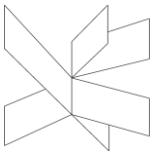
Online Gradebook

Project Report

Mircea Ionut Dobre - 293117

Pavel Balan - 293129

Sandut Chilat - 293086



Supervisors:

Ole Ildsgaard Hougaard

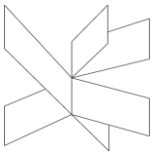
Jakob Knop Rasmussen

VIA University College

29425 characters

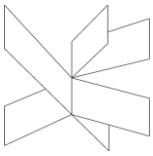
Software Technology Engineering

3rd semester



December 2020

Table of content



1. Abstract

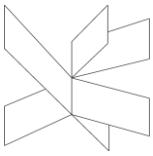
*The **Online Gradebook** represents an online school grading system that improves upon the currently existing traditional physical ones. It allows for the registration of students and teachers by the school secretary, which is pre-registered into the system.*

A student can view their grades and absences at their respective courses, as well as display their account information (ID, name, address, phone number). A teacher can view all of the afore-mentioned information of their students from the classes that they teach. Additionally, a teacher can add grades/absences for that day and excuse the absences. The pre-existing secretary can create/delete/edit students, teachers, and classes.

Based on a customer's potential needs, 9 use-cases and 16 system requirements have been established. All of them have been successfully implemented in the software.

The software has been divided into 3 tiers - the first one written in C# and the other 2 in Java. The connection between tier 1 and tier 2 has been made through web-services, while the one between 2 and 3, through sockets. The GUI has been made using the Blazor UI framework. The relational database management system used is PostgreSQL, while the database itself is being stored locally.

After testing using the Black Box method, no unforeseen issues or exceptions have been found. The detailed results are contained in the "Test" section.



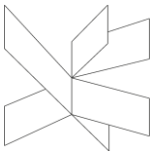
2. Introduction

A very popular solution for grading students in a lot of educational institutions is still the outdated practice of physical gradebooks. There, a teacher gives students grades and lists their absences in a catalog, writing them down by hand. This approach has several drawbacks (Network Support, 2019) with their origin in human error, unnecessary time consumption, and security issues. Writing things down by hand means that in order for the information to be distributed, it has to be re-written several times - from teacher to student or from teacher to administration. That always comes with the risk of human error, which is hard to rectify once written on paper, especially in regards to imperative information such as student grading. But perhaps most importantly, physical grading severely limits the students' access to the said information regarding their grades and attendance.

An alternative to the traditional grading practice is the online catalog system. Several higher educational institutions, especially universities and colleges, already use such grading solutions to more easily manage the hundreds or even thousands of students enrolled in a course. It gets rid of the superfluous and tedious task of having to re-write everything in order to distribute the information farther. It also takes care of the security issues because only the allowed personnel can make changes to the grades. And finally, it gives easy access to students and parents to view the respective information about their academic success online.

Online course-board programs like Blackboard (Blackboard, 2020) and Moodle (Moodle, 2020) can be extremely useful in the spread of data and the assortment of tasks in enormous courses like those offered as a major aspect of the core curriculum. In any case, the mechanized evaluating abilities of these projects are commonly limited to question banks with clearly defined right and wrong answers. More advanced computer-assisted grading systems have been developed for the assessment and grading of more subjective assignments such as essays (Sagrader, 2020), business case studies (Czaplewski A.J, 2009), and student software programs (Jones, E. L, 2001). Nonetheless, completely computerized frameworks are as yet restricted to applications with all around characterized rules and destinations.

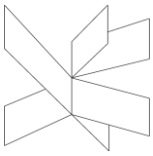
In conclusion, following the aforementioned statements, more and more educational institutions are transitioning to an online solution for grading, as the traditional means of cataloging activities through physical gradebooks prove to be time-consuming and inefficient.



3. Analysis

3.1. Functional requirements

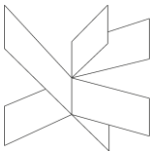
As a(n)...	I want too...	so that...
Parent	View the grades of my child	I can help him learn better when needed
Parent	View the absences of my child	I know when my child is skipping classes and take appropriate action
Student	View my grades	I can learn more on the courses that I have bad grades at
Student	View my absences	I can know which classes I cannot miss
Teacher	View the grades of my students	I am aware of their academic success
Teacher	View the absences of my students	I am aware of why and when they have been absent
Teacher	Assign grades to my students	Mark the grades that the student got from evaluations



Teacher	Assign absences to my students	Mark when a student missed a class
Teacher	Motivate absences	Mark that a student didn't miss a class but he was only late to it
Secretary	Change the secretary account password and username	Change the password and username in case of a security worry
Secretary	Create/edit/delete student accounts	Manage students
Secretary	Create/edit/delete teacher accounts	Manage teacher accounts
Secretary	Create/edit/delete classes	Manage classes
Secretary	Assign and remove students to classes	I can modify the class accordingly
Secretary	Assign and remove teachers to classes	I can modify the class accordingly
Secretary	Assign and remove courses to classes	I can modify the class accordingly

3.2. Non-functional requirements

- The system must respond in a timely manner
- The system must use the 3-tier architecture
- The system must use socket communication



- There can't be unhandled exceptions from incorrect user input
- The system has to be reliable
- The system has to not require any programming experience
- The system GUI must be easy to understand and use

3.3. Use case diagram

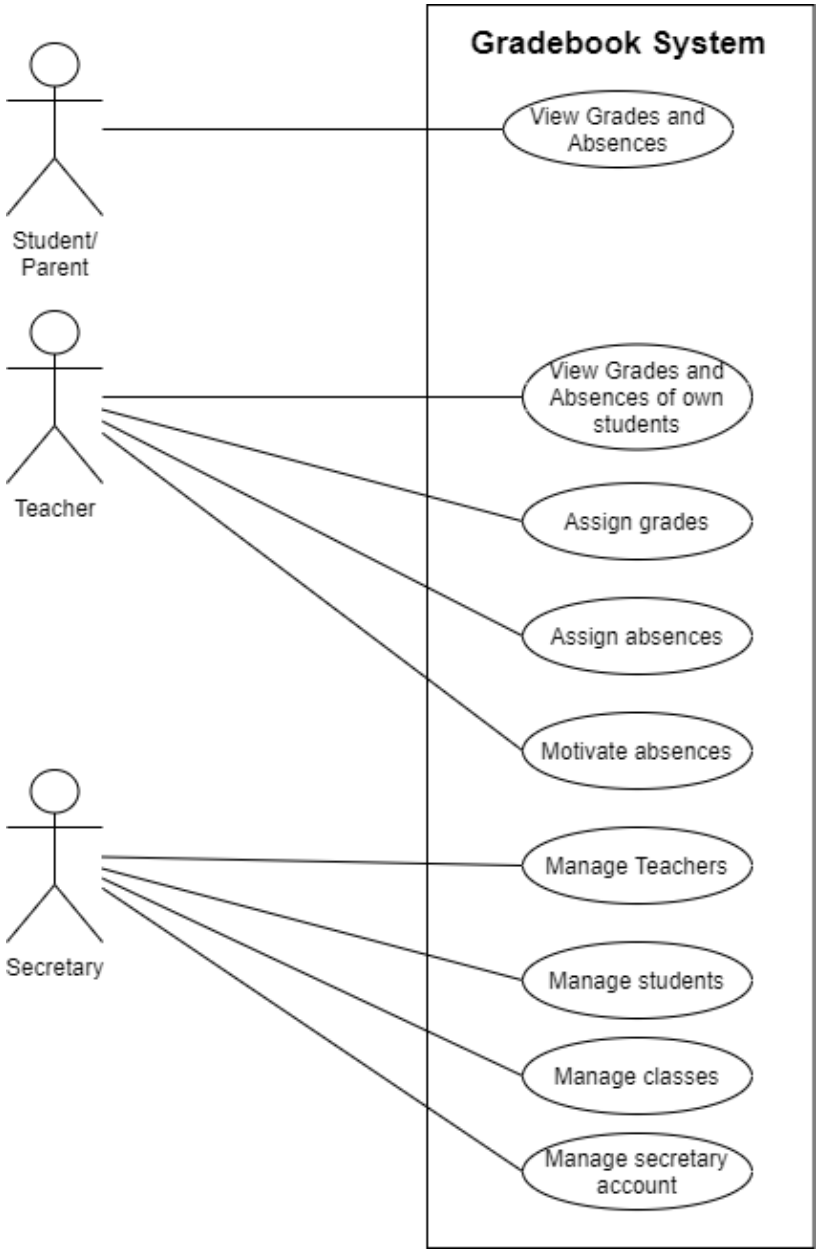


Figure 1 - Use Case Diagram

3.4. Use case descriptions and corresponding activity diagrams

Use case	**Assign grades**
Summary	Teachers can assign an individual grade for every singular student
Actors	Teacher
Precondition	In order to assign a grade to a student by the teacher, a teacher account should be created by a secretary In order for a student to be able to receive a grade, a student account should be created by a secretary
Postcondition	A student will be graded by the teachers for a specific course with a individual grade
Base sequence	1.In the main menu select the ID field or password field for filling out the necessary information. 2.Type your ID number. 3.Type your password. 4.Press the “Login” button. 5.Teacher’s name and his ID will be displayed. 6.Select a specific class which will be shown. 7.Select a specific student whose information will be presented. 8.In the panel “New Grade” from the “Course” drop-down menu select the required course. 9.In the same panel from the “Grade” drop-down menu select the individual grade. 10.Press the “Assign Grade” button. 11.In the pop-up window press “ok” if you are sure to assign this grade or press “cancel” and go to step 8 or 9.
Exception sequence	2a. If the ID is invalid go back to step 2. 3a. If you inserted a wrong password in the “Password” field go back to step 3.
Note	The ID will be automatically generated. Password can be either an only number password , only letters password or a combination between letters and numbers and it shouldn't be longer than 8 characters.

	There is a limited number of grades automatically given for teachers' choice (from 1 - 10). There is a limited list of courses automatically given for teachers' choice.

