different layers: Controller, DAO (Data Access Object) and Service. Controller layer mainly handles all the requests sent to the API server from Web Server, while DAO layer is responsible for the interaction between API server and database server and Service layer is used to process the

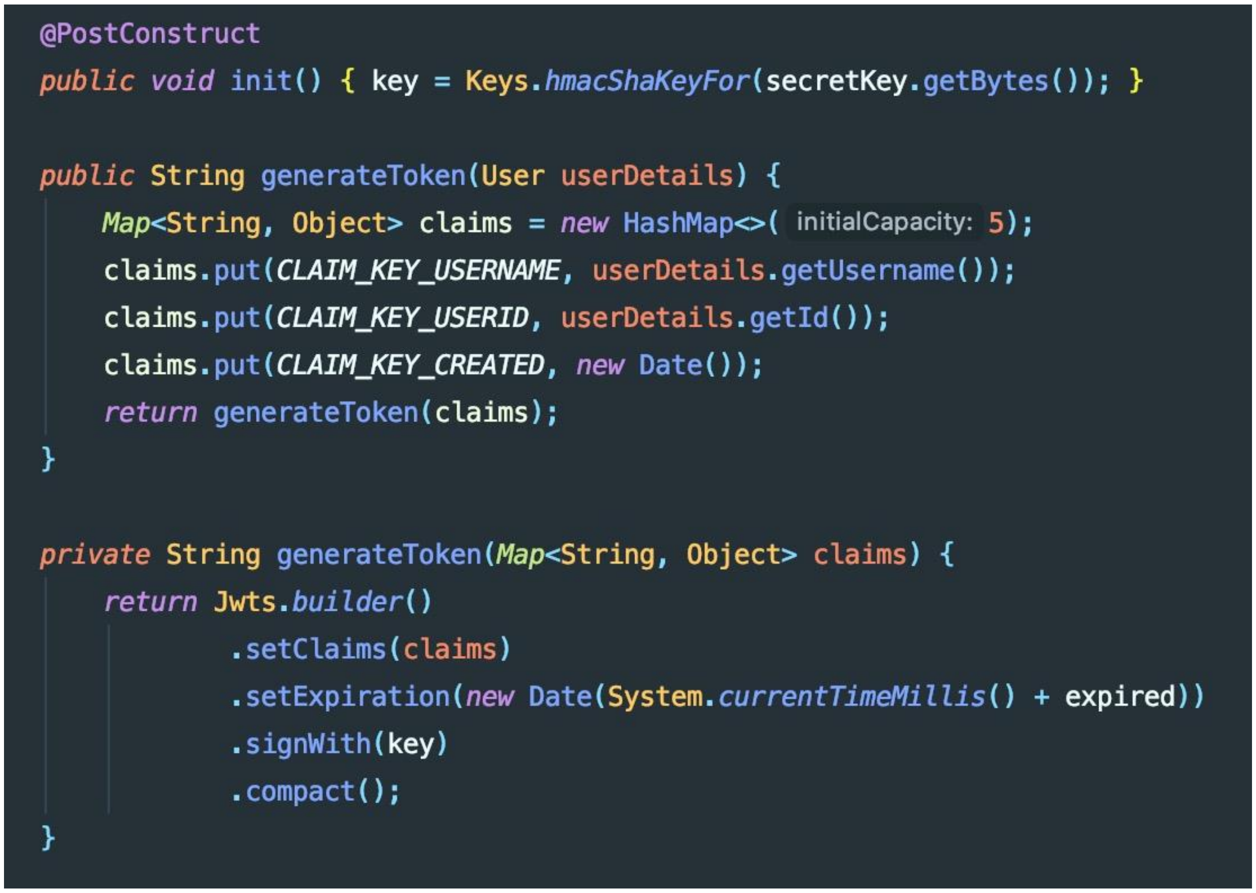
[1] (Dunning, 2017)

service and forward the data. This request interface is implemented following the RESTful API design principle by Microsoft[2]. All the requests to the specific URL are classified to four categories, which are GET, POST, PATCH and DELETE. In this way, the interface is overloaded and will respond differently according to the respective request type. The detailed documents about the API design can be referenced here in the website (see the first *appendix*). Besides, to ensure the accountability and for the purpose of auditing, we introduced a sophisticated monitoring module in our web system, Alibaba Druid. This module is an open-source database monitoring tool developed by Alibaba group. We configured this module delicately, deployed completely locally and set up a URL for visiting (see the second *appendix*). The visiting credentials are "aecl:admin" ($username:$password).

