**Post-secondary School Academic and Health Records Database**

Yongyi Zang Weihong Qi Erqian Xu

**1 Introduction**

**1.1 Problem Statement**

AHR database, short for post-secondary school academic and health records database, aims to help post-secondary school administration, faculty, health workers, and parents to track students’ information in real-time and provide support if necessary. It will store students’ various characteristics, including demographic, academic-related, and health-related information. Here, demographic information ranges from userID, username, a password for login, unique student ID, sex, age, and home address, to who his/her guardian is. Academic-related information contains the reason to choose this school, the number of school absences, whether or not there is family educational support, whether or not there is extra educational support, whether or not a student attends extra paid classes, and whether or not the student attends extracurricular activities, whether or not a student has attended nursery school before, student’s educational aspirations (i.e., whether or not a student wants to take higher education), mathematics and Portuguese language scores during multiple school periods. We will also include students’ health-related information to track their COVID vaccination status and dates in this database. We purposefully include COVID vaccination status in this database since it is of critical importance during the pandemic and may potentially impact each student’s future quarantine requirement.

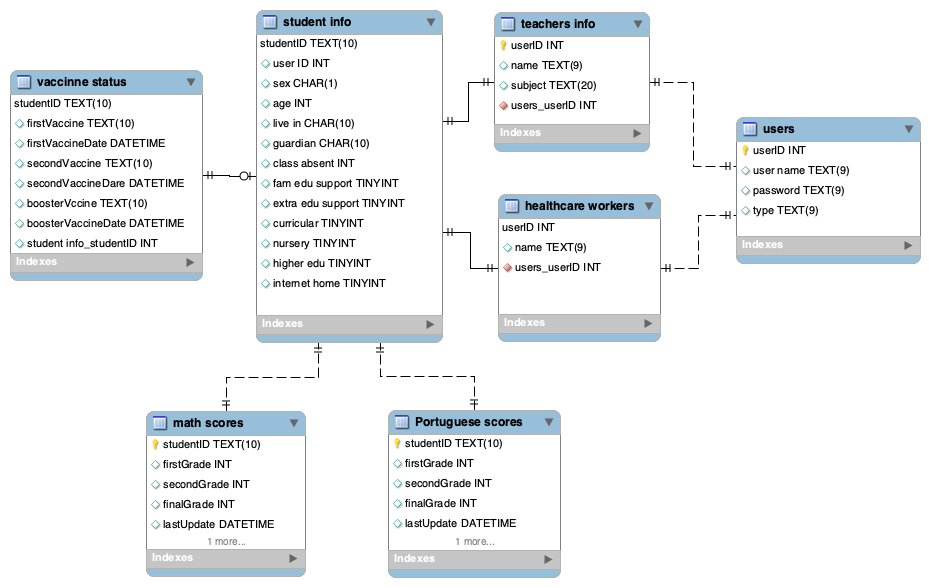
After creating a series of tables, the AHR database will better help us build the relationship between tables, which will be challenging to achieve through excel files. Compared to excel files, establishing our AHR database will also allow multiple users to monitor or update students’ information simultaneously, while with different authorizations of viewing or editing. In addition, the AHR database will protect students’ privacy from irrelevant personnel. For example, students’ primary care doctors will be only allowed to access and update COVID vaccination records but cannot peep into their patients’ academic performance.

**1.2 Target User**

The target users of the AHR database include parents, teachers, health workers and administrators. Parents will be able to view all the information but only have the authorization to edit students' demographic information. The educational practitioners will monitor academic-related items. Health workers such as primary care doctors and nurses will be responsible for updating students' health-related information. On the other hand, school administrators cannot edit anything but have the authorization to view all the information. This system will allow different people separate levels of information to better protect student privacy.

**1.3 List of Relations**

As seen in Figure 1, we have created the schema for the AHR database. It includes several sub-databases: users, *teachers' info*, *student info*, *healthcare workers*, *math scores*, *Portuguese scores*, and *vaccine status*. *Health workers*, *teacher info*, and *student info* are linked to the *users* sub-database. Each sub-database contains multiple attributes. More examples for each attribute can be found in Appendix A.