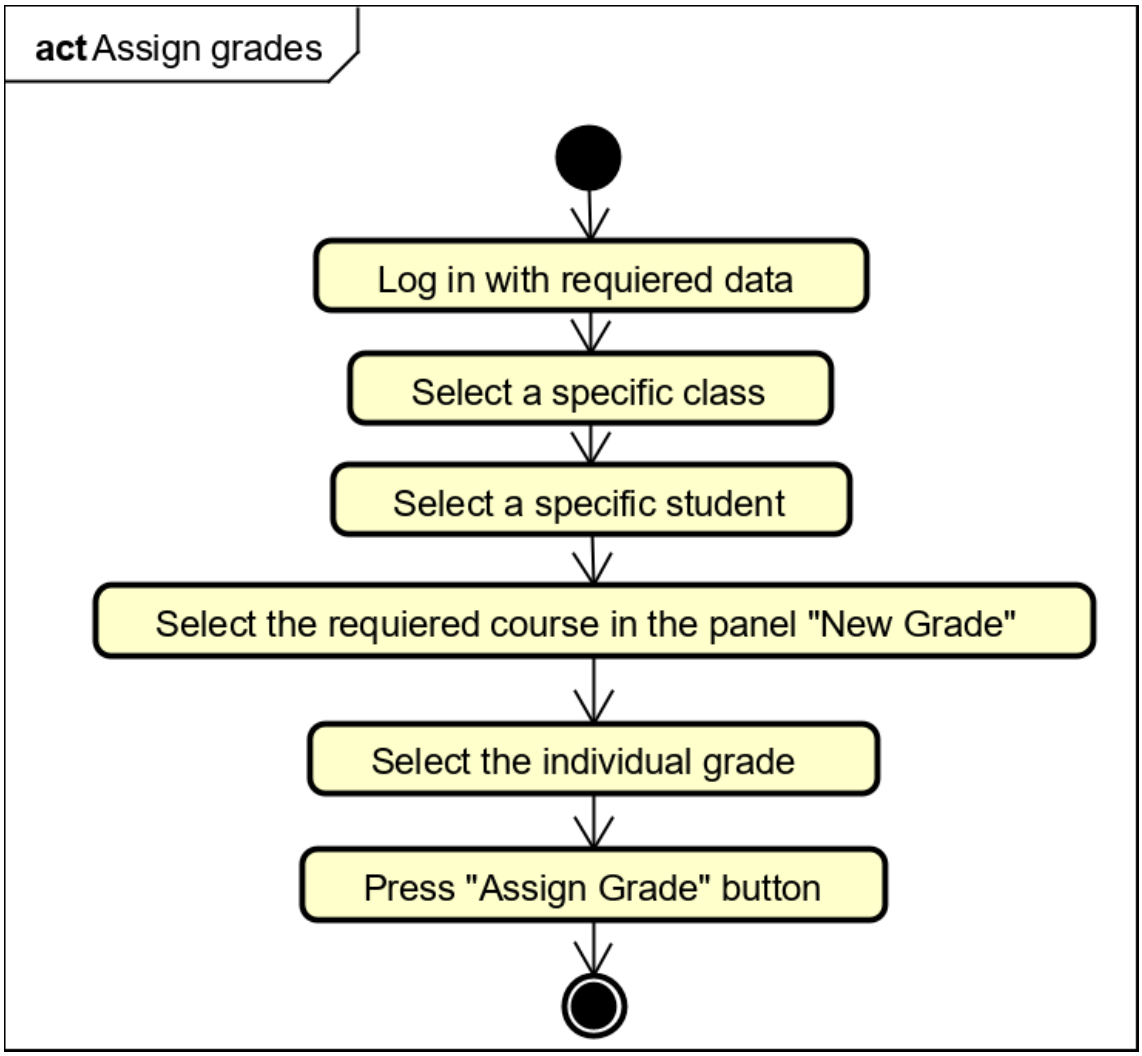


Figure 2 - "Assign grades" Activity Diagram

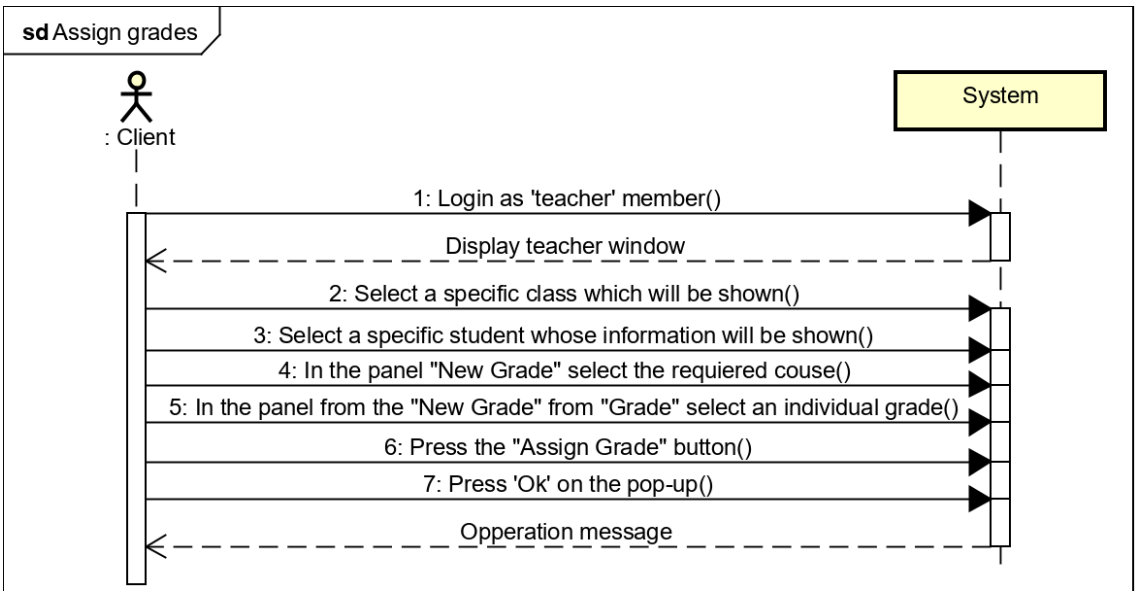


Figure 3 - "Assign grades" SSD

3.5. Domain model

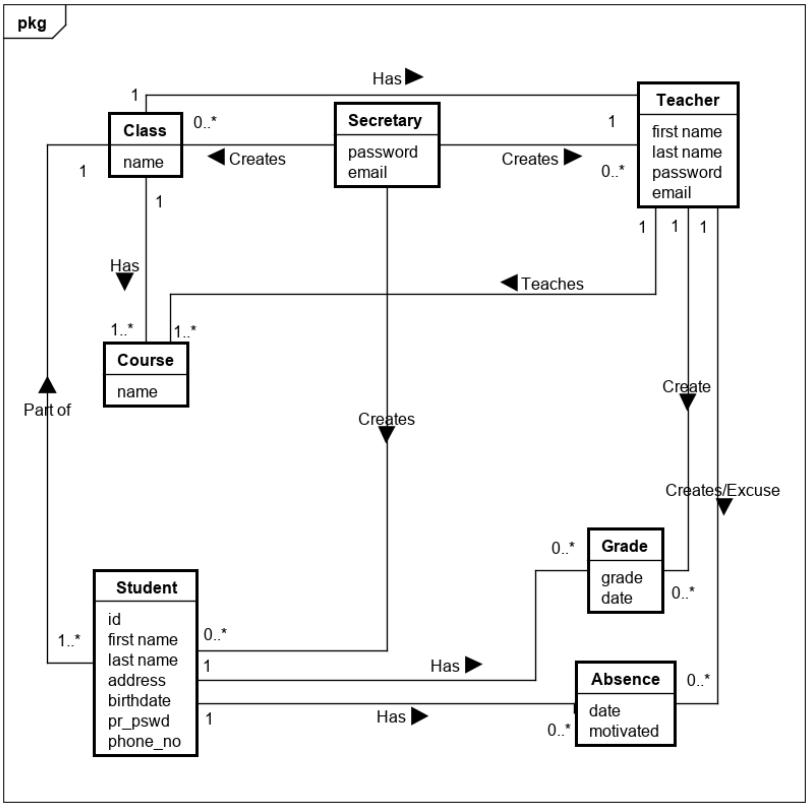
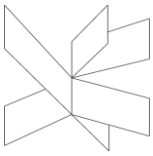


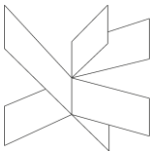
Figure 4 - Domain Model



3.6. **Security**

Security is one of the most important aspects of a system because it is connected to personal data and can affect all of the users using the system as well as the people administering the system. In the current case, the system we made offers an online gradebook for a more modern way of keeping grades and absences of school students. A lot of security issues had to be addressed so that that system is secure. Authorization, authentication and input validation are just a couple of the many responses our team had thought of. Some of those ideas have also made it into the final products but unfortunately some of them remained in the design stage.

- The most vulnerable part of the system to security attacks is the database tier. For example, if an unauthorized individual gets access to the database tier he would have access to all of the user data and could have malicious intent. In order to counteract this threat, the information stored inside the database has to be hashed and encrypted. For example the password can be stored as a hash.
- The database does also need continuous input validation to prevent damage to the user data and even database tier crashes. Also SQL commands can be inserted in text boxes and can be used to cause harm to the database or extract data via SQL Injection attack
- Another security problem is receiving TCP Socket connection requests from an application other than our own trying to connect to our database with malicious intent if he gets a hold of the IP address and port number. This issue can be resolved by Authentication Tokens
- In terms of tier 2 also known as the middleware, having a web service constantly listening for potential calls, proper authentication needs to be introduced to remove the risk of a hacker gaining access to the web service between tier 1 and tier 2 and be able to do what he wants.
- The front-end of the system is where most of the security takes place also being the place where the client interacts with the system. A login system makes sure that unauthorized people don’t have access to the system.



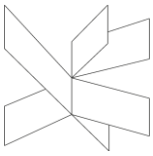
Risk number	Risk description	Likelihood(1-5)	Consequence(1-5)
1	Brute force log-in	5	5
2	The attacked used SQL Injection to access the DBS	5	5
3	The attacked gets access to a secretary account	2	5
4	The attacker use man in the middle attacks	5	5

- Risk prevention 1. In case the attacker tries to bruteforce a user account info, the account goes into lockdown after 10 password tries in the same day, thus partially preventing the attack.

- Risk prevention 2. In case the attacker tries SQL Injection, Tier 2 uses ‘if’ statements to check all input from the user and stops this attack from reaching the database tier.

- Risk prevention 3. In case the attacker gains access to a high security level account info, a 2 part authentication process can be implemented like NemID or SMS code authentication.

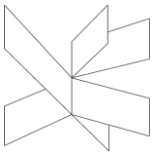
- Risk prevention 4. In case the attacker is doing a man in the middle attack, the TCP socket traffic and UserService data can be encrypted and validation tokens can be used in case of replay attacks.



4. Design

4.1. System architecture

The software is divided into 3 tiers, following the 3-tier architecture. The first one is written in C# using Blazor, and makes up the client-side, while the other two are written in Java and make up the server-side. The connection between the first and the second tier is made through web services, but the one between the second and the third is made using sockets. The information sent between the first 2 tiers is encapsulated into GET/POST requests, while between the second and the third - through network packages. The web service communication mostly but not fully follows the REST principles (REST API) due to the software architectural choices. The third tier communicates with the database by sending over SQL requests as strings. The development of the software has been attempted to be in-line with the SOLID design principles (Oloruntoba, 2015), as to make it more understandable, flexible and maintainable. The following packages containing their respective classes, are explained in detail below.



4.2. Security

Security is an important topic this semestester and the database is the most important part of the system when it comes to security. Knowing that, the team had to discuss the possible security breacher and here are some of the topics that were addressed as a result:

- Tier 2 and Tier 3 TCP Socket communication needs to be encrypted in order to avoid man in the middle attacks.
- Data needs to be checked before being put into the database to avoid potential program crashes from invalid data makings its way to the database
- The database needs to be encrypted in order to avoid personal data leakage
- Requests needs to be authenticated in order for the system to be sure that the request are legitimate and trustworthy

Unfortunately, the team was unable to implement all of the security problems stated above but they were thoroughly investigated and the following solutions were designed:

- Authentication methods can be used for requests
- Password needs to be hashed before put inside the database by tier 1, never passing as a normal text field within the system traffic
- Careful validation in tier one using **if** statements in tier 2 so that no invalid data can reach tier 3
- Communication between the tiers has to be encrypted

4.3. Tier 1 - Client (C#)

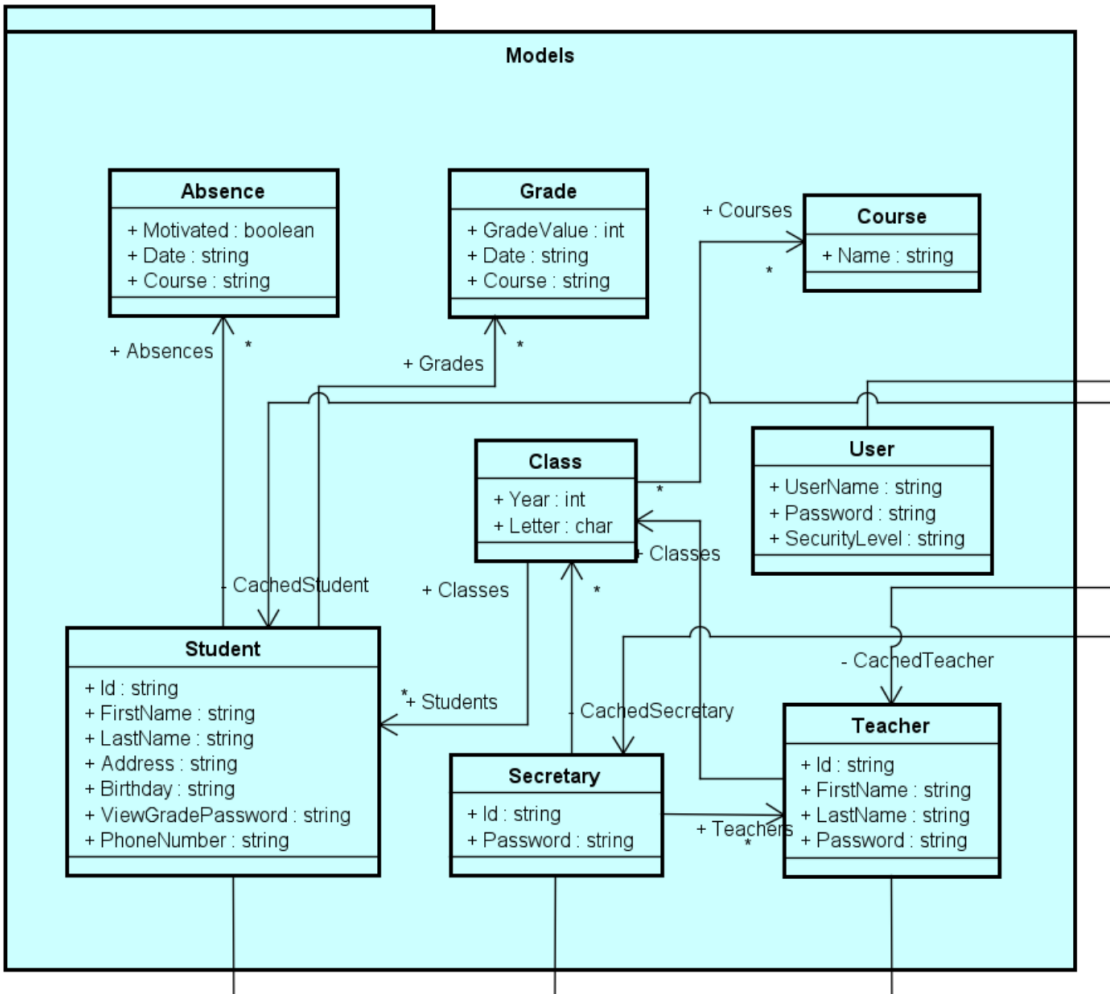


Figure 5 - Models folder class diagram

The diagram above represents all the objects from the `Models` folder which are being used in `Tier-1`: `Student`, `Teacher`, `Secretary`, `Class`, `Course`, `Grade`, `Absence`, `User`.

