최종보고서

〈 Knowledge Based Matching System 〉

최종 제출일 : 2015.06.17.

과목	소프트웨어공학
지도교수	최윤자 교수님
팀명	89s
	김낙정 (2012105110)
	라채현 (2012105112)

< 목 차 >

< 그림 목차 >

1. 프로젝트 개요

최근 산학 협력에 의해 회사와 학생들이 서로 win-win 할 수 있는 인턴쉽 제도가 많이 마련되어있다. 이때 회사는 업무에 알맞은 학생들을 인턴으로 채용하기를 원하고, 학생들은 인턴쉽 기회를 보다 많이 제공 받기를 원한다.

이러한 요구사항을 만족시키기 위해서 학과사무실을 통해 회사가 인턴 공고를 제출할 수 있고, 학생이 공고들을 확인하고 편리하게 인턴에 지원할 수 있도록 해주는 시스템 개발을 목표로 한다.

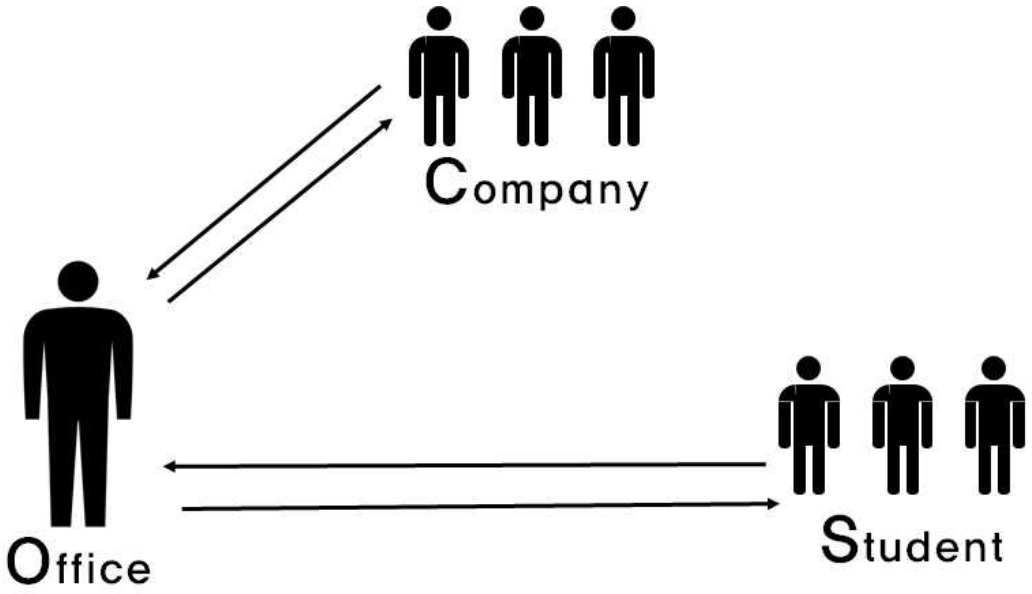
설 명

2인 소규모 팀으로 진행하는 프로젝트인 만큼 문서화 작업을 간략하게 시행하고, 리팩토링을 통한 코드의 품질을 높이는 것을 목표로 한다. 또, 기존 프로젝트에서 재사용 할 수 있는 컴포넌트를 최대한 활용하면서 새로운 기능을 추가하기 위하여 Incremental Development 개발 모델을 적용한다.

개발 범위는 크게 Server 와 Client 파트로 나뉜다. Customer에게 중간개발 결과를 Incrementally delivery 하고 피드백 받는 것을 모방하여 팀원 당 각각 한 명이 해당 파트에 대해 요구사항 도출하고, 상대방이 도출한 요구사항에 맞춰 나머지 팀원이 개발을 진행한다. 부분적인 개발이 끝난 뒤, 요구사항 대로 잘 개발 되었는지 팀원에게 피드백을 받고 수정한다.

2. 기존 프로젝트 설명 및 개발 현황

기존 프로젝트는 Server 역할을 하는 Office Manager와 Client 역할을 하는 Company, Student로 이루어져 있다. 아래에서 Server와 Client의 클래스 다이어그램을 확인하고 기존 프로젝트의 실행 결과 모습을 확인 할 수 있다.

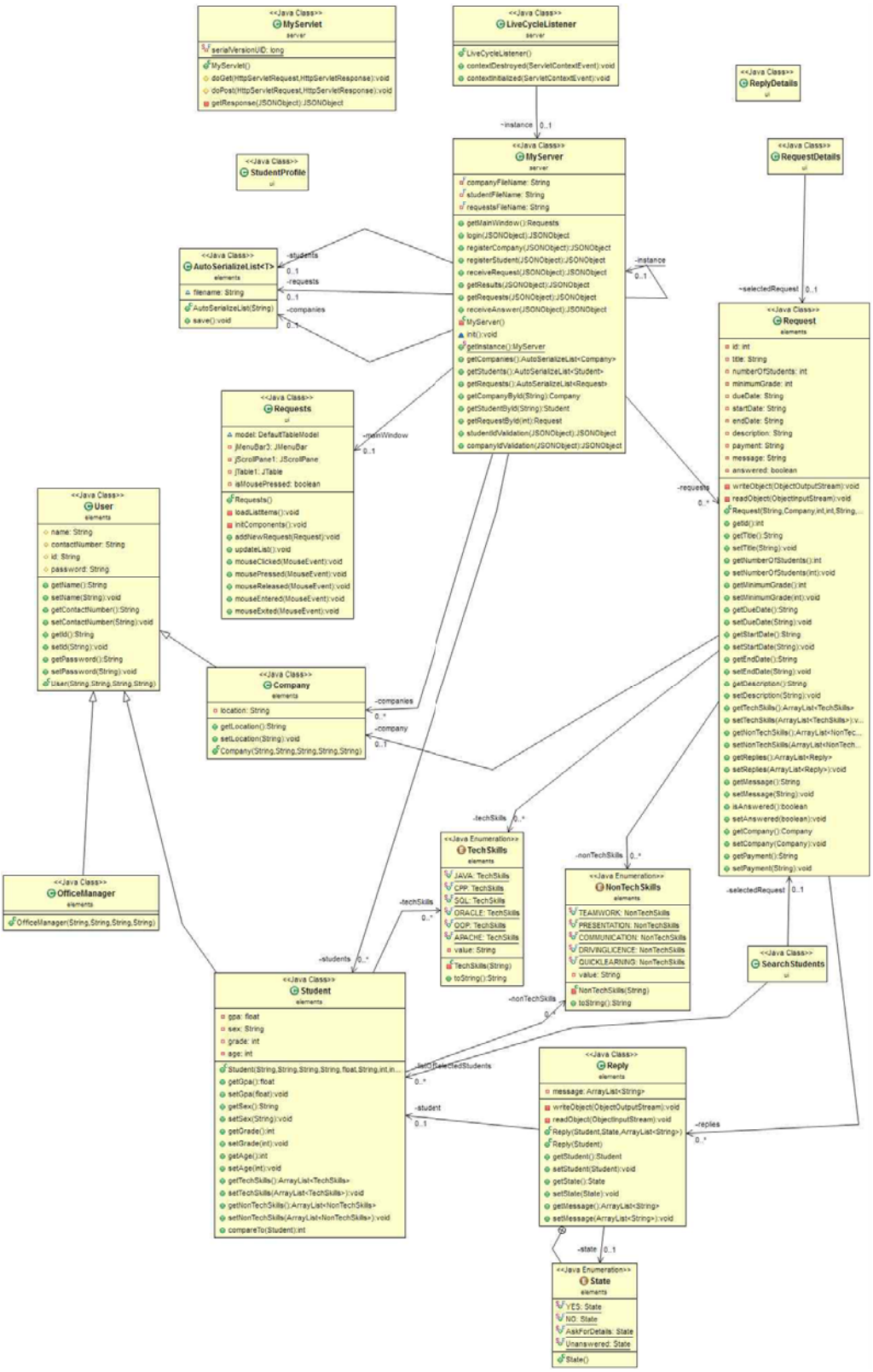


<그림 2-a. 프로젝트 설명>

2-1. 프로그램 기능

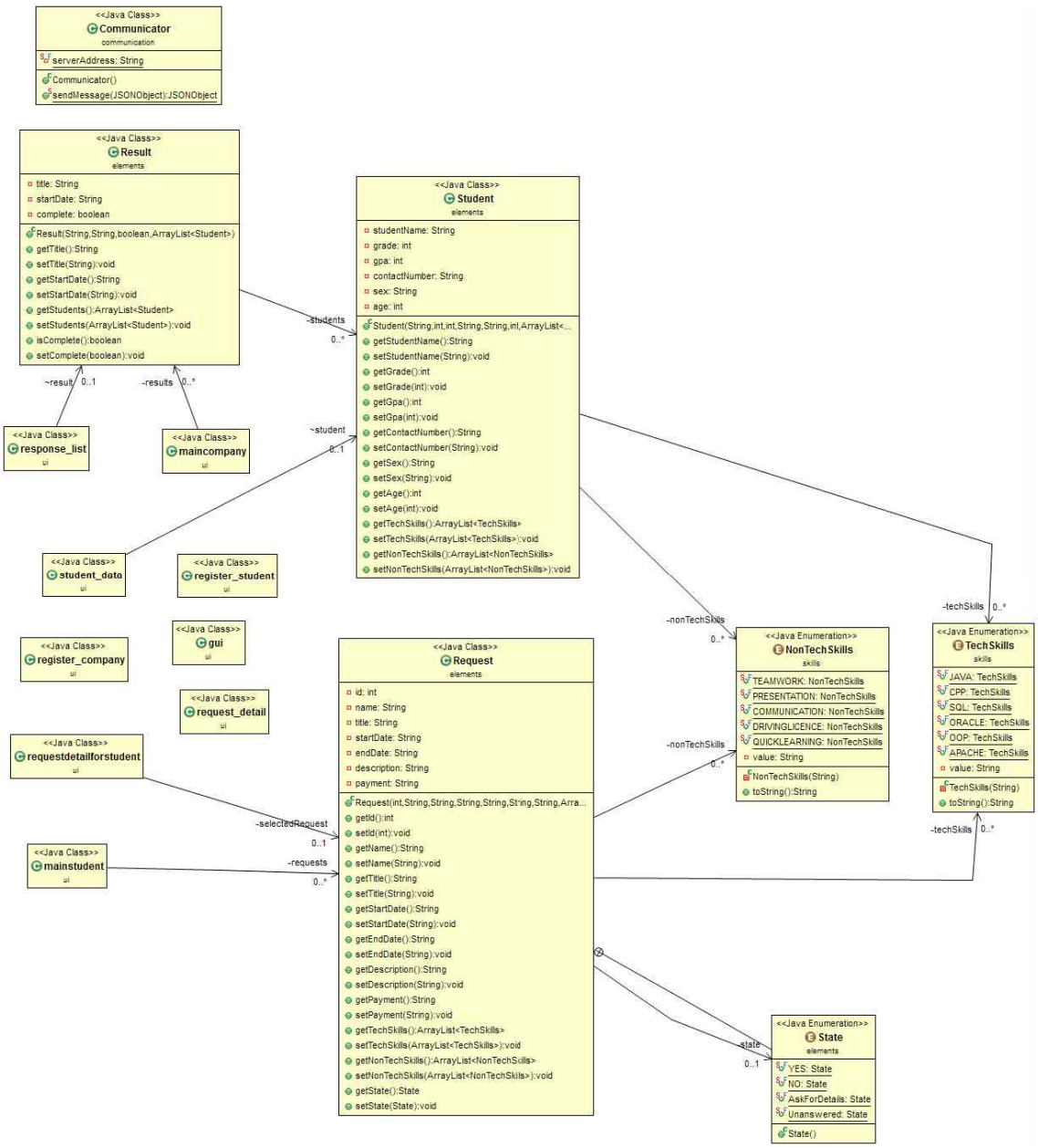
① 해당 프로그램을 사용하면, Company는 자사의 프로젝트에 적합한 학생들을 인턴으로 모집하기 위한 공고를 Office에게 제출 할 수 있다.
② Office는 Company로부터 인턴모집 공고를 받을 수 있고, 요구조건에 부합하는 학생들을 쉽게 검색 할 수 있다.
③ Office는 요구조건에 부합하는 학생들에게 인턴모집 공고에 대해 알려줄 수 있다.
④ 학생들은 Office로부터 받은 인턴모집 공고에 대하여 응답 할 수 있다.