Also, to ensure the data integrity and authenticity, we implemented an authentication measure within controller layer. All the request to RESTful APIs except registration and log-in APIs are intercepted by a filter first to go through this authentication process. The filter will check the XMLHttpHeader transmitted to the controller to find the "Authorization token". This authorisation token is an implementation of JWT (JSON Web Token) generated dynamically by API server when user logged in. Specifically, when the user successfully logged in, the API server will create a temporary User object within which it records the validity, effective time, user ID and username. It should be noted that we will not record any sensitive data including messages, password etc. in this User Object. Access control is also implemented during this process. We separate the administrator and normal user by split away the login page into two different sections. As *figure 10* shown below, the gateway of authentication of administrator is different from general users like students. it is easy to see the user information are processed separately amid different individuals with object "PersonInfo". There is specific object "Admin" set for higher level of management of the site. By recognizing different roles of the user, we will add relevant authentication to the temporary User Object. After the creation and authorization of the User object, we generated the JWT and sent it to the Web server. Therefore, within our JWT token there is no sensitive information like password or other private information.

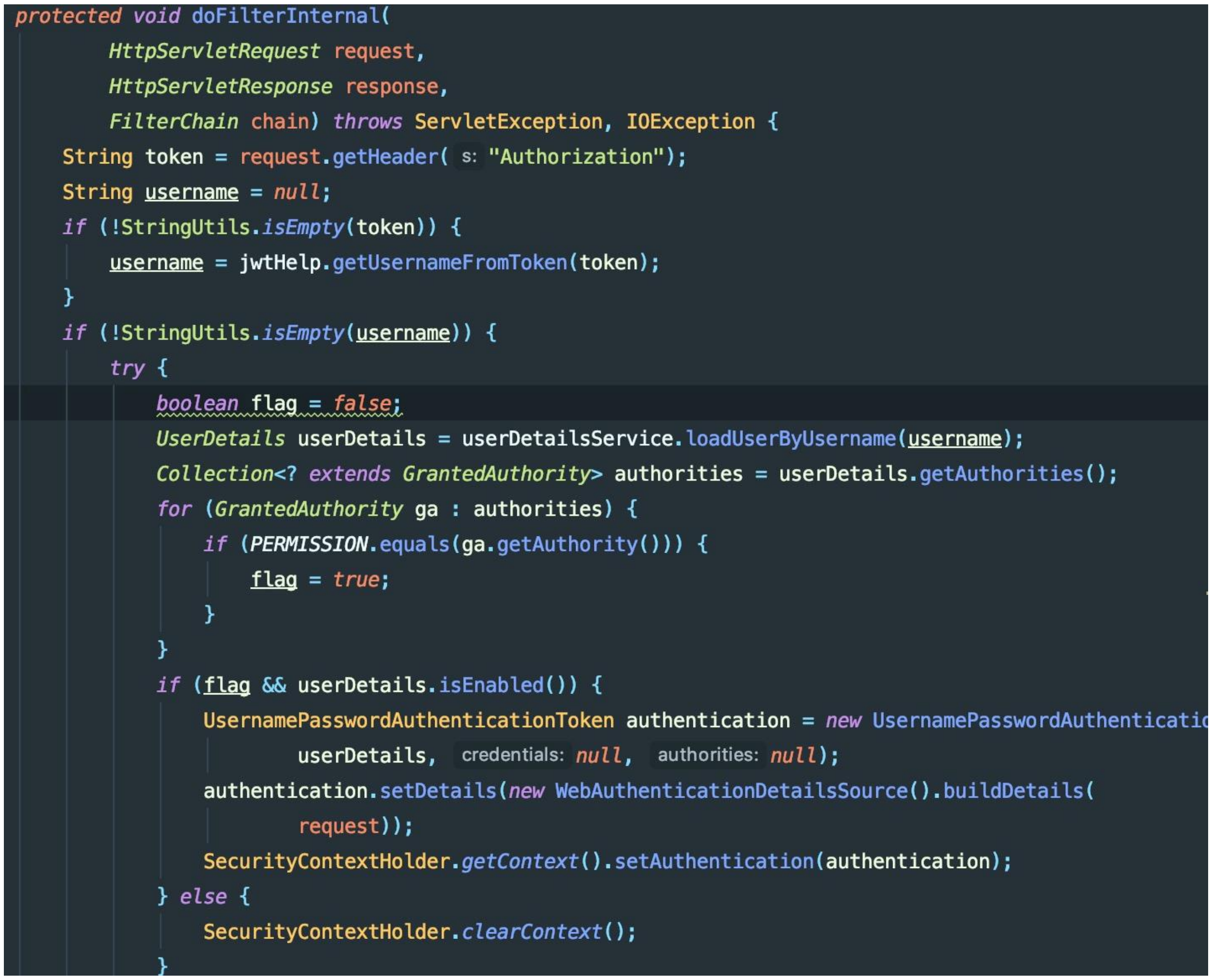
[2] ("API design guidance - Best practices for cloud applications", 2020)