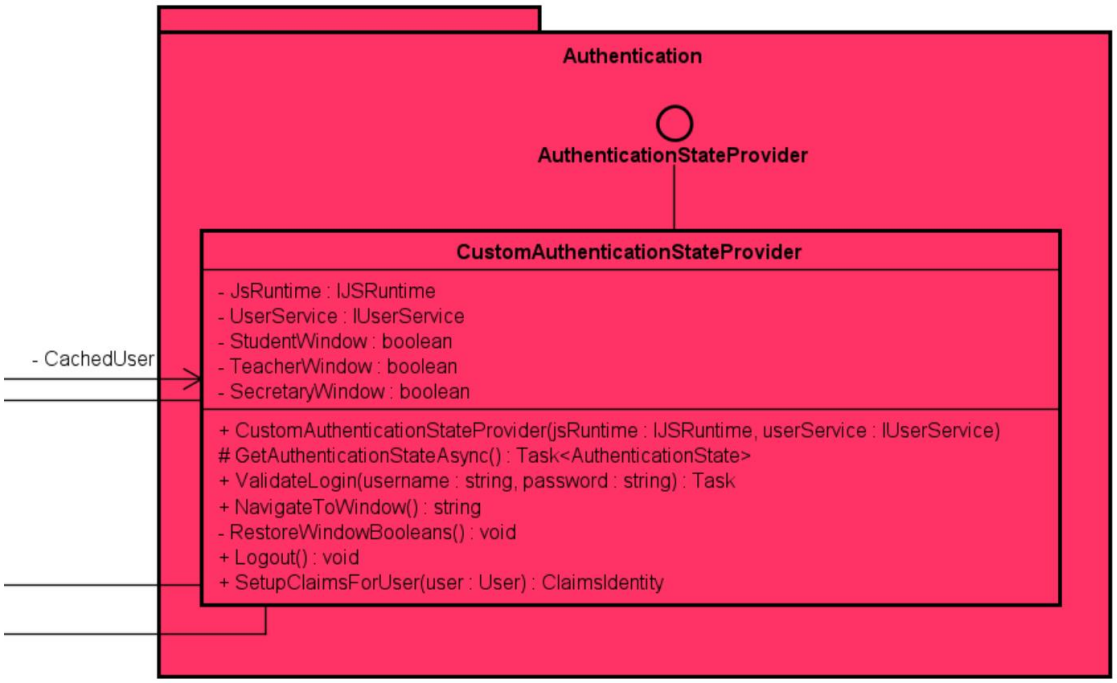


Figure 6 - Authentication folder class diagram

The `CustomAuthenticationStateProvider` class implements the `AuthenticationStateProvider` interface. It receives the cached `User` object and sets the respective claims, performs the login validation and the window navigation following the login.

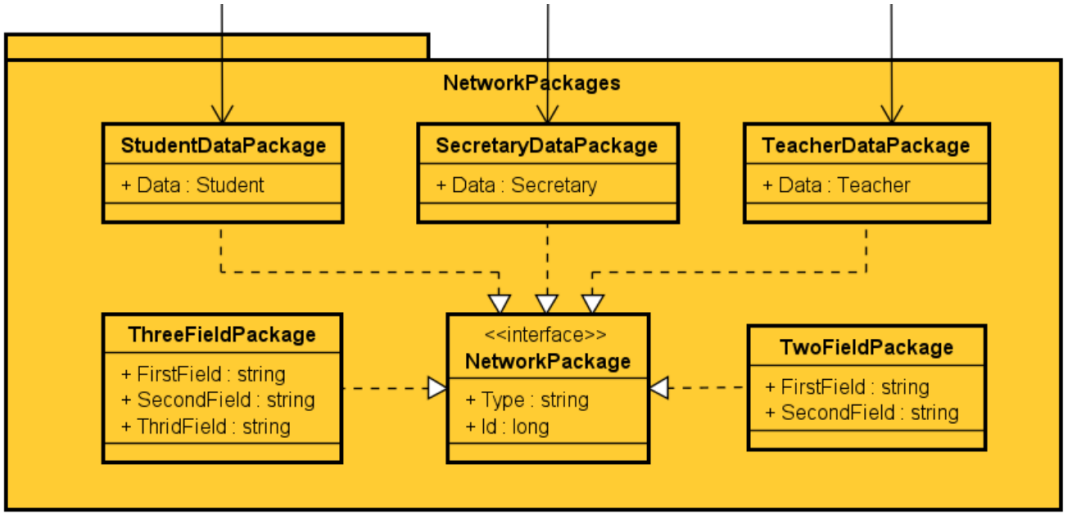


Figure 7 - Models folder class diagram

The data coming from `Tier 2` is encapsulated in the Network Packages. All of them implement the `NetworkPackage` interface, but hold different data (`Student`, `Teacher`, `Secretary`, 2 or 3 `string` values)

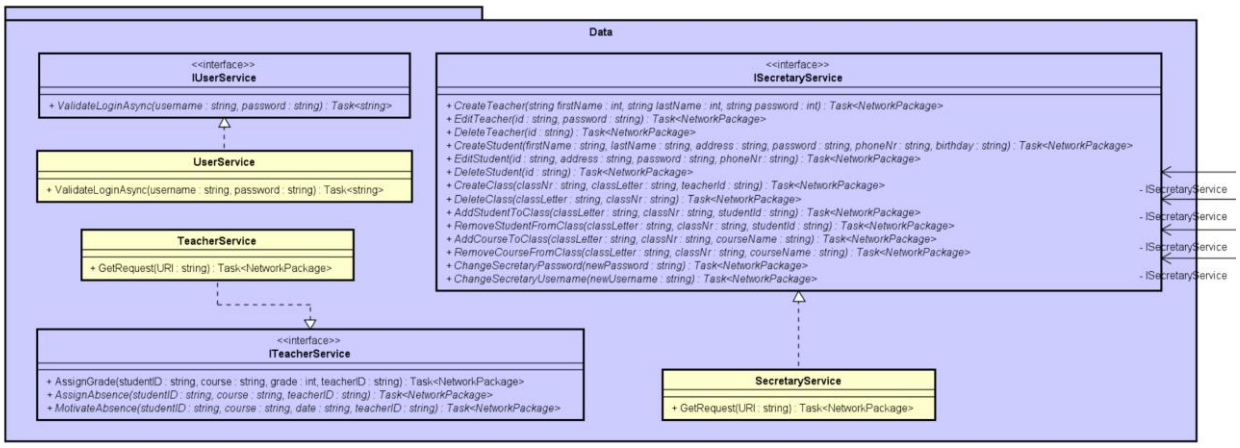


Figure 8 - Data folder class diagram

The `Data` folder contains all the service-classes and their implementations. They are used to communicate with tier-2 through web-services by sending mostly `GET` requests, except for the login request, which is of type `POST`. Each class is responsible for the requests of their respective object.

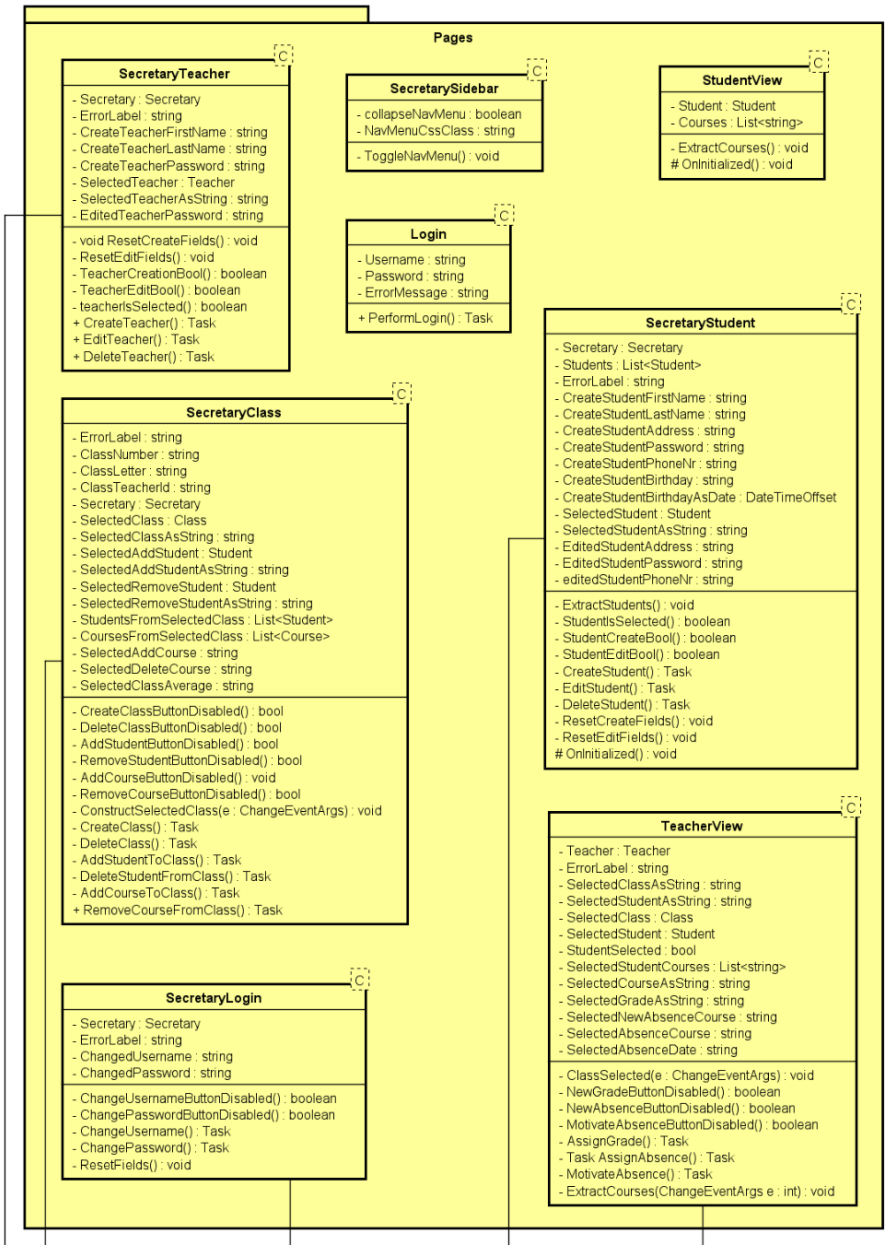


Figure 9 - Pages folder class diagram

The `Pages` folder contains all the `.razor` components used to display the information on `Tier 1`. Components have been used instead of actual `.cshtml` pages due to the software design needs.