2-2. Server Class Diagram



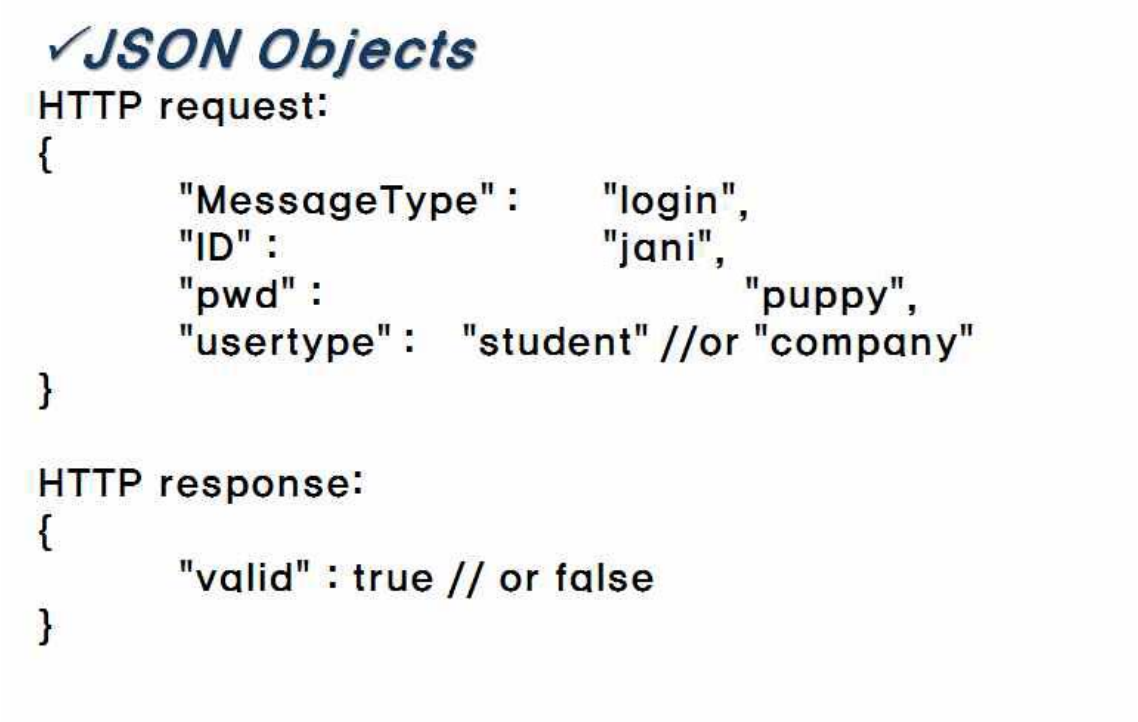
<그림 2-2-a. Server Class Diagram>

2-3. Client Class Diagram



<그림 2-3-a. Client Class Diagram>

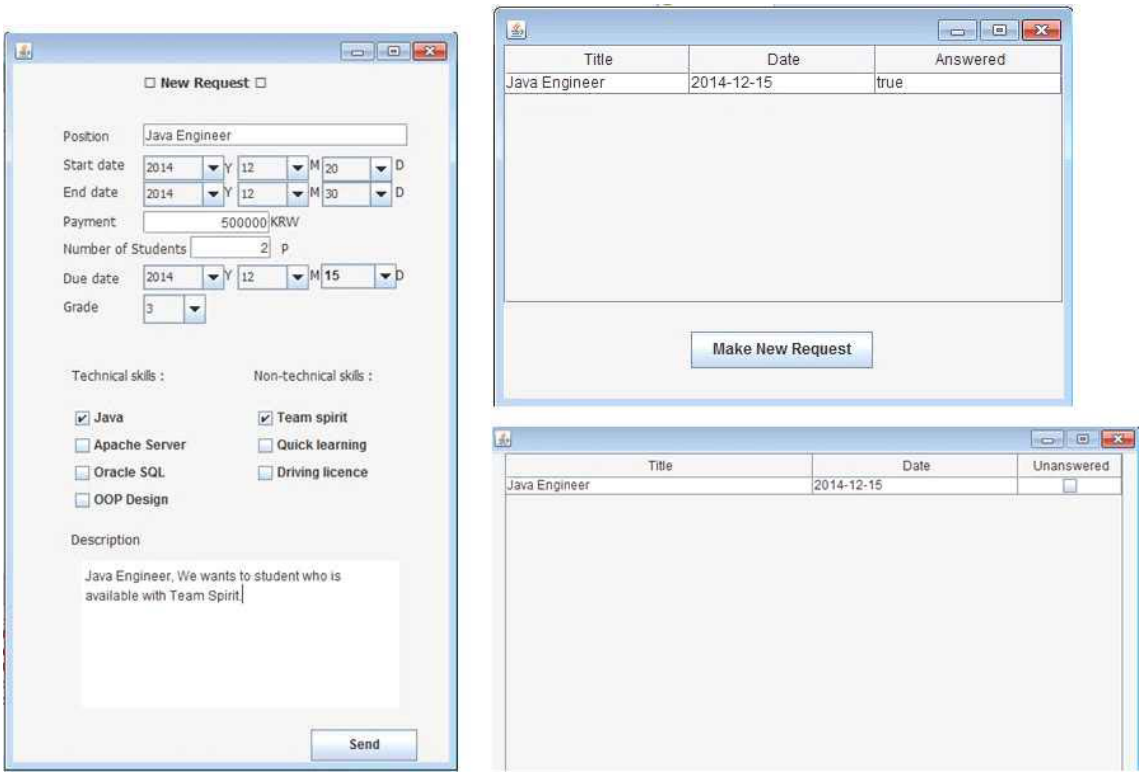
2-4. Server와 Client의 통신방법



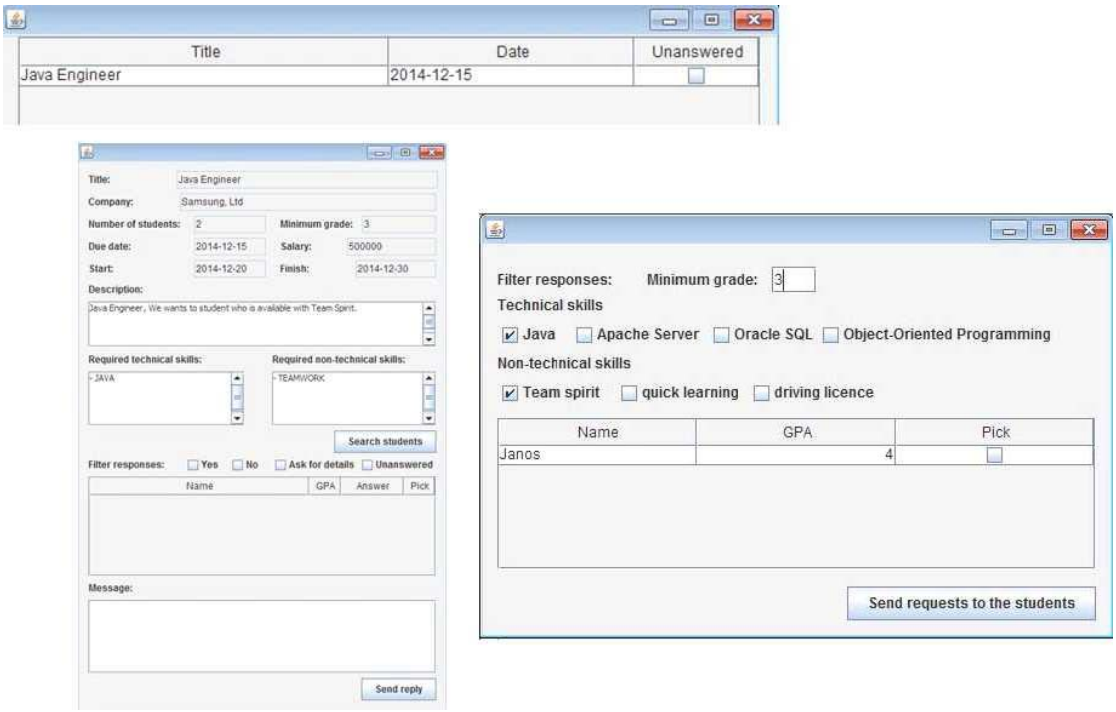
<그림 2-4-a. JSON Object Format>

서버는 아파치 톰캣을 사용한 HTTP Server로 구현되었으며, 클라이언트와의 통신은 JSON Objects를 주고받음으로서 진행된다.