The reason why we are using JWT instead of cookies, is that, our web system entails with multiple different applications, and the cookies are all cross-domain which makes the website very susceptible to CSRF (Cross Site Request Forgery) attack. Therefore, we only used XMLHttpHeader to transmit JWT token, and the only cookies we used in web server is for local storage only, hence, the Web server will not send any cookies to API server. Moreover, the JWT tokens are encrypted with HMAC-SHA-256 algorithm by a randomised 512-bit secret key. In this way, it is extremely difficult for hackers to decrypt the JWT token and accessed the content of the token. Or if the hacker is trying to modify the tokens, API server will check the token by decrypting it with private secret key (reference to bottom part of *figure 9*). If the token is corrupted, it will be recognized as unauthorized and declined the requests by reporting error to web server.



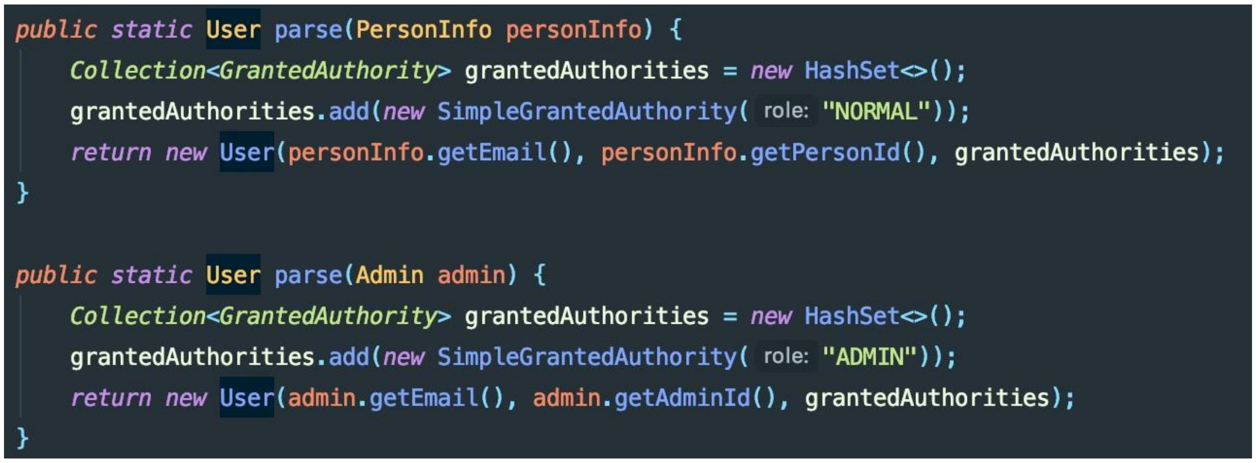
```
@PostConstruct
public void init() { key = Keys.hmacShaKeyFor(secretKey.getBytes()); }

public String generateToken(User userDetails) {
    Map<String, Object> claims = new HashMap<>( initialCapacity: 5);
    claims.put(CLAIM_KEY_USERNAME, userDetails.getUsername());
    claims.put(CLAIM_KEY_USERID, userDetails.getId());
    claims.put(CLAIM_KEY_CREATED, new Date());
    return generateToken(claims);
}

private String generateToken(Map<String, Object> claims) {
    return Jwts.builder()
            .setClaims(claims)
            .setExpiration(new Date(System.currentTimeMillis() + expired))
            .signWith(key)
            .compact();
}
```