4.4. Tier 2 - Server 1 (Java)

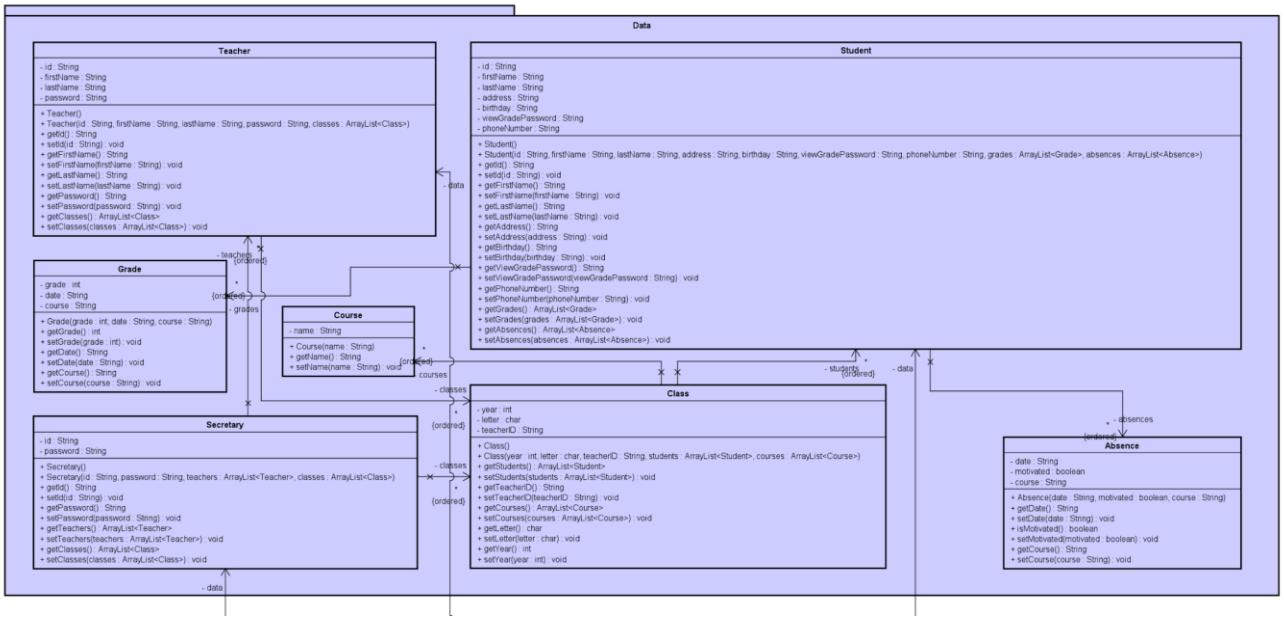


Figure 10 - Data folder class diagram

The `Data` folder holds all the object classes similarly to the `Models` folder from Tier1.

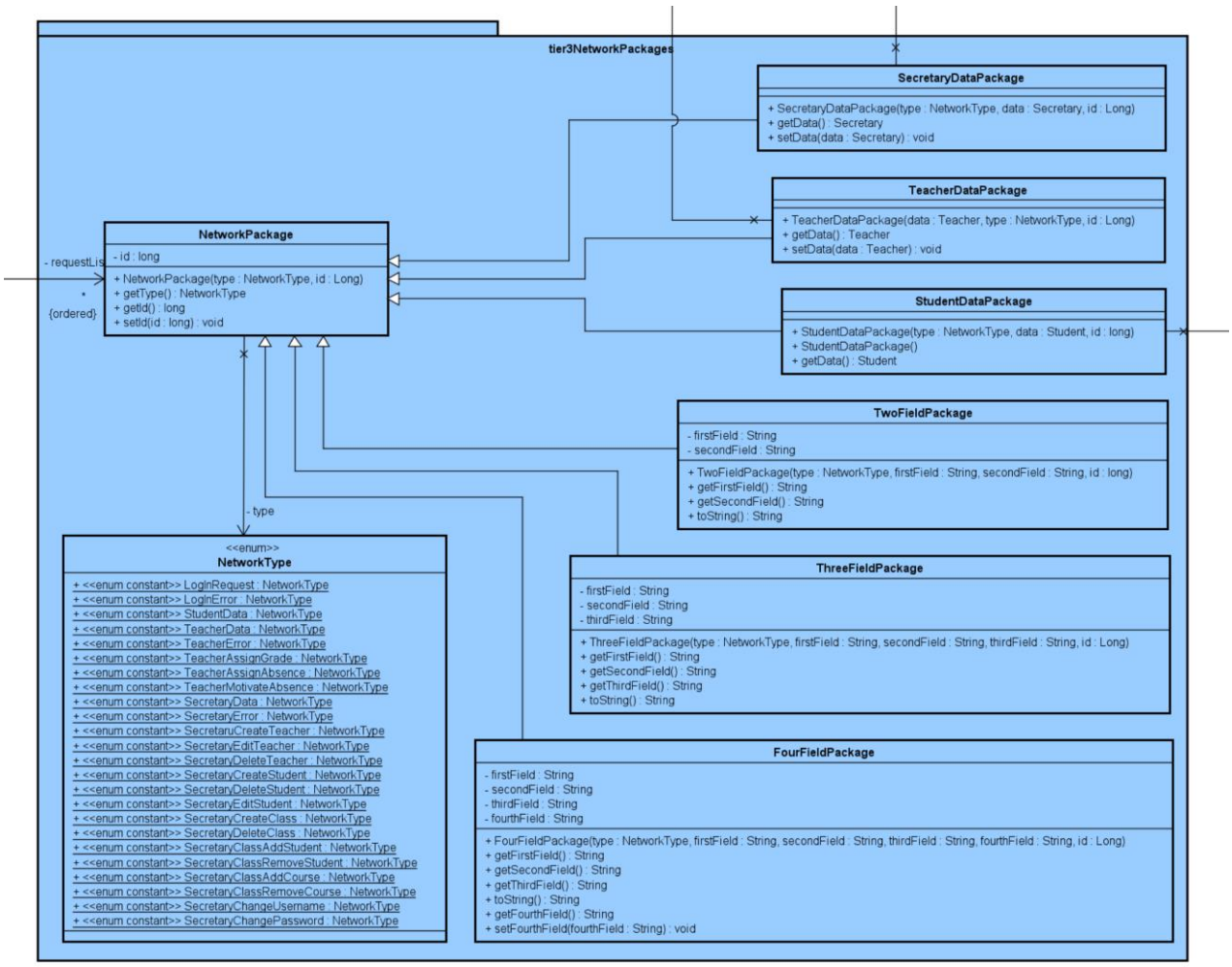


Figure 11 - tier3NetworkPackages folder class diagram

The `tier3NetworkPackages` folder, similarly to the `NetworkPackages` folder from Tier 1, holds the classes of the Network Packages. Additionally, it contains an enum `NetworkType` that defines the possible types of network packages.

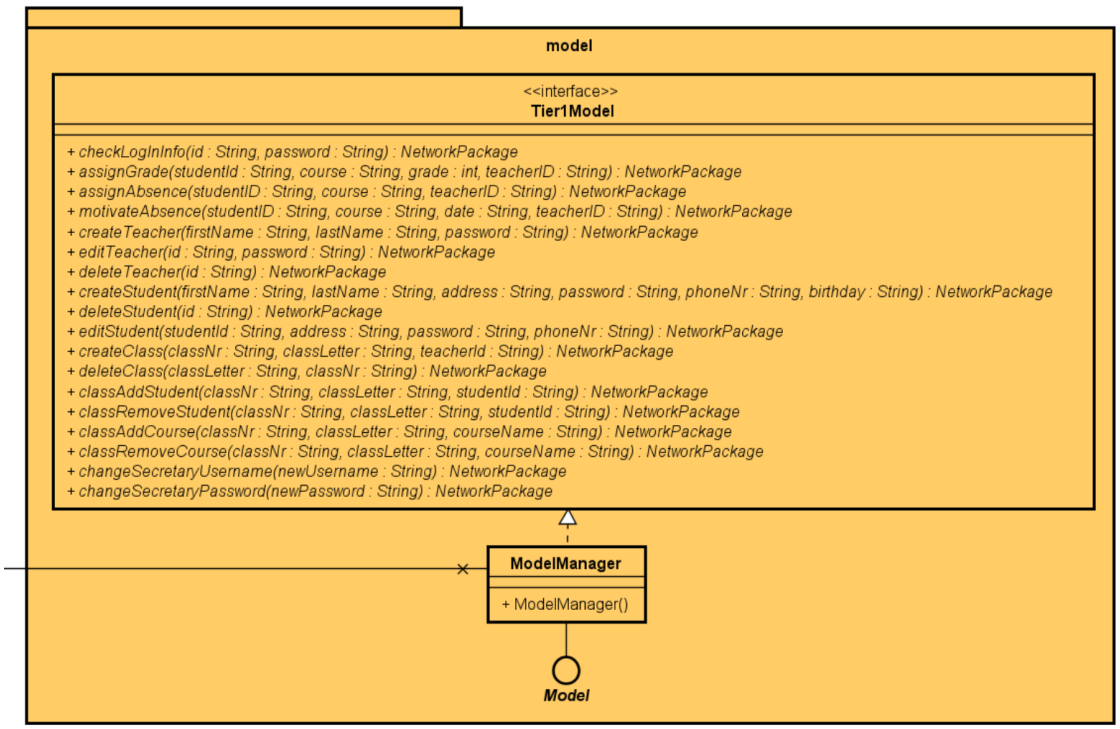


Figure 12 - model folder class diagram

The model manager implements the `Tier1Model` interface and transmits the information from the `tier1Mediator` to the `tier2Mediator`, shown below.

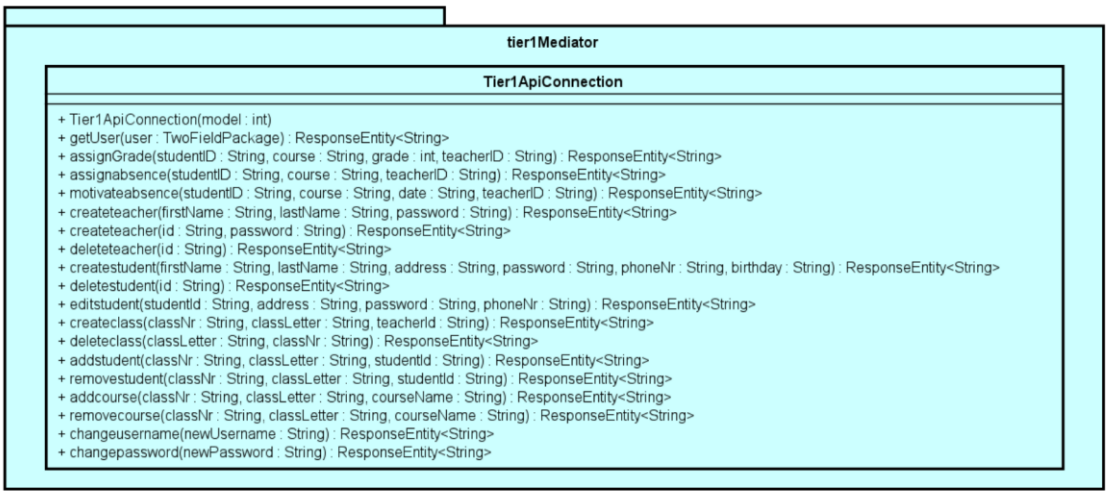


Figure 13 - tier1Mediator folder class diagram

The `Tier1ApiConnection` receives information from Tier 1 through `GET/POST` requests and transmits them to the `ModelManager` presented previously.

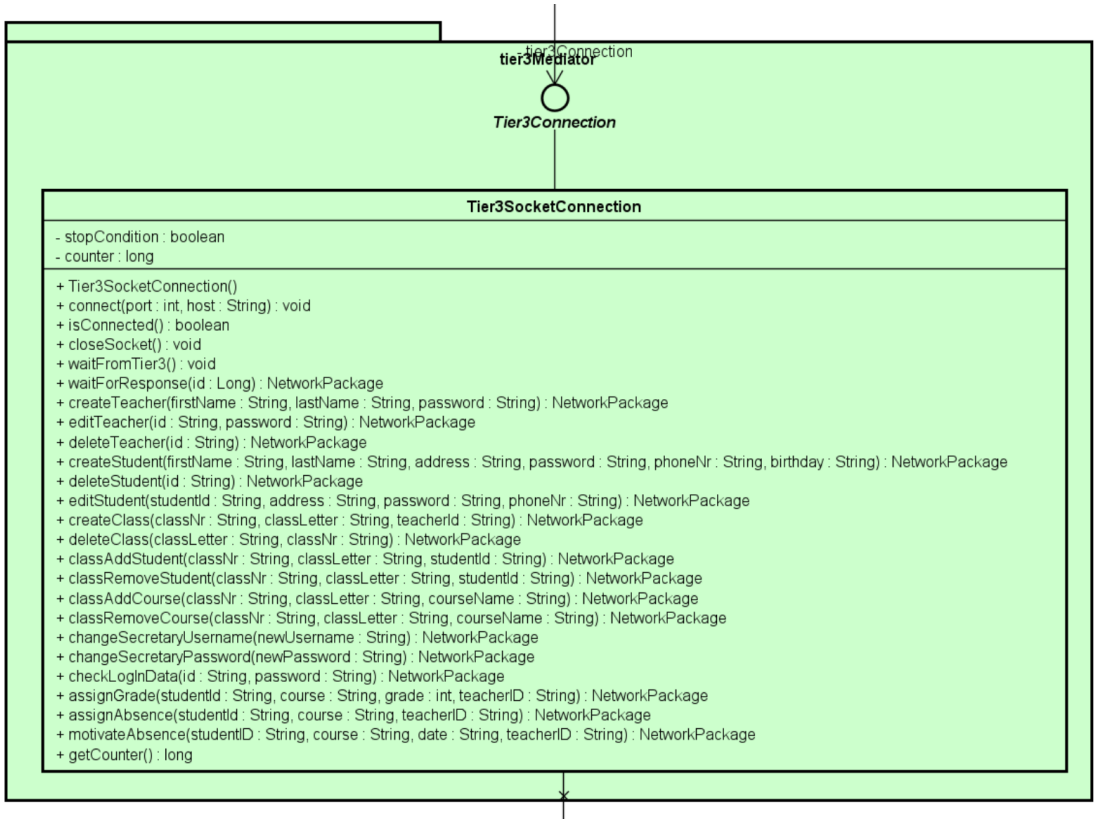


Figure 14 - tier3Mediator folder class diagram

The `Tier3SocketConnection` receives the information from the `Tier1ApiConnection` sent through the `ModelManager` and sends it further down to `Tier3` by connecting through sockets.