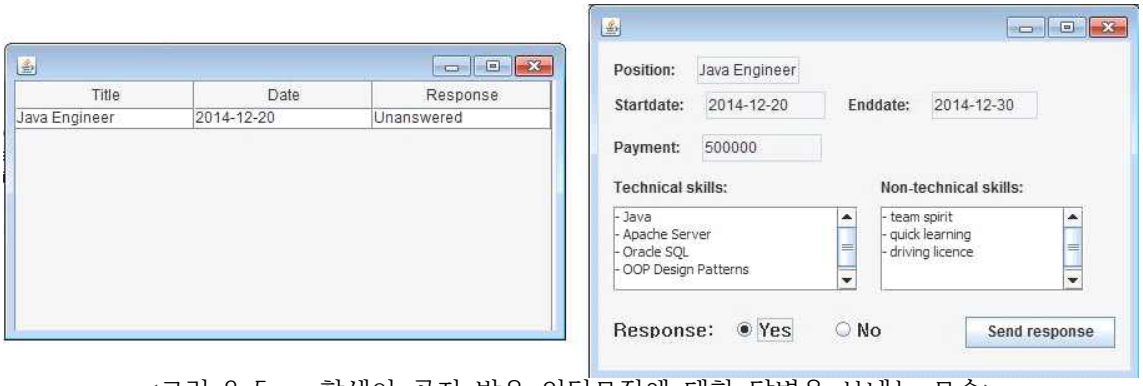
2-5. 기존 프로젝트의 실행 결과



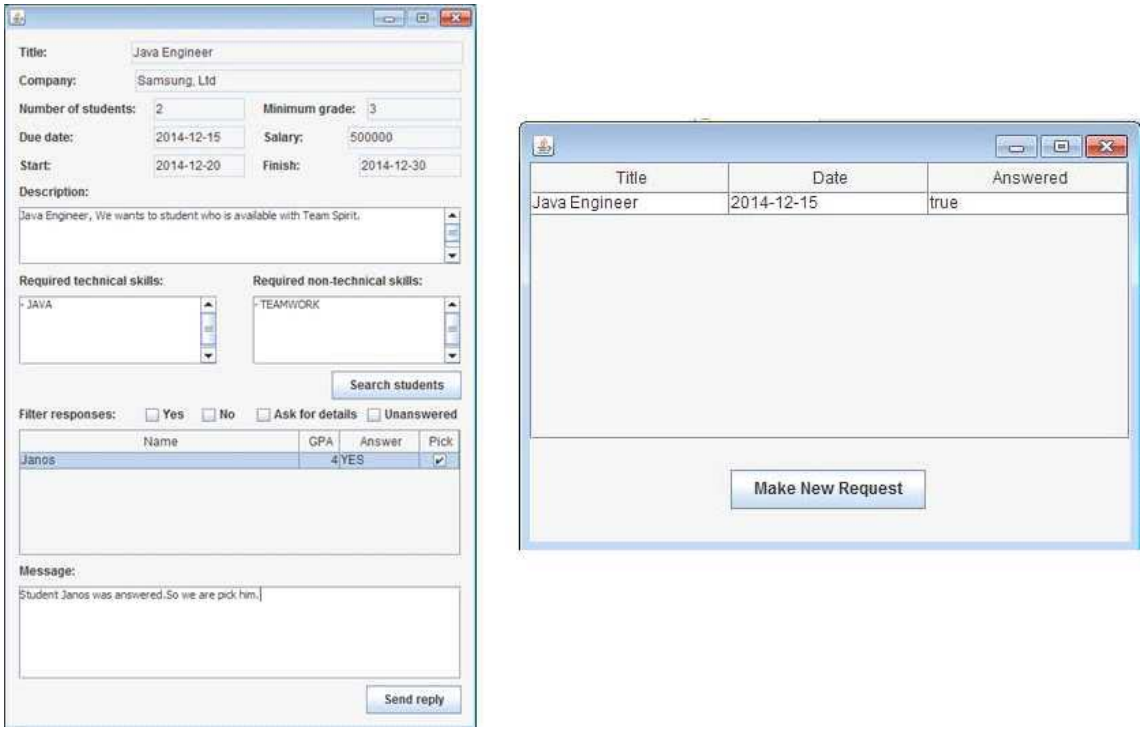
<그림 2-5-a. Company가 인턴 모집 공고를 요청하는 모습>



<그림 2-5-b. Office에서 학생에게 인턴모집 공고를 알려주는 모습>



<그림 2-5-c. 학생이 공지 받은 인턴모집에 대한 답변을 보내는 모습>



<그림 2-5-d. Company에서 인턴모집 공고에 대한 결과를 확인하는 모습>

3. 기존 프로젝트에서의 개선사항 목표

기존 프로젝트에서 큰 기능위주의 테스트 시 큰 문제가 발견되지 않았으나, 세세한 부분에서 미흡한 점이 여럿 발견 되었다.

그리하여 기존 프로젝트에서 처리되지 않은 에러들을 핸들링하고 인턴 공고 매칭 시스템이라는 주제에 걸맞게 추가 편의사항을 구현하는 것을 목표로 프로젝트를 진행하였다.

3-1. 문제점 분석

① 회원가입 시에 정보입력란에 범위를 벗어난 인풋이 입력 될 경우 프로그램 종료
② 인턴 모집 공고를 보낼 때 범위를 벗어난 인풋이 입력 될 경우 프로그램 종료
③ 최초 회원가입 시 입력한 정보를 수정하지 못하는 문제
④ 하나의 계정으로 중복 로그인이 되는 문제
⑤ GUI를 담당하는 클래스에 제어에 해당하는 코드가 하나로 묶여있는 문제
⑥ 연관 관계가 없는 독립된 클래스가 존재

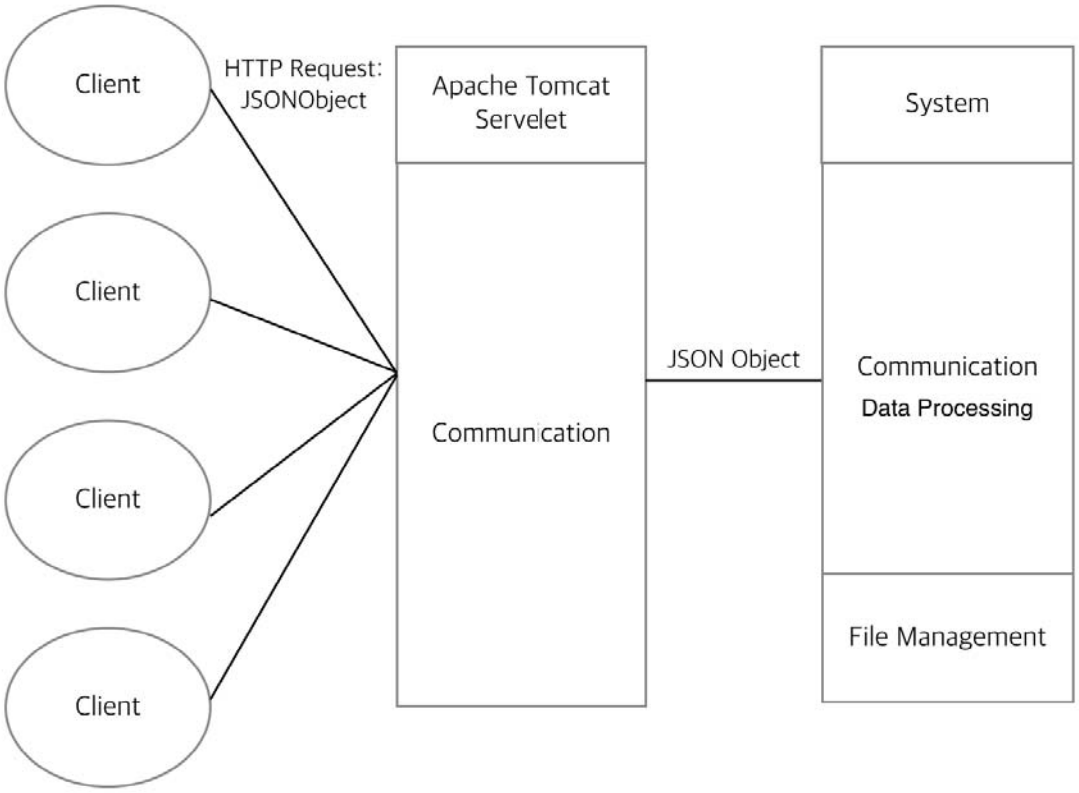
3-2. 프로젝트 개선을 위한 요구사항 도출

① 회원가입, 인턴 모집 공고 등 프로그램 내 입력란에 해당 하는 필드에서 입력 값에 대한 에러 핸들링을 한다.
② 사용자의 정보 갱신을 위한 개인정보 수정 기능을 추가
③ 안전한 시스템 종료를 위해 서버 , 클라이언트 각각 종료버튼 추가
④ 시스템 Reliability를 위해 중복 로그인 방지 기능 추가
⑤ 사용자가 항상 최신 버전의 프로그램을 사용 하도록 버전체크 기능 추가
⑥ 안전성을 위해 Server 측에서 Log File을 기록하는 기능을 추가
⑦ 유지보수를 위해 기존 클래스 구성 및 연관 관계를 수정
⑧ 사용 편의성을 위해 Office는 현재 가입된 Company의 정보를 직접 열람하는 기능 추가
⑨ 문의사항 문의를 위한 Company와 Student 간의 메시지 전송/확인 기능 추가
⑩ Progress Bar를 추가하여 프로그램 동작을 확인하는 기능

4. 아키텍쳐 및 설계

4-1. 서버-클라이언트 아키텍쳐

기존 프로젝트에서 사용한 Apache Tomcat을 이용한 HTTP Server 구조를 이용한다.
서버와 클라이언트는 JSON Object를 매개체로 하여 통신을 한다.