Figure 8. Encryption of Token



```
protected void doFilterInternal(
        HttpServletRequest request,
        HttpServletResponse response,
        FilterChain chain) throws ServletException, IOException {
    String token = request.getHeader( s: "Authorization");
    String username = null;
    if (!StringUtils.isEmpty(token)) {
        username = jwtHelp.getUsernameFromToken(token);
    }
    if (!StringUtils.isEmpty(username)) {
        try {
            boolean flag = false;
            UserDetails userDetails = userDetailsService.loadUserByUsername(username);
            Collection<? extends GrantedAuthority> authorities = userDetails.getAuthorities();
            for (GrantedAuthority ga : authorities) {
                if (PERMISSION.equals(ga.getAuthority())) {
                    flag = true;
                }
            }
            if (flag && userDetails.isEnabled()) {
                UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticati
                        userDetails,  credentials: null,  authorities: null);
                authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(
                        request));
                SecurityContextHolder.getContext().setAuthentication(authentication);
            } else {
                SecurityContextHolder.clearContext();
            }
```

Figure 9. Authentication of Token Log



```
public static User parse(PersonInfo personInfo) {
    Collection<GrantedAuthority> grantedAuthorities = new HashSet<>();
    grantedAuthorities.add(new SimpleGrantedAuthority( role: "NORMAL"));
    return new User(personInfo.getEmail(), personInfo.getPersonId(), grantedAuthorities);
}

public static User parse(Admin admin) {
    Collection<GrantedAuthority> grantedAuthorities = new HashSet<>();
    grantedAuthorities.add(new SimpleGrantedAuthority( role: "ADMIN"));
    return new User(admin.getEmail(), admin.getAdminId(), grantedAuthorities);
}
```

Figure10. Admin Role

DATABASE SECURITY

Since SQL is vulnerable to SQL Injection, that potential attackers can bypass endpoint defenses, and retrieve data, make modification easily when the database is invaded. Thus, we used some special MySQL user settings and Bcrypt hashing technique to achieve SQL injection prevention.

At first, instead of attaching MySQL root user to our website, we created a new user called ‘aecl’ which has less authority compared with the root user. With the user ‘aecl’, we can only execute selecting/updating/deleting/inserting instructions to data in the database. Other people have no right to do destructive things like dropping a table or modify names of its columns.

Rather than putting real passwords into our database, we store hashed passwords with the help of Bcrypt hashing function. Bcrypt is an algorithm based on the blowfish cipher which was specifically designed for handling passwords. It has both salts and work factors built in. So, compared with md5, it is much safer. The crypted key strengthening makes a password more secure against brute force attacks. Even though someone gains the access to our database, it can be extremely hard for him to get our real passwords.

To prevent SQL injection, we also create three layers (Entity/ mapper /Service) for our database in Java. We connect Mapper to SQL instructions in .xml files (mybatis) and use Service to store Encapsulation of methods in Mapper. We store concrete classes in Entity.

5. References

API design guidance - Best practices for cloud applications. (2020). Retrieved from https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

Dunning, D. (2017). What Are the Advantages & Disadvantages of Elliptic Curve Cryptography for Wireless Security?. Retrieved from https://www.techwalla.com/articles/what-are-the-advantages-disadvantages-of-elliptic-curve-cryptography-for-wireless-security

6. Appendix

- API: https://10.86.164.216:8080/swagger-ui.html
- Alibaba druid: https://10.86.164.216:8080/druid
- GitHub repository: https://github.sydney.edu.au/weli6728/Aecl