4.5. Tier 3 - Server 2 (Java)

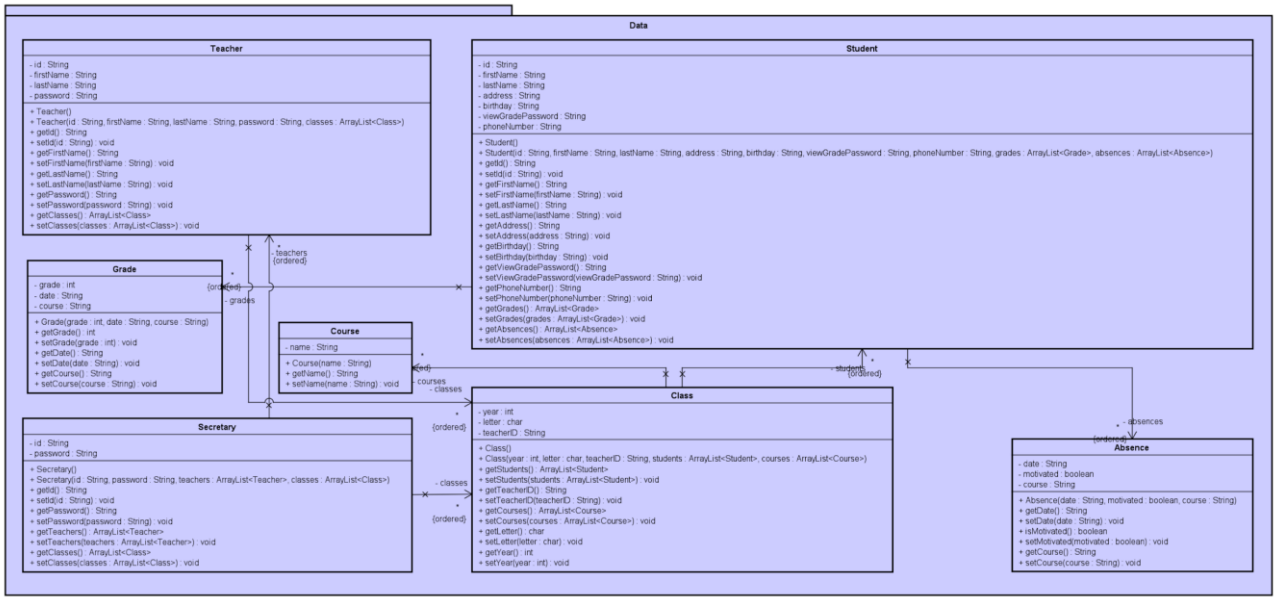


Figure 15 - Data folder class diagram

The `Data` folder is the same as the one from `Tier2` and the classes inside serve the same function.

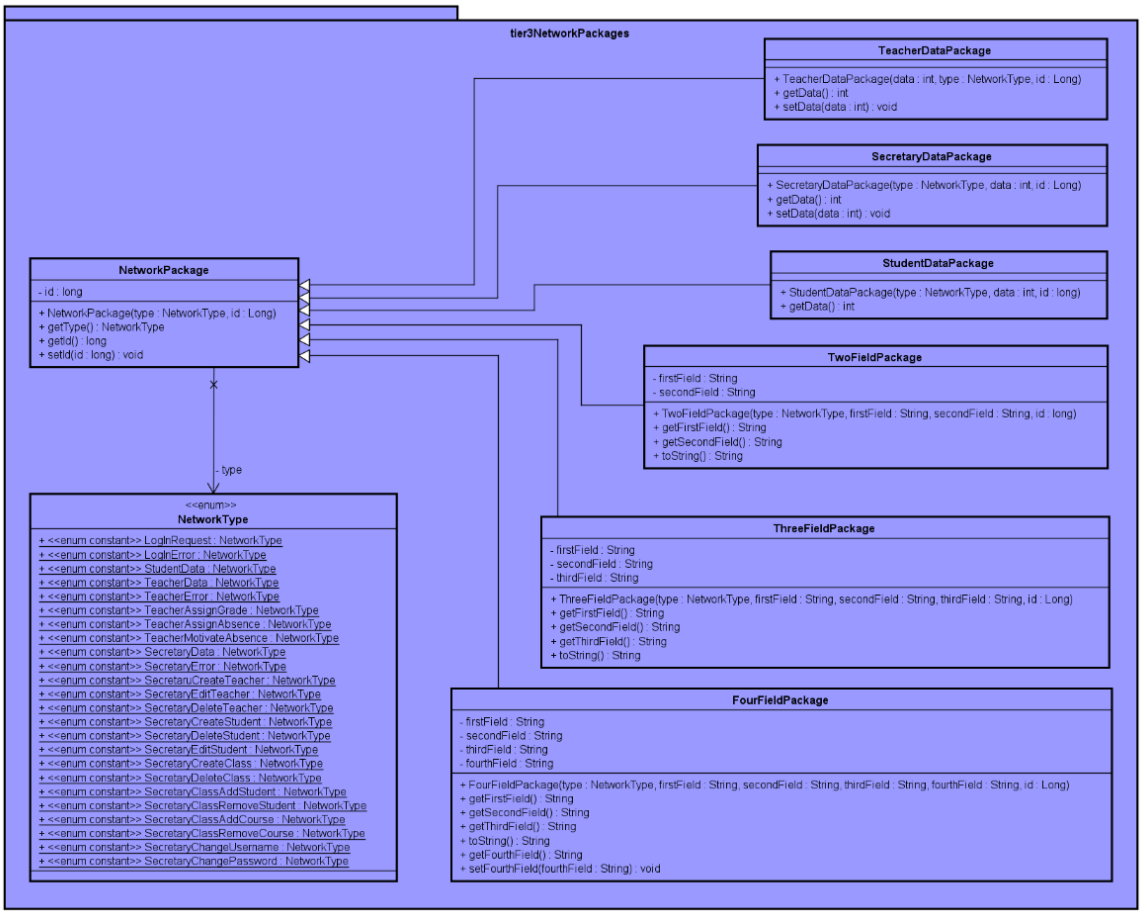


Figure 16 - tier3NetworkPackages folder class diagram

The `tier3NetworkPackages` folder is the same as the one from `Tier2` and the classes inside serve the same function.

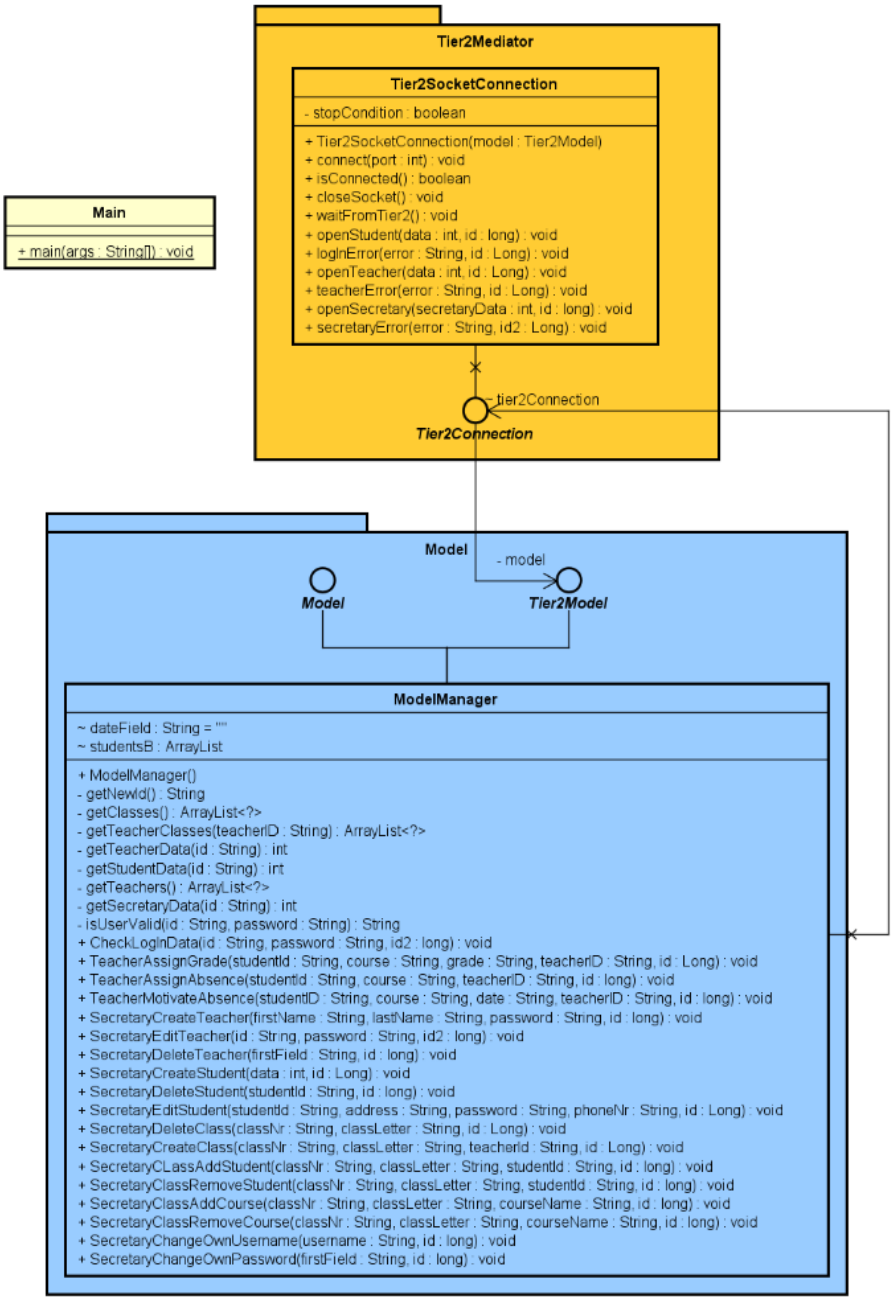


Figure 17 - Tier2Mediator and Model folder class diagram

The `Tier2SocketConnection` mediates the data sent to and from `Tier 2` through sockets. The `ModelManager` sends this data to the database in the form of statements through `SQL` queries.