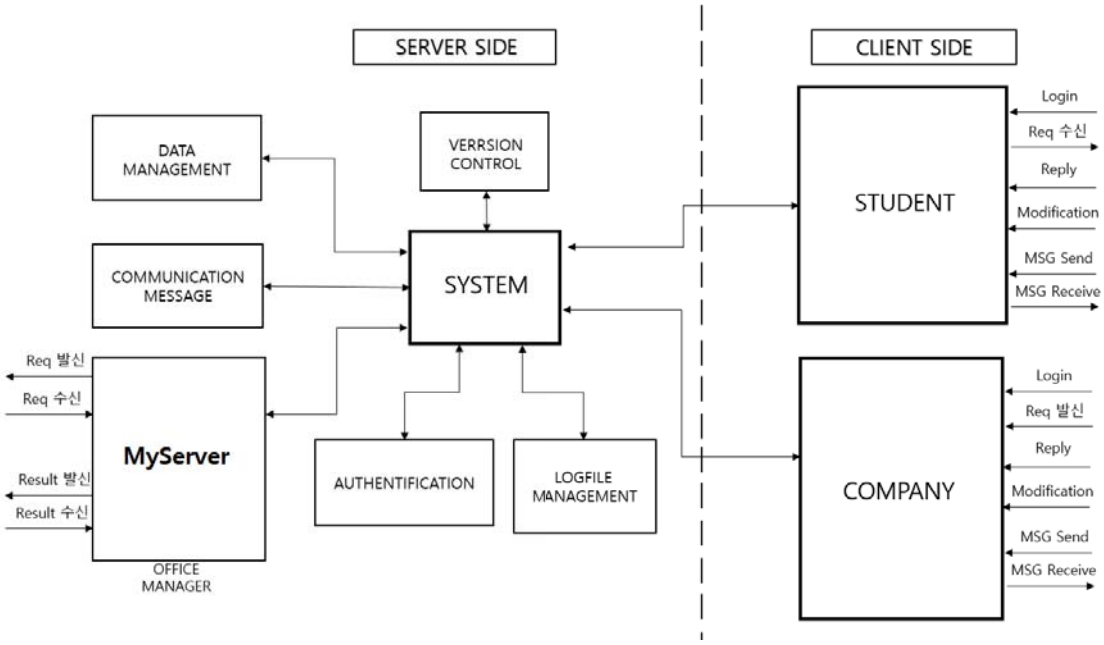


<그림 4-1-a. 프로젝트 아키텍쳐>

전형적인 Server-Client 모델로써 Office가 서버 역할을 하며, 다수의 클라이언트(Company, Student)에게 알맞은 서비스를 제공해준다.
Tomcat Servelet을 이용하게 되므로, 1 대 N 접속이 원활하게 가능하며, System에서는 톰캣을 거쳐 각 서버-클라이언트 1:1 통신 관계로 정의되어 있다.

시스템에서 복잡한 연산이 필요하지 않으며 서버와 다수의 클라이언트가 명확하게 메시지를 주고받는 기능이 핵심이라고 판단하여 서버-클라이언트 아키텍쳐가 가장 목적에 부합하는 모델이라고 판단하였다.

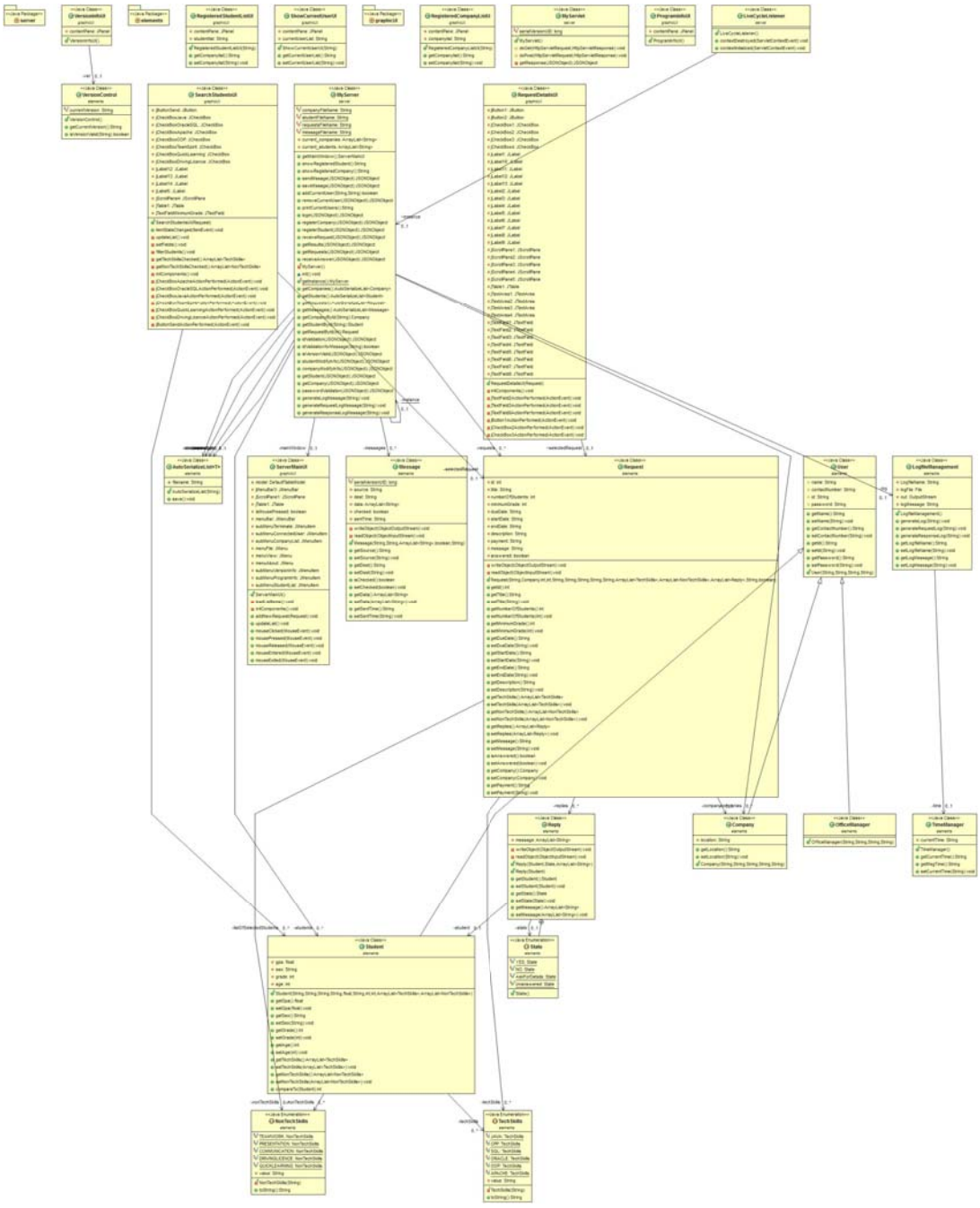
4-2. 설계



<그림 4-2-a. 프로젝트 설계>

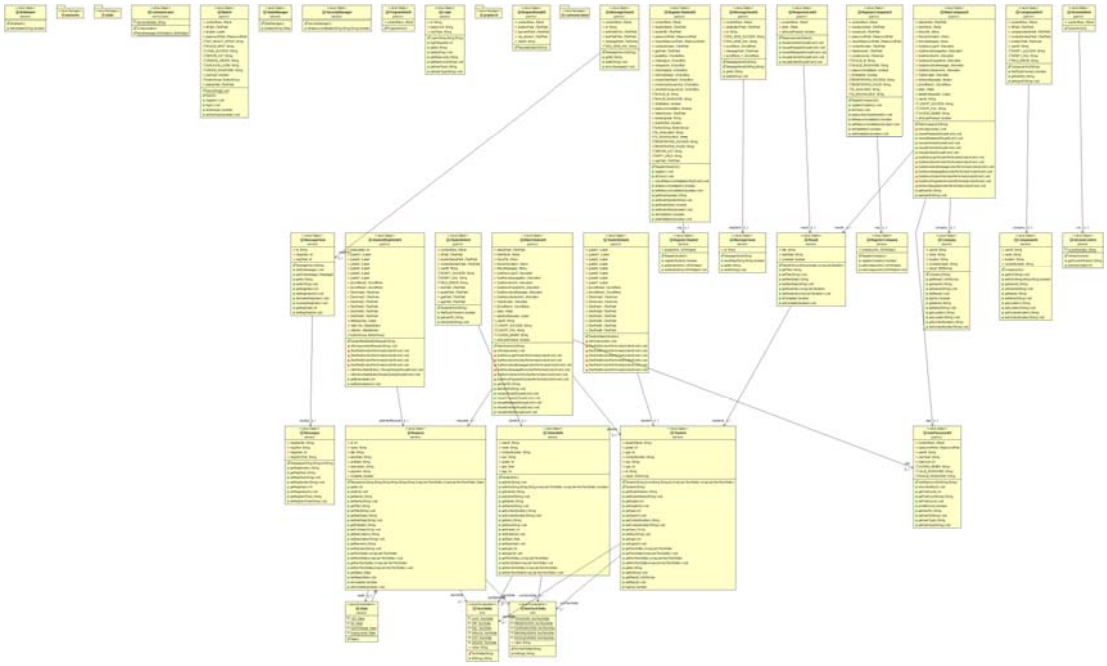
설계는 크게 각 기능을 중점으로 클래스로 구분되어있다. 기존의 프로젝트에서 구현 되어 있는 MyServer 클래스를 기준점으로 주요 추가 기능인 Version Control, LogFile Management, Login Authentification, Communication Message을 독립적인 컴포넌트로 구성하였다. 또 기존 프로젝트 코드의 구조적인 문제점인 GUI와 제어연산부분 코드를 독립적으로 분리시키는 작업도 진행하였다. 클라이언트 쪽은 서버에게 서비스를 요청하고 모든 연산을 서버가 책임지며 그 결과를 리턴 받아 사용자에게 알려주는 역할만을 담당하게 된다.

4-3. Server Class Diagram



<그림 4-3-a. Server Class Diagram>

4-4. Client Class Diagram



<그림 4-4-a. Client Class Diagram>

※ Server와 Client Class Diagram에서 Relation이 없는 클래스가 존재하는 이유는, GUI와 관련된 클래스에서 이러한 현상이 나타나고 있는데, 코딩을 하는 단계에서 GUI를 Attribute로 두고 사용한 것이 아니라, 매번 새로운 GUI가 생성되어야 하는 필요에 의해 (new GUI()).setVisible(true)의 형태로 코딩하였기 때문에 위와 같은 현상이 발생하였음.

5. 테스트 케이스 작성

기존 프로젝트를 유지보수 함에 있어서 작성 되어 있는 테스트 케이스가 존재하지 않아 여러 가지로 어려움이 많았다. 먼저 각 기능이 어떻게 관련되어있는지 파악하기 어려웠고 하나의 기능을 수정 시에 다른 기능에 영향을 미치지 않는지 파악하기도 어려웠다.
이번에 프로젝트를 Incremental Development로 진행하면서 명확하고 세세한 Test Case의 중요성을 체감 할 수 있었다.

5-1. 테스트 케이스 작성

5-1-1. 입력 값 필드에 대한 에러처리

Company Register : ID duplication check

INPUT	ID : abc
TEST PROCEDURE	ID : abc , PW : abc로 회원가입을 마친다. 다시 회원가입을 시도한다. ID Field에 abc를 입력하고 ID Check 버튼을 누른다.
EXPECTED OUTPUT	이미 존재하는 ID라고 알림을 준다.

Company Register : Password & Retype Password

INPUT	1.abc / abc 2.abc / abcd
TEST PROCEDURE	1.회원가입 시 Password란에 abc를 입력한다. Retype PW란에 abc를 입력한다. 2.회원가입 시 Password란에 abc를 입력한다. Retype PW란에 abcd를 입력한다.
EXPECTED OUTPUT	1. valid Password 라고 알려준다. 2. Invalid Password 라고 알려준다.