4.6. Sequence diagrams

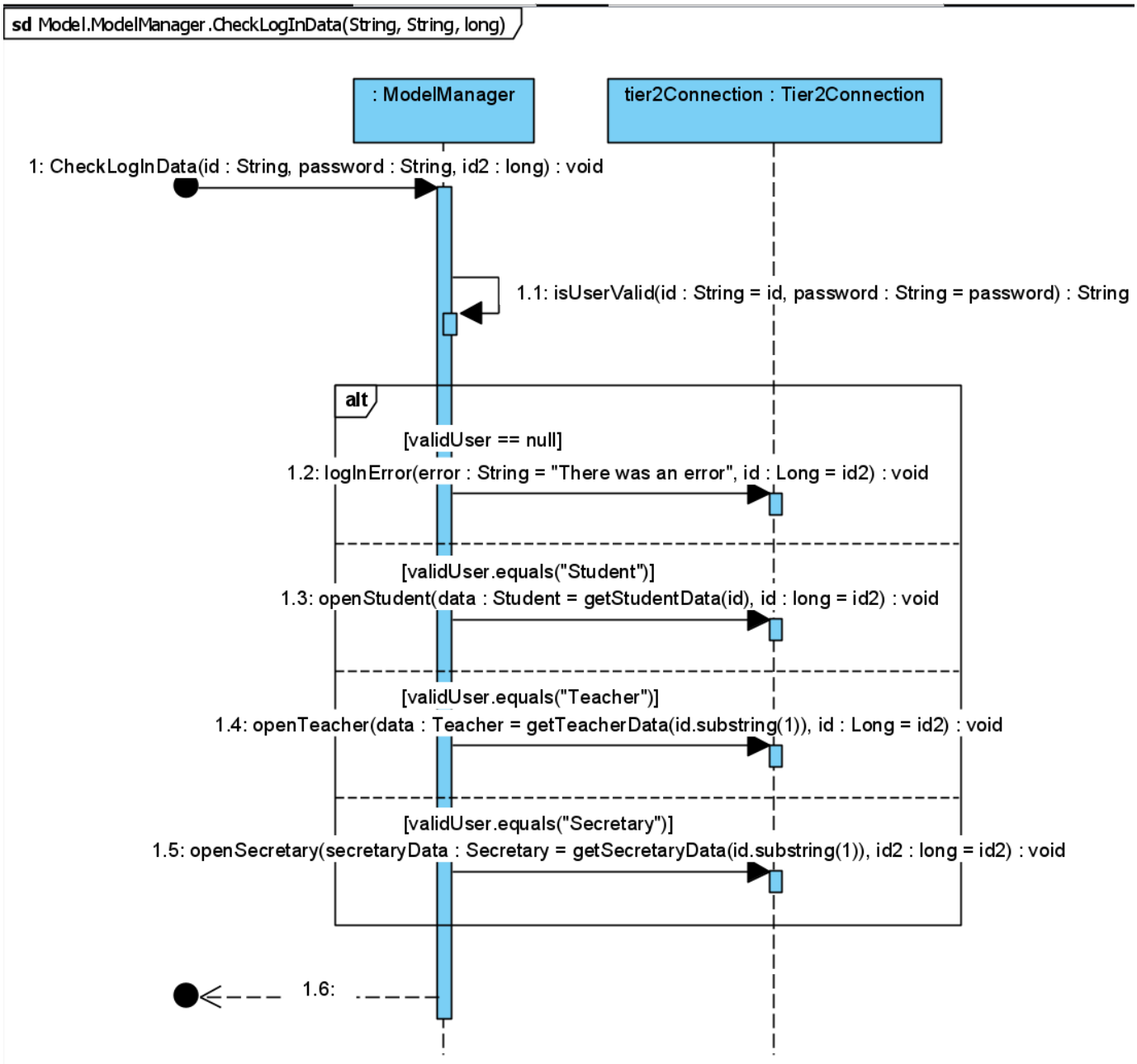


Figure 18 - Model Manager Check Log-in data Sequence diagram

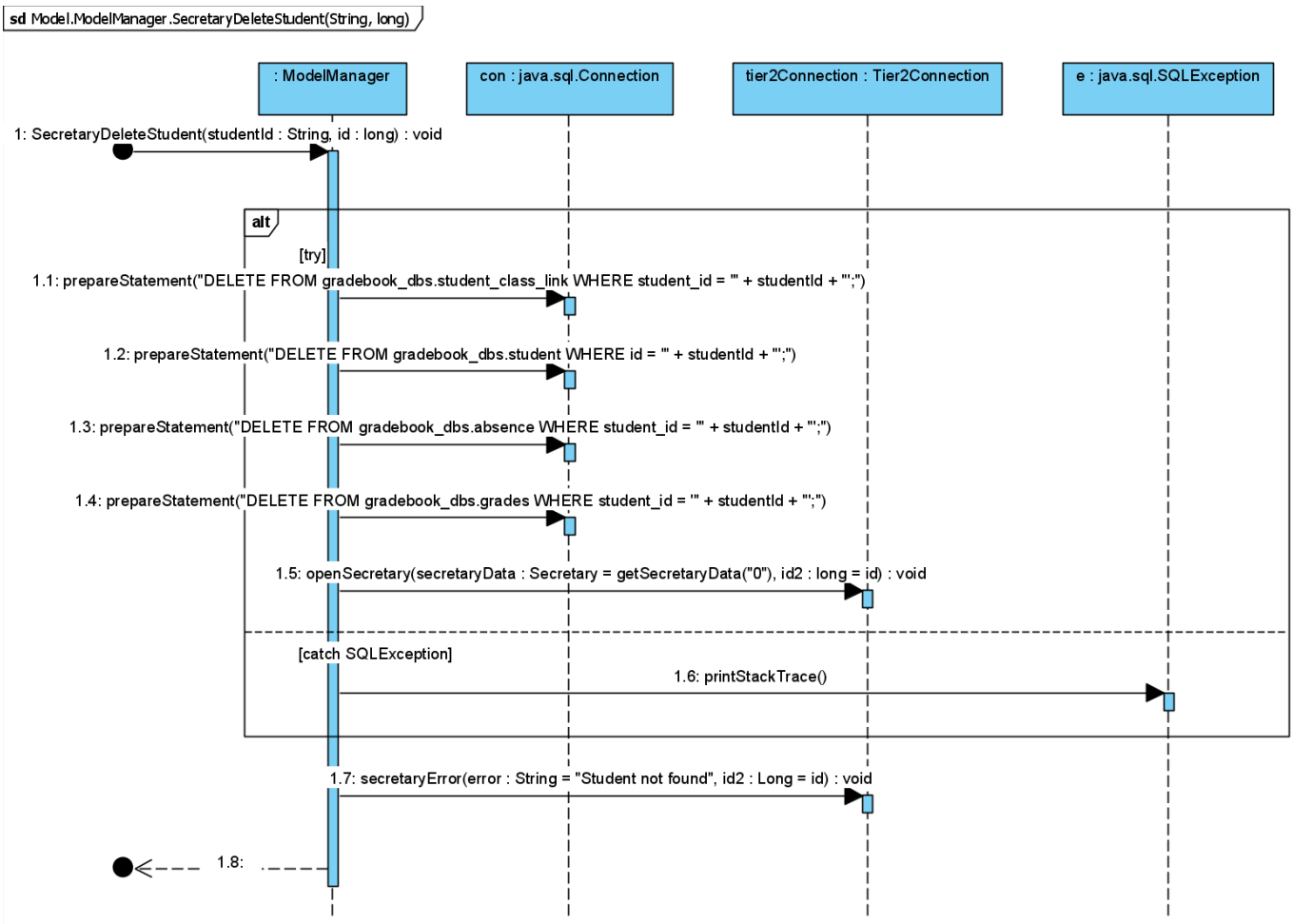


Figure 19 - Model Manager Secretary Delete Student Sequence diagram

4.7. Database

DML examples:

SELECT * FROM gradebook_dbs.student_class_link

SELECT * FROM gradebook_dbs.course WHERE class_id =

SELECT * FROM gradebook_dbs.teacher_class_link

DELETE FROM gradebook_dbs.course WHERE name = 'Math' AND class_id = '12';

SELECT * FROM gradebook_dbs.course

INSERT INTO gradebook_dbs.course VALUES(DEFAULT, 'SUGI', '1')

SELECT * FROM gradebook_dbs.student_class_link WHERE class_id =

SELECT * FROM gradebook_dbs.course WHERE class_id =

DELETE FROM gradebook_dbs.class WHERE id = '5'

SELECT * FROM gradebook_dbs.class WHERE year = '10' AND letter = 'C'

SELECT * FROM gradebook_dbs.secretary

UPDATE gradebook_dbs.secretary SET username = 'admin' WHERE id = '4'

UPDATE gradebook_dbs.secretary SET password = 'admin' WHERE username = 'admin'

ER Diagram:

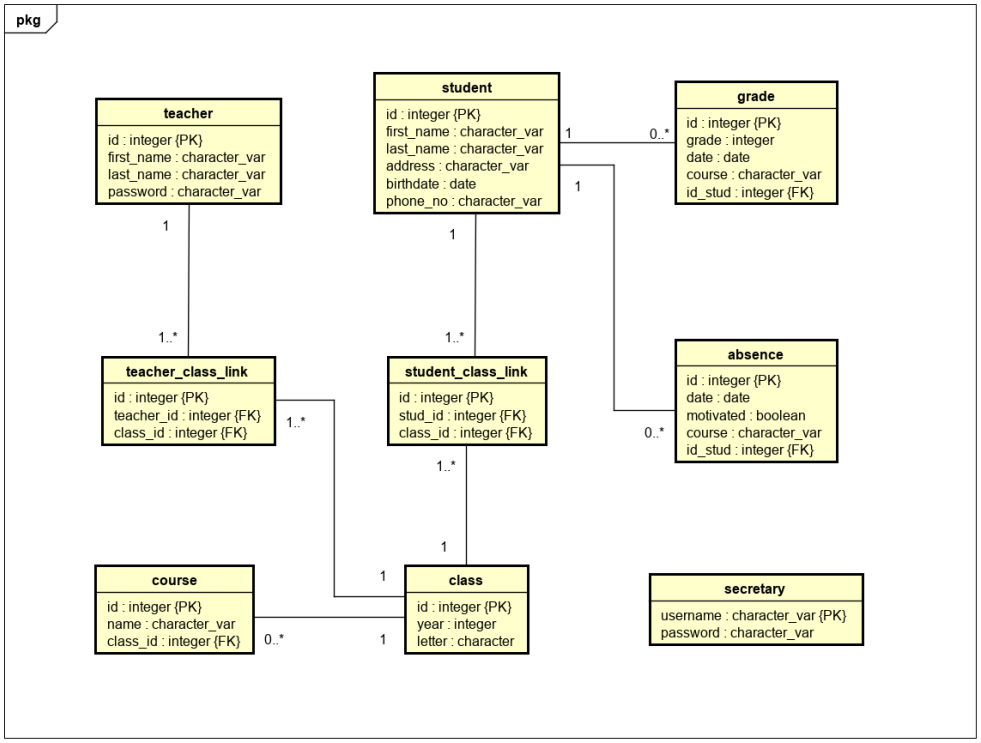
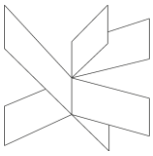


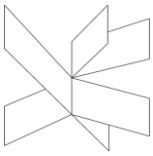
Figure 20 - ER Diagram



4.8. Technologies

The software is made up of 3 tiers, the first one written in the Blazor open-source web framework, while the other 2 in Java. The .Net IDE used for Tier 1 was JetBrains Rider based on the IntelliJ platform. The languages used in Tier 1 were mostly C#, HTML, CSS and occasionally JavaScript. The Bootstrap framework was used for most of the UI design, although in places, custom CSS classes were created. The NuGet package manager was used to install plugins such as the BlazorDateRangePicker. The communication to the second Tier was done through Web Service communication.

Both the second and the third Tier were written in Java in the IntelliJ IDEA. The web services were built in Spring Boot. Maven has also been used to aid in the package management of the project. The communication between the second and the third tier has been made through the use of sockets. Gson has also been used for serialization and deserialization of data. Github has been used for code sharing and collaboration between team members. For the database, PostgreSQL has been chosen. PgAdmin is being used as the database management tool. Astah Professional has been used for the diagram creation. The documentation has been written in Google Docs and at the end converted into PDF format.



5. Implementation

5.1. Tier 1

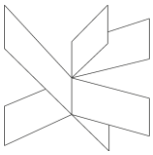
5.1.1. Login

```
public async Task PerformLogin()
{
    errorMessage = "";
    try
    {
        await ((CustomAuthenticationStateProvider) AuthenticationStateProvider).ValidateLogin(username, password);
        username = "";
        password = "";
        NavigationManager.NavigateTo( uri: CustomAuthenticationStateProvider.NavigateToWindow());
    }
    catch (Exception e)
    {
        errorMessage = e.Message;
    }
}
```

On the login page, upon clicking the login button, the above function is called. It sends the input credentials over to the CustomAuthenticationProvider for verification.

```
try
{
    string response = await userService.ValidateLoginAsync(username, password);
    NetworkPackage resultUser = JsonSerializer.Deserialize<NetworkPackage>(response);
    User user = new User();
    if (resultUser.type.Equals("StudentData"))
    {
        StudentDataPackage studentDataPackage = JsonSerializer.Deserialize<StudentDataPackage>(response);
        user.UserName = studentDataPackage.data.Id;
        user.Password =  studentDataPackage.data.Password;
        user.SecurityLevel = 1;
        CachedStudent = studentDataPackage.data;
        RestoreWindowBooleans();
        studentWindow = true;
    } else if (resultUser.type.Equals("TeacherData"))
    {
        TeacherDataPackage teacherDataPackage = JsonSerializer.Deserialize<TeacherDataPackage>(response);
        user.UserName = teacherDataPackage.data.Id;
        user.Password =  teacherDataPackage.data.Password;
        user.SecurityLevel = 2;
        CachedTeacher = teacherDataPackage.data;
        RestoreWindowBooleans();
        teacherWindow = true;
    } else if (resultUser.type.Equals("SecretaryData"))
```

Inside the CustomAuthenticationProvider, the login credentials are sent over to the UserService, where they are sent to tier2, and subsequently tier3 for validation.