Student Register : ID duplication check

INPUT	ID : kkk
TEST PROCEDURE	ID : kkk , PW : kkk로 회원가입을 마친다. 다시 회원가입을 시도한다. ID Field에 abc를 입력하고 ID Check 버튼을 누른다.
EXPECTED OUTPUT	이미 존재하는 ID라고 알림을 준다.

Student Register : Technical & Non-Technical Skills Check

INPUT	Technical & Non-Technical Skills를 제외한 나머지 부분을 모두 입력한다.
TEST PROCEDURE	1.Technical Skills만 체크하고 가입을 누른다. 2.Non-Technical Skills만 체크하고 가입을 누른다. 3.둘 다 체크하고 가입을 누른다.
EXPECTED OUTPUT	1번과 2번의 경우에는 가입이 되지 않는다. 3번의 경우 정상 가입이 된다.

Student Register : Password & Retype Password

INPUT	1.kkk / kkk 2.kkk / kka
TEST PROCEDURE	1.회원가입 시 Password란에 kkk를 입력한다. Retype PW란에 kkk를 입력한다. 2.회원가입 시 Password란에 abc를 입력한다. Retype PW란에 kka를 입력한다.
EXPECTED OUTPUT	1. valid Password 라고 알려준다. 2. Invalid Password 라고 알려준다.

Login Failure with Null input

INPUT	-
TEST PROCEDURE	ID와 비밀번호를 비워두고 로그인을 시도한다. Login 버튼을 클릭한다.
EXPECTED OUTPUT	ID와 비밀번호를 입력한 뒤에 로그인하라고 알려준다.

Login Failure with wrong password

INPUT	ID : abc , PW : abcd (원래 PW : abc)
TEST PROCEDURE	ID : abc , PW : abcd를 입력하여 Login 버튼을 클릭한다.
EXPECTED OUTPUT	ID와 PW가 일치하지 않는다고 알려준다.

5-1-2. 기능 테스트

Server & Client : Termination

INPUT	-
TEST PROCEDURE	메뉴 탭에서 File - Termination을 클릭한다. 서버 및 클라이언트가 정상적인 프로세스로 종료된다.
EXPECTED OUTPUT	서버와 클라이언트가 정상적으로 종료 된다.

Version Control

INPUT	-
TEST PROCEDURE	Client를 실행한다. 메뉴 탭에 About - Version Info를 클릭한다. Version Information 창이 실행되고 최신 버전을 확인한다.
EXPECTED OUTPUT	현재 사용 중인 버전과 최신버전이 일치하면 Latest Version. 일치하지 않으면 Update를 하라고 알려준다.

Server : Log File System

INPUT	-
TEST PROCEDURE	서버를 실행한다. 프로그램이 여러 가지 서비스를 제공한다. 서버를 종료한다.
EXPECTED OUTPUT	서버 종료 후 생성된 Log File에 서버가 켜져 있는 동안 접속한 Client 정보와 실행된 서비스들이 제대로 Log로 기록되어 있다.

Client : Duplicated Login Check

INPUT	이미 회원가입 된 ID : abc, PW : abc 계정 정보를 이용
TEST PROCEDURE	abc / abc 계정정보를 이용하여 1차 로그인을 시도한다. 로그인에 성공한다. 해당 프로그램을 종료하지 않은채로 다시 abc / abc 계정정보를 이용하여 2차 로그인을 시도한다.
EXPECTED OUTPUT	해당 계정은 이미 서버에 접속 중이라고 알려준다.

Progress Bar Test

INPUT	ID : abc , PW : abc를 입력하여 로그인한다.
TEST PROCEDURE	로그인 이전에 Progress bar는 Ready를 보여준다. abc / abc 계정정보를 이용하여 로그인을 시도한다.
EXPECTED OUTPUT	로그인을 시도하여 서버와 로그인을 위해 통신중일 때 Progress bar에 Login..을 보여준다. 로그인이 완료되면 서버와의 연결 상태를 알려준다.

Message System between Company and Student

INPUT	1. Company(ID : abc)가 Student(ID : kkk)에게 메시지 보내는 경우 2. Student가 Company에게 메시지 보내는 경우
TEST PROCEDURE	1-1. kkk에게 메시지를 전송한다. 1-2. kkkk에게 메시지를 전송한다. 2-1. abc에게 메시지를 전송한다. 2-2. abcd에게 메시지를 전송한다.
EXPECTED OUTPUT	1-1. kkk에게 정상적으로 메시지가 전송된다. 1-2. kkkk라는 ID는 Invalid하므로 전송 실패한다. 2-1. abc에게 정상적으로 메시지가 전송된다. 2-2. abcd라는 ID는 Invalid하므로 전송 실패한다.

5-2. JUnit Test Code

```
@Test
public final void testRegisterCompany() {
    MyServer server = getDummy();
    JSONObject message = new JSONObject();
    JSONObject response = null;

    try {
        message.put("CompanyName", "testCompany");
        message.put("ID", "testCompany");
        message.put("Password", "1234");
        message.put("Location", "Testcase");
        message.put("Contact number", "0123456");

        response = server.registerCompany(message);
    } catch (JSONException e) {
        e.printStackTrace();
        fail();
    }

    assertNotNull(response);

    try {
        assertEquals(true, response.getBoolean("valid"));
    } catch (JSONException e) {
        e.printStackTrace();
        fail();
    }

}
```

<그림 5-2-a. testRegisterCompany>

```
@Test
public final void testReceiveRequest() {
    MyServer server = getDummy();
    JSONObject message = new JSONObject();
    JSONObject response = null;

    try {
        message.put("ID", "test");
        message.put("Position", "test");
        message.put("StartDate", "2014-1-1");
        message.put("EndDate", "2015-1-1");
        message.put("Payment", "30000");
        message.put("NumberOfStudents",1);
        message.put("DueDate", "2014-2-1");
        message.put("Grade", 1);
        message.put("Description", "test case");
        message.append("TechSkills", TechSkills.JAVA.name());
        message.append("NonTechSkills", NonTechSkills.DRIVINGLICENCE.name());

        response = server.receiveRequest(message);
    } catch (JSONException e) {
        e.printStackTrace();
        fail();
    }

    assertNotNull(response);

    try {
        assertEquals(false, response.getBoolean("error"));
    } catch (JSONException e) {
        e.printStackTrace();
        fail();
    }
}
```

<그림 5-2-b. testReceiveRequest>

```java
@Test
public final void testStudentModifyInfo() {
    MyServer server = getDummy();
    JSONObject message = new JSONObject();
    JSONObject response = null;

    try {
        JSONObject element = new JSONObject();

        element.put("userID", "student");
        element.put("Name", "changetest");
        element.put("ContactNumber", "testcase");
        element.put("Gpa", 3.1);
        element.append("TechSkills", TechSkills.JAVA.name());
        element.append("NonTechSkills", NonTechSkills.COMMUNICATION.name());
        message.put("student", element);

        response = server.studentModifyInfo(message);
    } catch (JSONException e) {
        e.printStackTrace();
        fail();
    }

    assertNotNull(response);

    try {
        assertEquals(true, response.getBoolean("valid"));
    } catch (JSONException e) {
        e.printStackTrace();
        fail();
    }
}
```

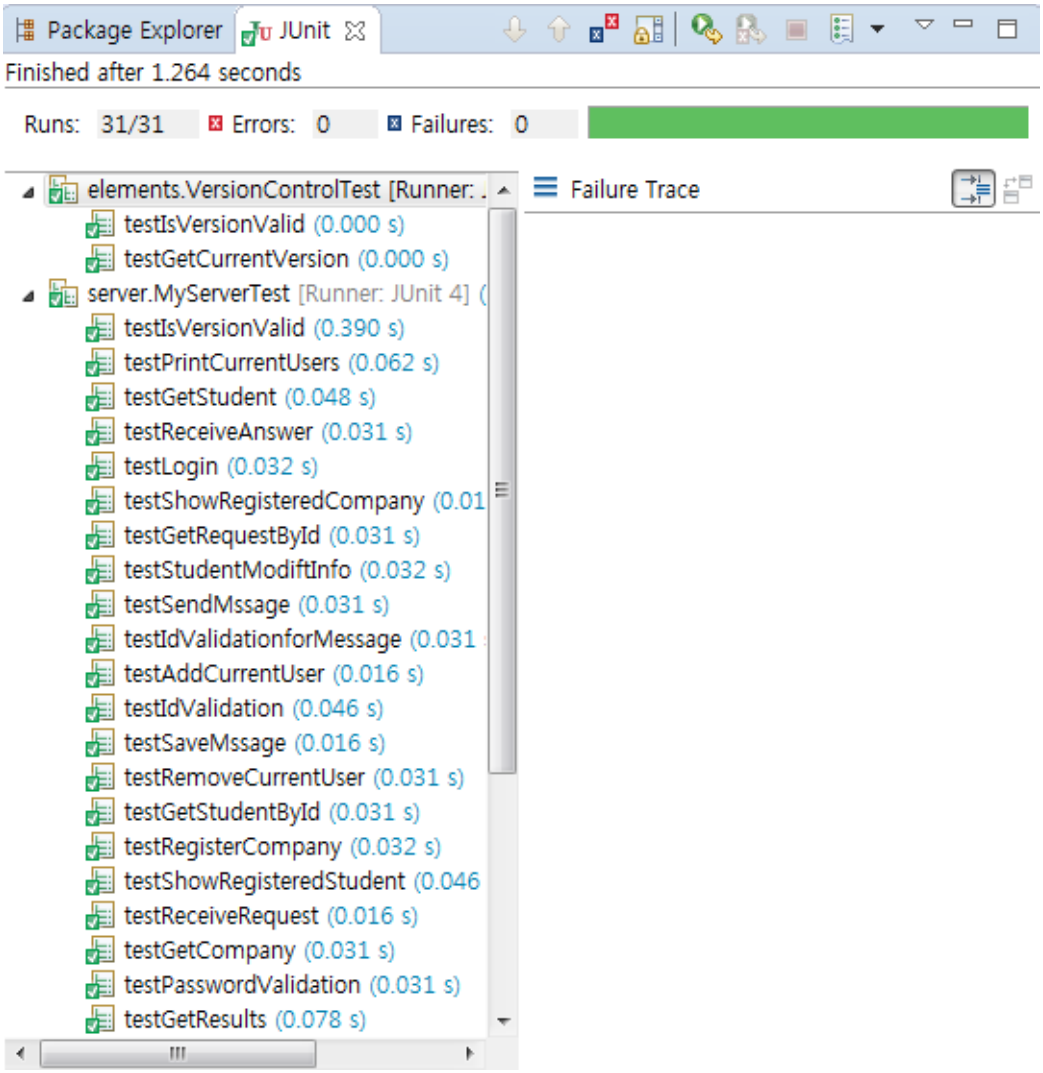
<그림 5-2-c. testStudentModiftInfo>

```java
@Test
public final void testPasswordValidation() {
    MyServer server = getDummy();
    JSONObject message = new JSONObject();
    JSONObject response = null;

    try {
        message.put("userType", "company");
        message.put("userID", "test");
        message.put("password", "test");

        response = server.passwordValidation(message);
    } catch (JSONException e) {
        e.printStackTrace();
        fail();
    }

    assertNotNull(response);

    try {
        assertEquals(true, response.getBoolean("valid"));
    } catch (JSONException e) {
        e.printStackTrace();
        fail();
    }
}
```