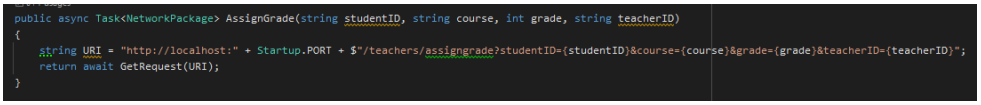
```
1 usage
public static string NavigateToWindow()
{
    if (studentWindow) return "/Student";
    else if (teacherWindow) return "/Teacher";
    else return "/SecretaryTeacher";
}
```

Depending on what type of NetworkPackage is being returned, the respective page will open. The Logout function acts similarly.

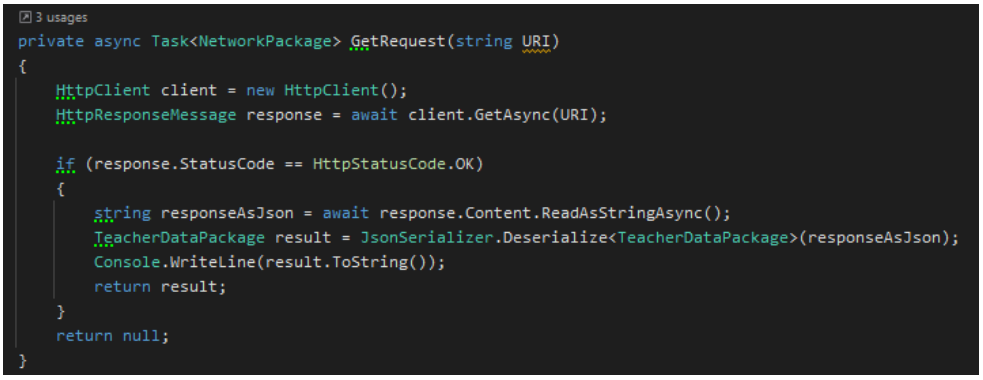
5.1.2 Requests to Tier 2

```
async Task AssignGrade()
{
    NetworkPackage package = await teacherService.AssignGrade(selectedStudent.Id, selectedCourseAsString,  grade: Int32.Parse(selectedGradeAsString), teacher.Id);
    Console.WriteLine(selectedStudent.Id + " " + selectedCourseAsString + " " + Int32.Parse(selectedGradeAsString));
    if (package != null)
    {
        TeacherDataPackage teacherDataPackage = (TeacherDataPackage) package;
        CustomAuthenticationStateProvider.CachedTeacher = teacherDataPackage.data;
        teacher = teacherDataPackage.data;
    }
    else if (package == null)
    {
        Console.WriteLine("BAD REQUEST - GRADE ASSIGNMENT");
    }
}
```

Any requests, such as assigning a grade, is sent over to the respective Service class, such as the TeacherService class in this example. In case of a successful Network Package return, the data is assigned to the cached object, such as the Student/Teacher/Secretary, otherwise, an error is displayed.

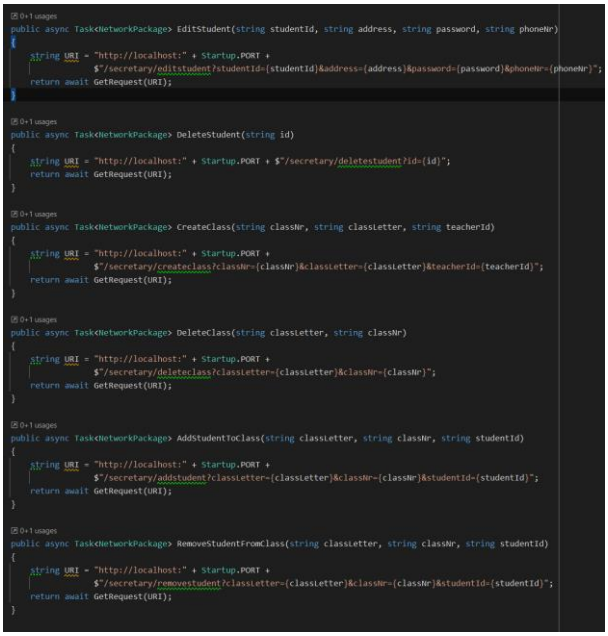


```
public async Task<NetworkPackage> AssignGrade(string studentID, string course, int grade, string teacherID)
{
    string URI = "http://localhost:" + Startup.PORT + $"/teachers/assigngrade?studentID={studentID}&course={course}&grade={grade}&teacherID={teacherID}";
    return await GetRequest(URI);
}
```



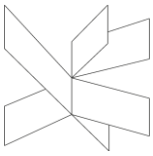
```
3 usages
private async Task<NetworkPackage> GetRequest(string URI)
{
    HttpClient client = new HttpClient();
    HttpResponseMessage response = await client.GetAsync(URI);

    if (response.StatusCode == HttpStatusCode.OK)
    {
        string responseAsJson = await response.Content.ReadAsStringAsync();
        TeacherDataPackage result = JsonSerializer.Deserialize<TeacherDataPackage>(responseAsJson);
        Console.WriteLine(result.ToString());
        return result;
    }
    return null;
}
```

Inside the Service class, the data is sent over through either as a GET request or a POST one. The data is received back and deserialized as a Network Package, the type of which depends on the request URI.



```
0+1 usages
public async Task<NetworkPackage> EditStudent(string studentId, string address, string password, string phoneNr)
{
    string URI = "http://localhost:" + Startup.PORT +
                 $"/secretary/editstudent?studentId={studentId}&address={address}&password={password}&phoneNr={phoneNr}";
    return await GetRequest(URI);
}

0+1 usages
public async Task<NetworkPackage> DeleteStudent(string id)
{
    string URI = "http://localhost:" + Startup.PORT + $"/secretary/deletestudent?id={id}";
    return await GetRequest(URI);
}

0+1 usages
public async Task<NetworkPackage> CreateClass(string classNr, string classLetter, string teacherId)
{
    string URI = "http://localhost:" + Startup.PORT +
                 $"/secretary/createclass?classNr={classNr}&classLetter={classLetter}&teacherId={teacherId}";
    return await GetRequest(URI);
}

0+1 usages
public async Task<NetworkPackage> DeleteClass(string classLetter, string classNr)
{
    string URI = "http://localhost:" + Startup.PORT +
                 $"/secretary/deleteclass?classLetter={classLetter}&classNr={classNr}";
    return await GetRequest(URI);
}

0+1 usages
public async Task<NetworkPackage> AddStudentToClass(string classLetter, string classNr, string studentId)
{
    string URI = "http://localhost:" + Startup.PORT +
                 $"/secretary/addstudent?classLetter={classLetter}&classNr={classNr}&studentId={studentId}";
    return await GetRequest(URI);
}

0+1 usages
public async Task<NetworkPackage> RemoveStudentFromClass(string classLetter, string classNr, string studentId)
{
    string URI = "http://localhost:" + Startup.PORT +
                 $"/secretary/removestudent?classLetter={classLetter}&classNr={classNr}&studentId={studentId}";
    return await GetRequest(URI);
}
```

In order to make the code more readable, all the URIs are passed down to the GetRequest(string URI) method, as not to have code duplication.



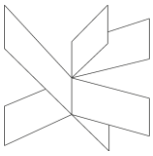
5.1.3. Other details

```
<AuthorizeView Policy="SecurityLevel3">
    <div class="sidebar">
        <SecretarySidebar/>
    </div>
</AuthorizeView>

<div class="main">
    <AuthorizeView>
        <div class="top-row px-4">
                <Authorized>
                  <a href="" @onclick="PerformLogout">
                    Log out
                    </a>
                </Authorized>
        </div>
    </AuthorizeView>

    <div class="content px-4">
        @Body
    </div>
</div>
```

Some elements are hidden depending on the SecurityLevel of the user logged-in. The Log-out button is only displayed to logged-in users, while the SecretarySidebar is only displayed only to the Secretary.

```
.blue-border{
    border-width: 2px;
    border-style: solid;
    border-color: #03c2fc;
    border-radius: 25px;
    padding: 0.5em;
}

.center-horiz{
    margin:auto;
    width: fit-content;
    text-align: center;
}

.selector{
    border:none;
    height: 2em;
    text-align: center;
    text-align-last:center;
}
```

`overallTheme.css` holds the thematics used all across the software, except for the Login page.



5.2. Tier 2

5.2.1. Web-service connection to Tier 1 using REST

```
@PostMapping("/users")
public ResponseEntity<String> getUser(@RequestBody TwoFieldPackage user)
{
    String username = user.getFirstField();
    String password = user.getSecondField();
    NetworkPackage networkPackage = model.checkLogInInfo(username, password);
    if (networkPackage.getType().equals(NetworkType.StudentData)) {
        StudentDataPackage studentDataPackage = (StudentDataPackage) networkPackage;
        String studentAux = gson.toJson(studentDataPackage);
        return new ResponseEntity<String>(studentAux, HttpStatus.OK);
    } else if (networkPackage.getType().equals(NetworkType.TeacherData)) {
        TeacherDataPackage teacherDataPackage = (TeacherDataPackage) networkPackage;
        String teacherAux = gson.toJson(teacherDataPackage);
        return new ResponseEntity<String>(teacherAux, HttpStatus.OK);
    } else if (networkPackage.getType().equals(NetworkType.SecretaryData)) {
        SecretaryDataPackage secretaryDataPackage = (SecretaryDataPackage) networkPackage;
        String secretaryAux = gson.toJson(secretaryDataPackage);
        return new ResponseEntity<String>(secretaryAux, HttpStatus.OK);
    } else return new ResponseEntity<String>(HttpStatus.BAD_REQUEST);
}
```

Shown above is the ‘getUser’ method from the ‘Tier1ApiConnection’ class that uses SpringBoot connected to the Presentation layer(Tier 1). This method checks the login data received from Tier 1 via a ‘Network Package’ that contains the ID and password of an user and returns the appropriate user data package according to the data received from the database layer(Tier 3) or if the data given is wrong it returns an error package.

5.2.2. Tier 1 and Tier 3 connection method

```
24      public NetworkPackage checkLogInInfo(String id, String password)
25      {
26          return tier3Connection.checkLogInData(id, password);
27      }
```

The method shown above **'checkLogInInfo'** is a method located in the 'ModelManager' class that connects the 'Tier1APIConnection' and 'Tier3SocketConnection' classes.

5.2.3. Tier 2 to Tier 3 socket communication

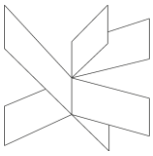
```
260     public NetworkPackage checkLogInData(String id, String password)
261     {
262         long currentCounter = getCounter();
263         writer.println(gson.toJson(new TwoFieldPackage(NetworkType.LogInRequest, id, password, currentCounter)));
264
265         return this.waitForResponse(currentCounter);
266     }
```

The method shown above is used to communicate to the Database Layer via TCP Socket sending a Network Package that is translated into a Json file that can be transported over the TCP Socket, each package containing a unique id that is used by the 'waitForResponse' method that returns the response package received via the TCP Socket from Tier 3 removing the need to have a callback function.

5.2.4. Waiting for a response network package from Tier 3

```
125     public NetworkPackage waitForResponse(Long id)
126     {
127         NetworkPackage responsePackage = null;
128         boolean bai = true;
129         while(bai) {
130             for (int i = 0; i <  requestList.size(); i++) {
131                 if (requestList.get(i).getId() == id)
132                 {
133                     responsePackage = requestList.get(i);
134                     requestList.remove(i);
135                     bai = false;
136                 }
137             }
138             try {
139                 Thread.sleep( millis: 50);
140             } catch (InterruptedException e) {
141                 e.printStackTrace();
142             }
143         }
144         return responsePackage;
145     }
```

The above method await for a response from the Tier 3 socket connection that is automatically placed into an ArrayList **'requestList'**, each **'NetwokPackage'** inside this ArrayList has an unique that this method is looking given at the call of this method, returning the **'NetworkPackage'** that has this id at the moment when it is found. This method is used to remove the need for a callback function to Tier 1 because the TCP Socket used for connection Tier 2 and Tier 3 is asynchronous.



5.3. Tier 3

5.3.1. Await request from Tier 2

```
public void waitFromTier2()
{
    Runnable runnable = () -> {
        String message = null;
        NetworkPackage dataPackage;
        while (!stopCondition) {
            try {
                message = reader.readLine();
            } catch (IOException e) {
                System.out.println(socket.getRemoteSocketAddress().toString() + " - disconnected");
                stopCondition = true;
                e.printStackTrace();
            }

            dataPackage = gson.fromJson(message, NetworkPackage.class);

            switch (dataPackage.getType()) {
                case LogInRequest:
                    TwoFieldPackage twoFieldPackage = gson.fromJson(message, TwoFieldPackage.class);
                    System.out.println("Log In Request" + twoFieldPackage.getFirstField());
                    model.CheckLogInData(twoFieldPackage.getFirstField(), twoFieldPackage.getSecondField(), twoFieldPackage.getId());
                    break;
```

The method shown above **'awaitFromTier2'** is used to wait for TCP Socket requests from Tier 2, decoding the Json packages into the appropriate network packages and calling the method requested by Tier2.

5.3.2. Returning the appropriate data package to Tier 2

```
338         @Override
339         public void CheckLogInData(String id, String password, long id2)
340         {
341             ///FOR NOW
342             String validUser = isUserValid(id, password);
343             if (validUser == null) {
344                 tier2Connection.logInError( error: "There was an error", id2);
345                 System.out.println("!!!!!!!!!!!!!!!!");
346             } else if (validUser.equals("Student"))
347                 tier2Connection.openStudent(getStudentData(id), id2);
348             else if (validUser.equals("Teacher"))
349                 tier2Connection.openTeacher(getTeacherData(id.substring(1)), id2);
350             else if (validUser.equals("Secretary"))
351                 tier2Connection.openSecretary(getSecretaryData(id.substring(1)), id2);
352         }
```

The method shown above used the **'isUsedValid'** method to determine if the login data received is correct and if it is it calls the appropriate response according to the response received from the **'isUserValid'** method. For example if the received information are of a 'Student' account, this method calls the 'openStudent' method from the Tier2SocketConnection' method that send the **'StudentDataPackage'** of that particular student.

5.3.3. Checking the login data received

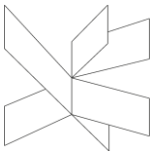
```
@   private String isUserValid(String id, String password)
    {

        if(id.charAt(0) != 'T' && id.charAt(0) != 'S')
        {
            try {
                ResultSet rs = con.prepareStatement( sql: "SELECT * FROM gradebook_dbs.student WHERE id ='" + id + "'").executeQu
                while (rs.next()) {
                    if (rs.getString( columnIndex: 1).equals(id) && rs.getString( columnIndex: 5).equals(password)) {
                        return "Student";
                    } else System.out.println(rs.getString( columnIndex: 1) + " - " + rs.getString( columnIndex: 5));
                }

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
            if(id.charAt(0) == 'T')
            {
                id = id.substring(1);
            try {
                ResultSet rs = con.prepareStatement( sql: "SELECT * FROM gradebook_dbs.teacher WHERE id ='" + id + "'").executeQu
                while (rs.next()) {
                    if (rs.getString( columnIndex: 1).equals(id) && rs.getString( columnIndex: 4).equals(password)) {
                        return "Teacher";
                    }
                }

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(id.charAt(0) == 'S')
        {
            id = id.substring(1);
            try {
                ResultSet rs = con.prepareStatement( sql: "SELECT * FROM gradebook_dbs.secretary WHERE username ='" + id + "').e
                while (rs.next()) {
                    if (rs.getString( columnIndex: 2).equals(id) && rs.getString( columnIndex: 3).equals(password)) {
                        return "Secretary";
                    } else System.out.println("NOPE");
                }

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return null;
    }
```

This method checks if the login data received corresponds to any of the users registered in the PostGreSQL Database and if it is found it returns the type of user found, otherwise it returns a null String stating that there is no user registered with this data.

5.3.4. Creating and returning a Teacher Data Package

```
private Teacher getTeacherData(String id)
{
    System.out.println("----------TEACHER DATA ");
    Teacher teacher = null;
    ArrayList<Class> classes = this.getTeacherClasses(id);


    ResultSet rs3 = null;
    try {
        rs3 = con.prepareStatement( sql: "SELECT * FROM gradebook_dbs.teacher WHERE id ='" + id + "'").executeQuery();
        while (rs3.next())
        {
            teacher = new Teacher(rs3.getString( columnIndex: 1), rs3.getString( columnIndex: 2), rs3.getString( columnIndex: 3), rs3.getString( columnIndex: 4), classes);
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return teacher;
}
```

The method shown above created and returns a **Teacher** data object according to the teacher user with the id provided when calling this function by getting the data of this particular teacher from the PostGreSQL Database and creating a new Teacher object then returning it.



5.3.5. Returning the teacher package

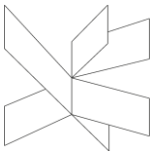
```
@Override
public void openTeacher(Teacher data, Long id)
{
    writer.println(gson.toJson(new TeacherDataPackage(data, NetworkType.TeacherData, id)));
}
```

The method shown above is used in the **'Tier2SocketConnection'** class to return a **'TeacherDataPackage'** to Tier 2 containing the data a teacher user needs to conduct his operations.

5.3.6. Deleting a student from the database

```
@Override
public void SecretaryDeleteStudent(String studentId, long id)
{
    try {
        con.prepareStatement( sql: "DELETE FROM gradebook_dbs.student_class_link WHERE student_id = '" + studentId + "';").executeUpdate();
        con.prepareStatement( sql: "DELETE FROM gradebook_dbs.student WHERE id = '" + studentId + "';").executeUpdate();
        con.prepareStatement( sql: "DELETE FROM gradebook_dbs.absence WHERE student_id = '" + studentId + "';").executeUpdate();
        con.prepareStatement( sql: "DELETE FROM gradebook_dbs.grades WHERE student_id = '" + studentId + "';").executeUpdate();
        tier2Connection.openSecretary(getSecretaryData( id: "0"), id);
        return;
    } catch (SQLException e) {
        e.printStackTrace();
    }

    tier2Connection.secretaryError( error: "Student not found", id);
}
```

The method shown above deletes all the user data of a Student from the PostGreSQL Database, deleting the absences, grades, links to classes and personal data, deleting all traces of the student ever being in the system. After the deletion a new **'SecretaryDataPackage'** is sent to the Tier 2 to be transmitted to Tier 1, refreshing the data that the secretary uses to manage the system.



6. Testing

The purpose of this section is to demonstrate that the code follows the use case descriptions. The testing method that was used is Black Box Testing. In Black Box Testing, the testing was done from the perspective of the user that doesn't know any information of the code behind making the program work. The results of this testing were gathered in multiple sections of different people testing all the functionality of the program.

6.1. Black Box Testing

6.1.1. View grades and absences

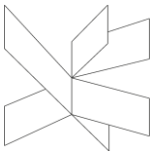
Test ID	Description	Result	Comments
TC - 1.1	The student views his grades and absences	Passed	

6.1.2. View grades and absences of own students

Test ID	Description	Result	Comments
TC - 2.1	The teacher views the grades and absences of students	Passed	The teacher can only view the grades and absences of the students assigned to him/she

6.1.3. Assign grades

Test ID	Description	Result	Comments
TC - 3.1	Teachers assigns grades	Passed	The teacher can only assign grades to the students assigned to him/she



6.1.4. Assign absences

Test ID	Description	Result	Comments
TC - 4.1	Teacher assigns absences	Passed	The teacher can only assign absences to the students assigned to him/she

6.1.5. Excuse absences

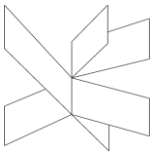
Test ID	Description	Result	Comments
TC - 5.1	Teacher excuses absences	Passed	The teacher can only excuse the absences of the students assigned to him/she

6.1.6. Manage teachers

Test ID	Description	Result	Comments
TC - 6.1	Create teacher	Passed	
TC - 6.2	Edit teacher	Passed	Only the password can be changed
TC - 6.3	Delete teacher	Passed	Teacher can be deleted only if he is not teaching any class anymore

6.1.7. Manage students

Test ID	Description	Result	Comments
TC - 7.1	Create student	Passed	
TC - 7.2	Edit student	Passed	The address, password and phone number can be edited
TC - 7.3	Delete student	Passed	

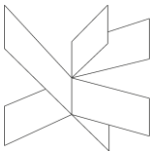


6.1.8. Manage class

Test ID	Description	Result	Comments
TC - 8.1	Create class	Passed	Class needs to have unique year and letter
TC - 8.2	Deleting a class	Passed	
TC - 8.3	Add student	Passed	Student needs to be classless
TC - 8.4	Remove Student	Passed	
TC - 8.5	Add Course	Passed	
TC - 8.6	Remove course	Passed	

6.1.9. Change log in

Test ID	Description	Result	Comments
TC - 9.1	Change password	Passed	These changes apply only for the secretary account
TC - 9.2	Change username	Passed	These changes apply only for the secretary account



7. Results and Discussion

After analyzing the supposed functionality that had to be implemented into the software, as to achieve the desired product, a list of 16 requirements was created. All of them have been successfully implemented.

A login window for all the user types is accessible on the application launch. Subsequently, depending on the user type, they have access to different pages.

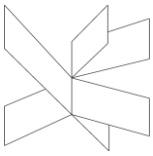
The student has a page in which they can view their own grades, absences and personal information.

The teacher also has only one page in which they can view all of their classes and all the respective students from those classes. Upon selecting a student the teacher can view or add grades and absences. The teacher can also excuse absences.

Finally, the secretary has 4 different pages. One for classes, one for teachers, one for students and one for their own Log-in information.

All the pages combined contain the functionality that fulfills the client's requirements.

There are still a few unintended effects in the UI which have to be solved in the future, such as the entire UI refreshing upon making a change to the student as the teacher, but other than that, the resulting software is working mostly as intended, with all the functionality working well.



8. Conclusions

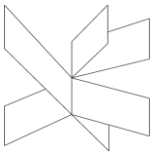
In conclusion, an online gradebook system has been successfully created according to the requirements set in place at the start of the project.

The system created is able to keep track of every grade and absence of any particular student and make respective modifications.

Also, the teacher can assign grades, absences and motivate absences of the students assigned to him by the secretary.

The secretary manages all of the classes, students and teachers, creating the relations between them.

The software follows all of the requirements, as shown by the testing results using the use cases as reference points to what the program needs to do.

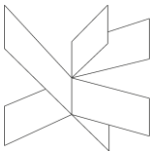


9. Project future

The program made as a result of this project is constantly improving, the current design of this program is modular allowing for easy upgrades and changes without the occurrence of conflicts in other areas of the program.

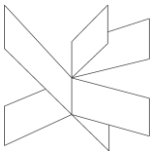
Here are listed a few proposals in terms of what the future of this program could look like:

- Fixing the UI refreshing every time a teacher makes a change to a student
- Removal of grades in case of grade contestation by the student/parent or in case the teacher assigns the wrong grade to the student
- Hashing of the password stored inside the database in case of a security breach
- A mobile java application for easy grade accessibility of scholar situation for students and parents
- Automatic SMS messages to the student's parent's when their respective children receives a new grade or absence
- Encryption of the communication between the 3 tiers of the system
- In app messaging service for teachers to be able to talk to a particular students parents
- Course calendar so that students can know their program



10. Sources of information

1. *k12teacherstaffdevelopment.com.*[online] Available at: *https://k12teacherstaffdevelopment.com/tlb/online-vs-hard-copy-gradebooks-pros-and-cons/* [Accessed 21 Sep. 2020].
2. *Blackboard.com*. [online] Available at: http://www.blackboard.com [Accessed 21 Sep. 2020].
3. *Moodle.org*. [online] Available at: http://moodle.org [Accessed 21 Sep. 2020].
4. *SAGrader*. [online] Available at: https://www.sagrader.com [Accessed 21 Sep. 2020].
5. Czaplewski, A. J. (2009) Computer-Assisted Grading Rubrics: Automating the Process of Providing Comments and Student Feedback, Marketing Education Review, 19[book].
6. Jones, E. L. (2001) Grading Student Programs – A Software Testing Approach, Journal of Computing Sciences in Colleges, 16[book]



11. Appendices

- A – Project Description
- B – Analysis
- C – Process Report
- D – SCRUM
- E – User Guide
- F – Class Diagrams