<그림 5-2-d. testPasswordValidation>

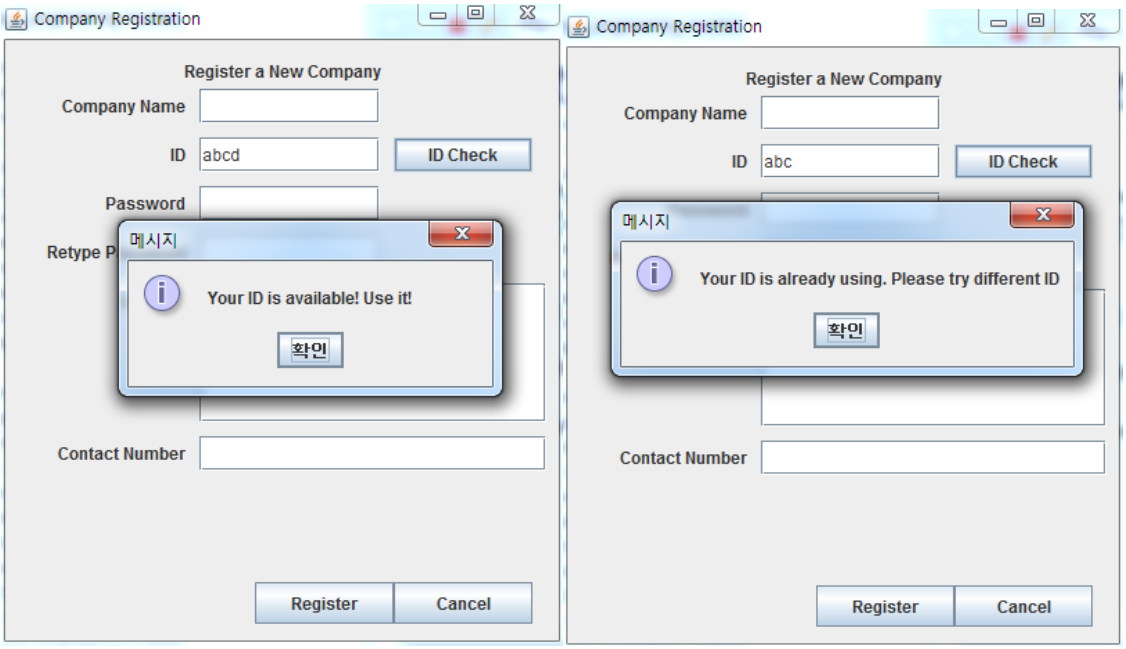
6. 테스트 케이스 검증

6-1. JUnit Run 결과

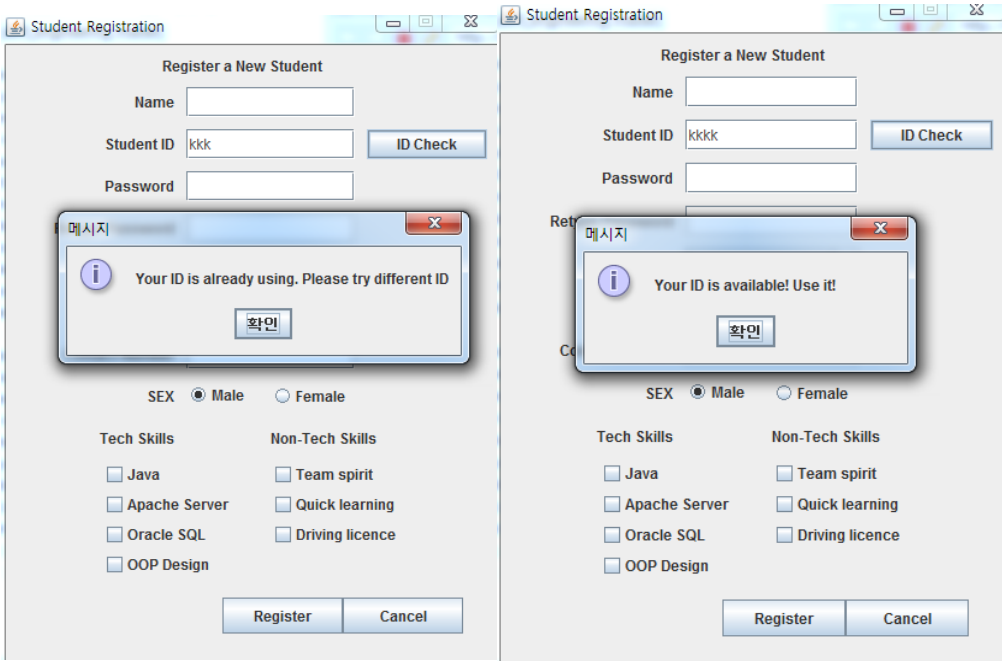


<그림 6-1-a. JUnit Run 결과>

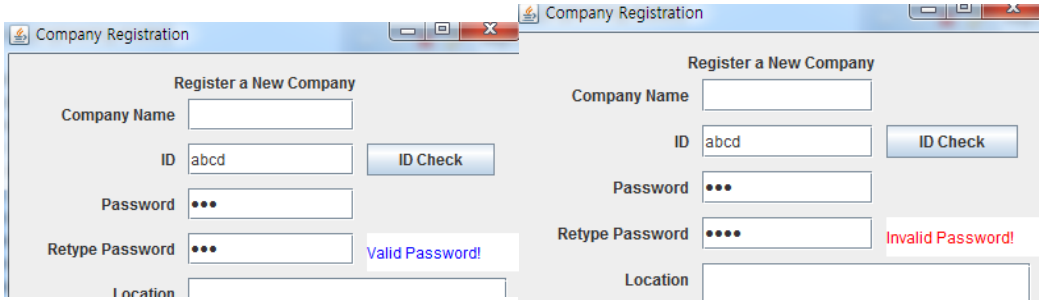
6-2. 실행결과 (Release Test)



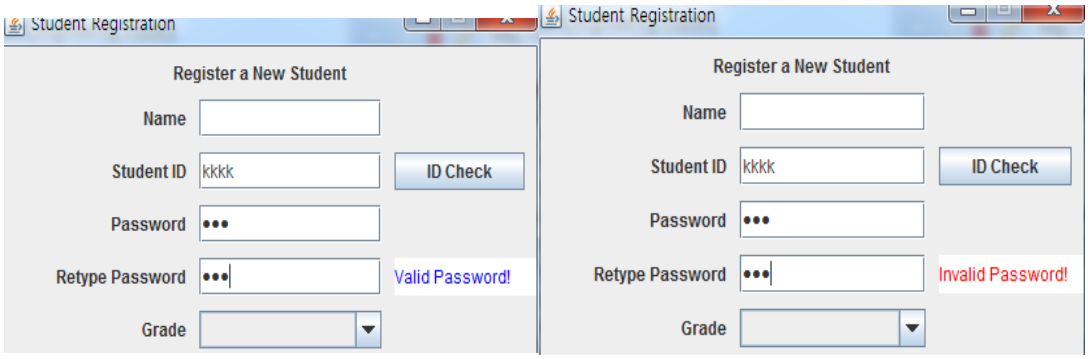
<그림 6-2-a. Company ID 중복 테스트 결과>



<그림 6-2-b. Student ID 중복 테스트 결과>



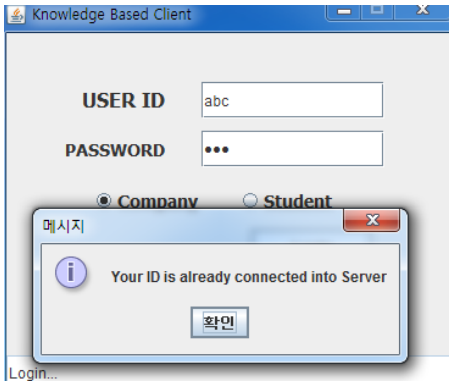
<그림 6-2-c. Company Password 확인 테스트 결과>



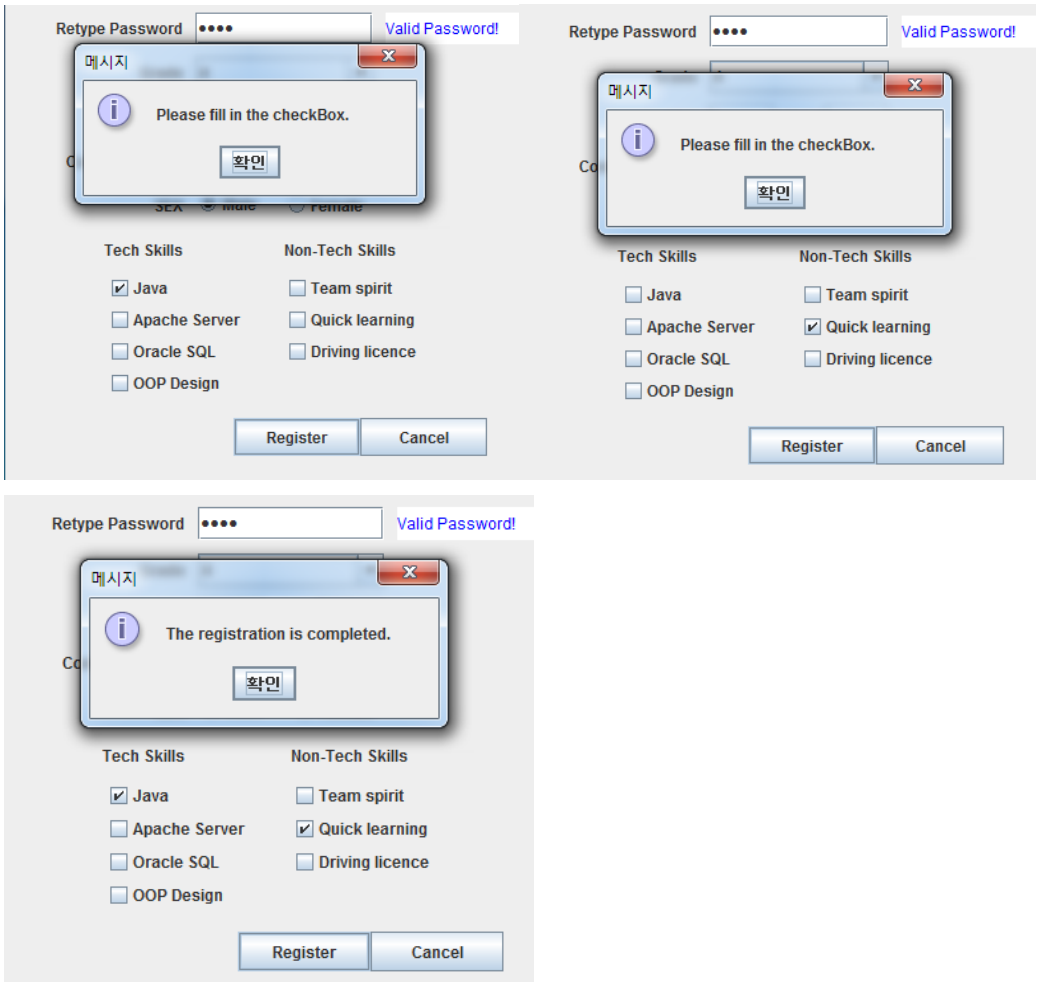
<그림 6-2-d. Student Password 확인 테스트 결과>

<그림 6-2-e. 로그인 실패 - No input>

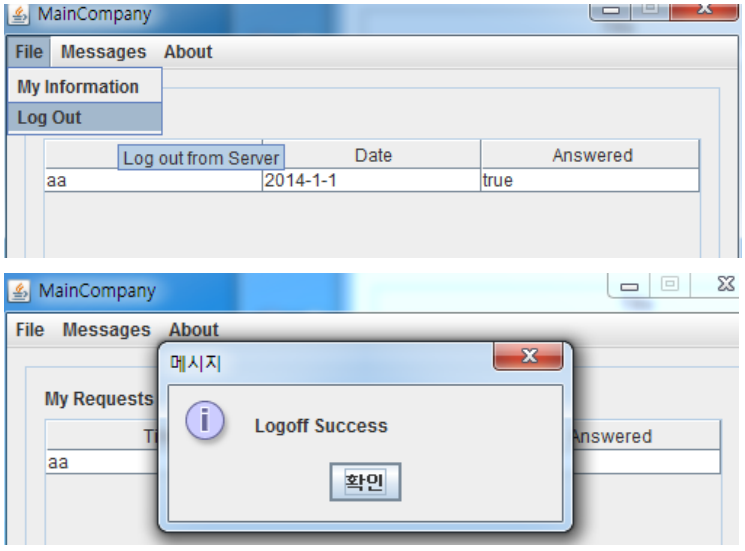
<그림 6-2-f. 로그인 실패 - Wrong Password>



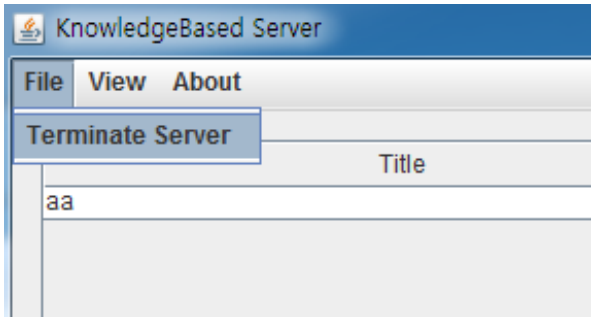
<그림 6-2-g. 로그인 실패 - Duplicated Login>



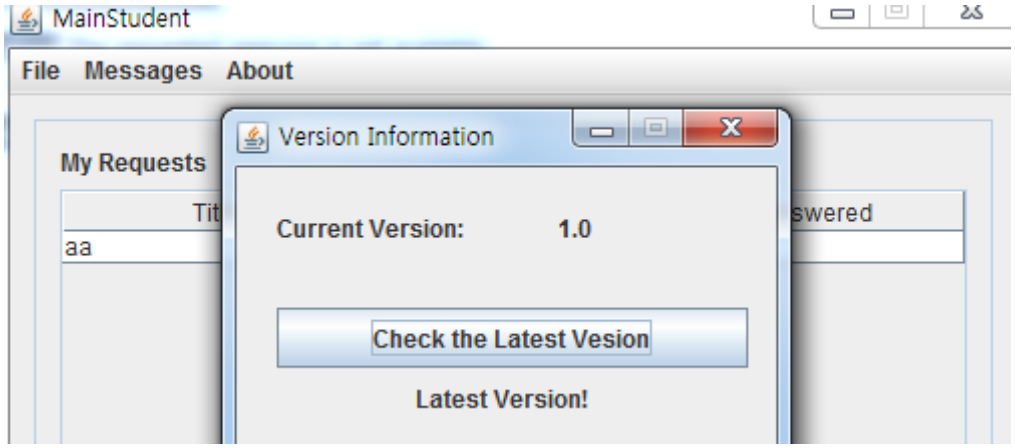
<그림 6-2-h. Tech & Non-Tech Skill 테스트 결과>



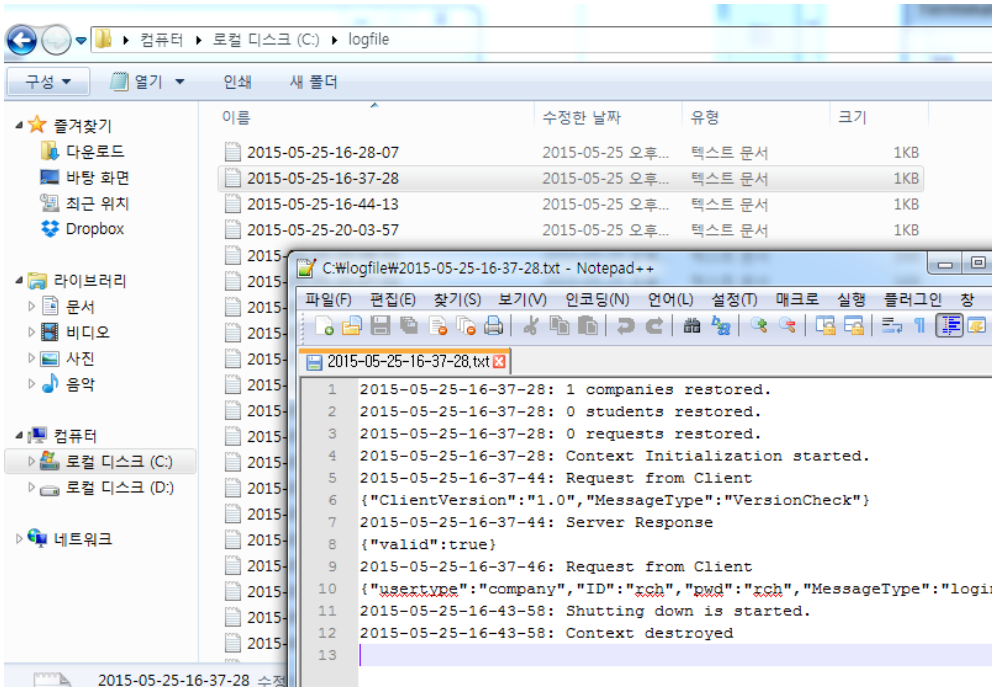
<그림 6-2-i. Client Termination 테스트 결과>



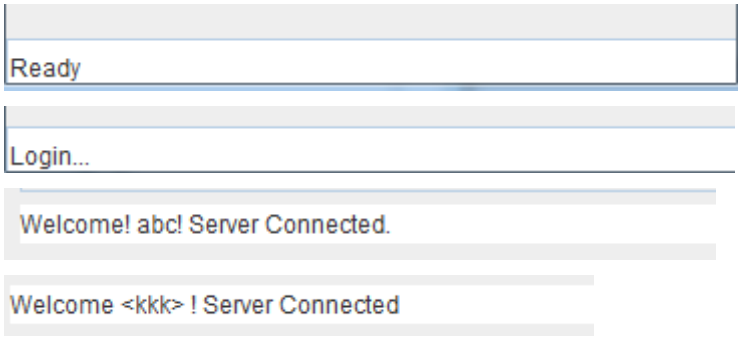
<그림 6-2-j. Server Termination 테스트 결과>



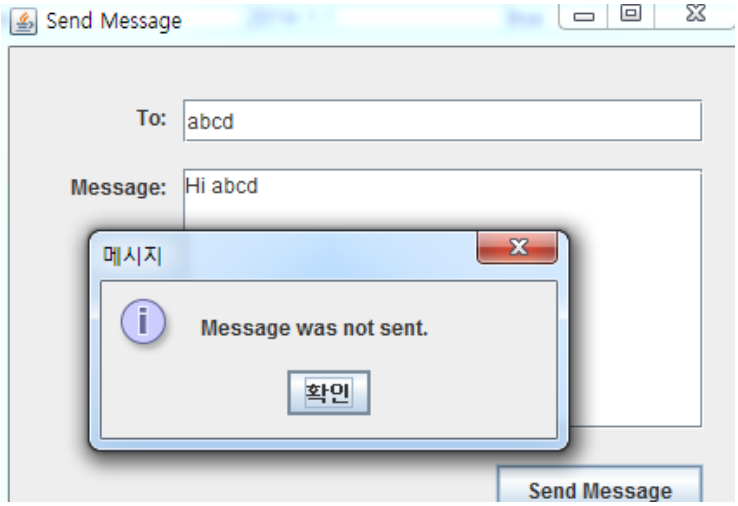
<그림 6-2-k. Version Control 테스트 결과>



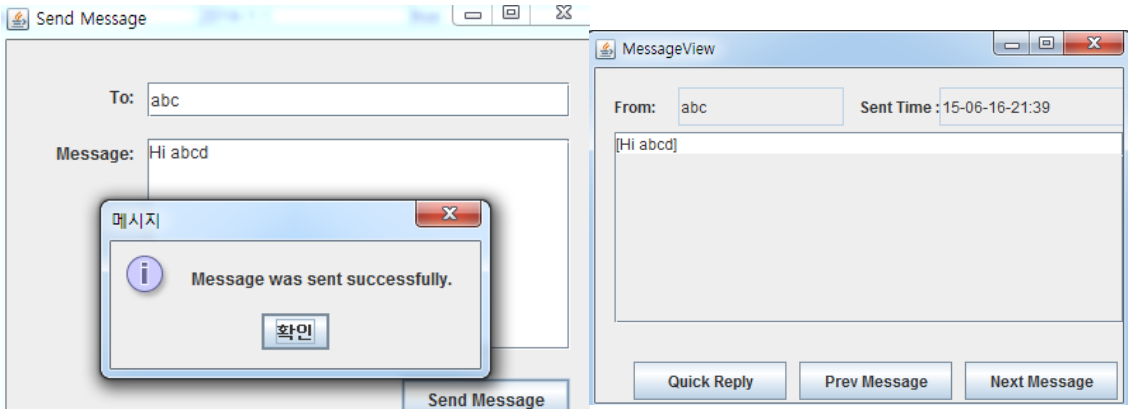
<그림 6-2-l. Log File 테스트 결과>



<그림 6-2-m. Progress Bar 테스트 결과>



<그림 6-2-n. Invalid ID로 메시지 전송 불가 테스트 결과>



<그림 6-2-o. Message 전송 테스트 결과>

7. JSON Format

case : "login"	case : "companyregister"
{“MessageType”, “login”} {“usertype”, Stringvalue} {“ID”, Stringvalue} {“pwd”, Stringvalue}	{“MessageType”, “companyregister”} {“CompanyName”, Stringvalue} {“ID”, Stringvalue} {“password”, Stringvalue} {“Location”, Stringvalue} {“Contactnumber”, Stringvalue}

case : "Studentregister"	case : "newrequest"
{“"MessageType"”, “"studentregister"”} {“"StudentName"”, Stringvalue} {“"ID"”, Stringvalue} {“"Password"”, Stringvalue} {“"Grade"”, intvalue} {“"Age"”, intvalue} {“"Gpa"”, Doublevalue} {“"Contact number"”, Stringvalue} {“"Sex"”, Stringvalue} {“"TechSkills"”, JSONArray} {“"NonTechSkills"”, JSONArray}	{“"MessageType"”, “"newrequest"”} {“"ID"”, Stringvalue} {“"Position"”, Stringvalue} {“"StartDate"”, Stringvalue} {“"EndDate"”, Stringvalue} {“"Payment"”, Stringvalue} {“"NumberOfStudents"”, intvalue} {“"DueDate"”, Stringvalue} {“"Grade"”, intvalue} {“"Description"”, Stringvalue} {“"TechSkills"”, JSONArray} {“"NonTechSkills"”, JSONArray} {“"error"”, boolean value}

case : "getresults"	case : "getrequests"
{“"MessageType"”, “"getresults"”} {“"ID"”, Stringvalue} {“"Title"”, Stringvalue} {“"Date"”, Stringvalue} {“"Complete"”, boolean value} {“"Students"”, JSONObject}	{“"MessageType"”, “"getrequests"”} {“"ID"”, Stringvalue} {“Requests”, JSONObject}

case : "answer"	case : "studentidValidation"
{“"MessageType"”, “"answer"”} {“"ID"”, Stringvalue} {“"RequestID"”, intvalue} {“"Answer"”, Stringvalue}	{“MessageType”, “studentidValidation”} {“ID”, Stringvalue}

case : "VersionCheck"	case : "LogOff"
{""MessageType"", ""answer""} {""ID"", Stringvalue} {""RequestID"", intvalue} {""Answer"", Stringvalue}	{""valid"", boolean value}

case : "login_valid"	case : "others"
{""MessageType"", ""login_valid""} {""result"", intvalue}	{""valid"", boolean value}

8. 품질 향상도 평가

기존의 프로젝트를 유지보수하고 새로운 기능을 추가 하는 것은 막대한 비용이 들어간다. 그러므로 기존의 프로젝트에서 재사용 할 수 있는 부분과 새롭게 구성해야 되는 부분을 잘 파악하여 최대한 적은 비용으로 효율적으로 프로젝트를 수정하는 것이 중요하다.

이 프로젝트는 궁극적으로 학생들이 인턴쉽을 좀 더 편리하게 지원하고 결과를 알아보기 위해 제작한 시스템이다. 이번학기 교내 현장학습 지원 시스템을 체험해봄으로써 해당 프로젝트에 어떠한 기능이 추가되면 사용성이 증가될지 생각 할 수 있는 좋은 계기였다.

소프트웨어의 품질을 평가하는데 있어 여러 가지 요소가 있다.
그 중 이번 프로젝트는 **Reliability**와 **Availability** 그리고 **Usability**를 높이는데 초점을 맞추어서 진행되었다. 해당 프로그램이 사람의 재산을 다루거나 생명에 크리티컬한 시스템이 아니기 때문에 위의 사항을 중점으로 진행 되었다.

우선 Usability를 높이기 위해서 GUI를 수정하였다. 사용자들이 가장 크게 체감 할 수 있는 부분이 바로 사용하면서 마주하는 GUI부분이다. 기존의 프로그램은 프로그램 개발자의 입장에서 버튼들이 배치되어 처음 사용하는 사용자가 제대로 각 기능들을 알아보기 어려웠다. 또, 여러 가지 인풋 케이스에 대한 핸들링이 부족하여, 부주의한 인풋의 결과로 프로그램이 빈번히 강제종료 되는 문제점이 있었다. 이에 각 입력필드마다 인풋 값을 제한하고 잘못된 값이 입력되었을 시에 강제종료 되지 않고 사용자에게 알림을 줌으로써 사용성을 증대시켰다.

또 서버는 언제든지 접속하는 사용자에게 서비스를 제공할 수 있어야 하므로 Reliability를 높일 필요가 있다고 판단하였다. 이를 보완하기 위한 기능으로 올바른 종료를 유도하기 위해 종료버튼을 추가하였고, 사용자가 현재 접속 및 진행 상태를 판단하기 위한 프로그레스 바도 추가하였다. 그리고 서버가 수행하는 작업에 대하여 Log File을 남김으로써 차후 문제가 발생 시에 어디에서 문제가 발생하는지 확인하고 보완할 수 있는 시스템을 추구하였다.

기능적으로 가장 큰 추가사항은 메시지 전송 시스템이다. 교내 현장실습 지원 시스템을 비롯하여 이전 프로젝트에서는 학생이 해당 회사에 궁금한 점이 있어도 질문을 하기가 굉장히 까다로운 시스템이었다. 하지만 직접 메시지를 보낼 수 있는 기능을 추가함으로써 애매하거나 추가로 필요한 부분에 대해 질문을 하고 빠른 시간 내에 답변을 받을 수 있는 시스템을 만들었다.

다음에 이 프로젝트를 유지보수하게 될 팀을 위하여, 새로 추가한 기능에 대해서 뿐만 아니라 기존의 프로젝트의 컴포넌트에 대해서도 최대한 많은 Test Case를 작성하였으며, 기존에 GUI와 제어 부분의 코드가 하나의 클래스에 엉켜있던 부분을 분리하는데 많은 공을 들였다.

이와 같이 미흡하지만 처음 Incremental Development를 진행하면서 프로젝트의 중요 요소들이 어떠한 것들이 있는지 이해하고 개선시키기 위해서 최대한 노력하였다.

9. 프로젝트 일정 및 업무량 할당

9-1. 프로젝트 일정

<그림 9-a. 프로젝트 일정>

9-2. 업무량 할당

이 프로젝트는 2인의 팀원으로서 진행되었다. 소수의 인원이 빠른 진행일정을 가지고 개발을 진행 하였다. 2인으로서 각자 Server/Client 구분 없이 하나의 기능 단위로 Implement 하였고 상대방이 요구사항을 충분히 만족하는지 테스터 역할을 담당하였다.

예를 들어 한 명이 중복로그인 방지 기능에 대하여 예상 Test Case를 작성하고, 기능 구현을 한다면, 나머지 한명이 예상 Test Case를 이용하여 검증하고 기존에 의도했던 요구사항에 부합하는지를 판단하는 과정으로 개발을 진행하였다.

한 주에 한번 씩 프로젝트 결과물을 전체적으로 검토하고, 기한 이내에 개발이 가능할지, 요구사항을 충분히 만족하도록 개발이 진행되었는지에 대하여 회의를 하였다.

10. 최종 결론 (프로젝트를 진행하며 배운 점, 느낀 점)

이번 프로젝트는 체계적인 프로젝트 개발 방법론을 적용하여 개발을 실시한 첫 사례이다. 그동안 오로지 기능적인 부분만 생각하여 주먹구구식으로 일정을 계획하고 개발을 진행했던 반면에 여러 방면으로 검토를 하고 어떤 절차로 프로젝트를 계획하고 진행하는지 배울 수 있었던 소중한 시간이었다.

특히 요구사항 도출과 설계가 프로젝트의 전체에서 아주 큰 비중을 차지하며 성공적인 프로젝트를 위해 필수적인 요소라는 걸 깨달을 수 있었다. 평소 개발자들이 프로그램을 만든 후에 보면 사용자가 요구하는 기능이 아니거나, 충분한 기능을 제공하지 못하는 사례를 종종 들을 수가 있다. 이는 소프트웨어 공학적 측면으로 충분히 검토하지 않고 개발했기 때문이다.

요구사항을 변경할 때에 드는 비용은 초기 요구사항을 도출할 때의 비용의 수십 배가 소비되기 때문에 무엇보다 올바른 요구사항을 도출하는 것이 가장 중요한 첫 걸음이다.
이 프로젝트에서는 실제로 학생입장에서 사용자의 측면을 상상해보고 필요한 요구기능을 비교적 쉽게 떠올릴 수 있었기 때문에 꼭 필요한 요구사항을 도출하는데 수월하였다.

하지만 기존의 프로젝트를 수정하는 작업이었기 때문에 내 의도와 다르게 이미 많은 부분이 구현되고 결합되어있어 수정하기 곤란한 부분이 많았다. 또 수업시간에 유지보수 팀이 다를 시에 서로 입장차이가 발생한다고 한 부분을 실감할 수 있었다. 2명이서 짧은 시간에 개발을 해야 하는데 프로젝트를 파악하기 위한 문서도 빈약했고 테스트 코드도 많이 부족했다. 만약 회사에서 사용되는 커다란 프로그램이 이런 식으로 개발되어지고 나에게 유지보수 업무가 맡겨진다고 생각해보면 엄청 괴로울 것 같다.

이번 학기 소프트웨어공학 수업을 통해서 여러 가지 다양한 규모와 목적의 프로젝트에 맞는 개발방법론이 제시되어있으며, 특히 규모가 크거나 크리티컬한 시스템을 개발할 시에 이러한 개발방법론을 적용하는 것이 더욱 더 중요시 된다는 걸 배울 수 있었다. 앞으로 남은 학부 프로젝트에서도 소프트웨어공학적인 접근을 강조하여 결과물 뿐만 아니라 개발 과정 자체도 의미 있고 시행착오를 줄일 수 있는 방법으로 개발을 진행할 수 있을 것 같다.