******

**Figure 1.** AHR Database Schema

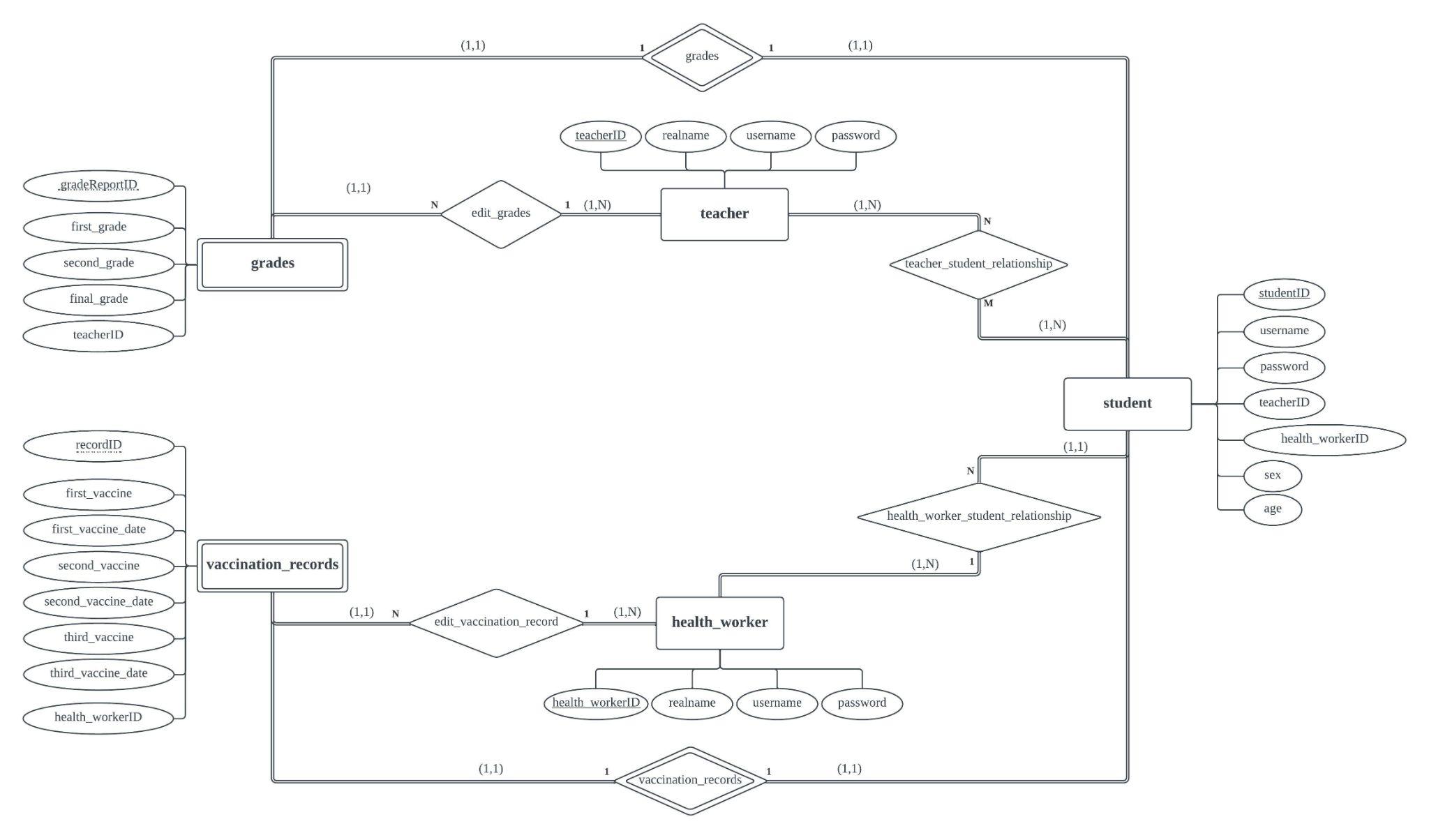
**1.4 Data**

The data is mainly based on the Student Performance Data Set we retrieved from UCI Machine Learning Repository (<https://archive.ics.uci.edu/ml/datasets/student+performance>). In the meanwhile, other data that are not present within the UCI dataset will be populated randomly by us to serve the purpose of the database. Specifically, studentID will be pre-generated by an outside python program. Also, we will randomly generate data for vaccination status and vaccination dates for both shots and boosters since we will add a simple COVID vaccine monitor feature to the application. If more than one shot is applicable, we will use 21 days and 180 days apart respectively between first, second and third shot, as recommended by Centers for Disease Control and Prevention (CDC).

**2 Database management system design**

**2.1 ER diagram**

We began this project by designing the following ER diagram:



**Figure 2.** ER Diagram

As depicted in Figure 2, we followed Chen Notation to create this diagram. The cardinality, total participation and (min, max) constraints are enumerated as on our diagram. There are in total of three strong entities as follows:

1. Student
2. Health\_worker
3. Teacher

And also two weak entities:

1. Grades
2. Vaccination\_records

There are no class hierarchies in our ER diagram.

Our assumptions are as follows:

1. All students have one (and only one) vaccination record (identified by recordID.)
2. All students have one (and only one) grade report (identified by gradeReportID in grades.)
3. Each grade report is only created and can be only edited by one teacher.
4. Each vaccination record is only created and can only be edited by one health worker.
5. Each student should have at least one teacher and one health worker.
6. Each student could have multiple teachers, but only one health worker.
7. Every teacher would have at least one student, and every health worker should have at least one student.
8. Each teacher could teach multiple students.
9. Each health worker could oversee multiple students.

**2.2** **Relational Database Design Using ER-to-Relational Mapping**

***Step 1: Mapping of Strong Entities***

We have a total of 3 strong entities in our database. Each entity has its own primary key:

**student**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| studentID | username | password | teacherID | health\_workerID | sex | age |

**teacher**

|  |  |  |  |
| --- | --- | --- | --- |
| teacherID | realname | username | password |

**health\_worker**

|  |  |  |  |
| --- | --- | --- | --- |
| health\_workerID | realname | username | password |

***Step 2: Mapping of Weak Entities***

We have a total of 2 weak entities in our database: grades and vaccination\_records. In this step, we will create two relations: grades and vaccination\_records, corresponding to grades and vaccination\_records.

**grades**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| studentID | gradeReportID | first\_grade | second\_grade | final\_grade | teacherID |

The primary key of grades is the combination {studentID, gradeReportID}, since gradeReportID is the partial key of grades.

**vaccination\_records**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| studentID | recordID | first\_vaccine | first\_vaccine\_date | second\_vaccine | second\_vaccine\_date | third\_vaccine | third\_vaccine\_date | health\_workerID |

The primary key of vaccination\_records is the combination {studentID, recordID}, since recordID is the partial key of vaccination\_records.

***Step 3: Mapping of Binary 1:1 Relation Types***

The only two 1:1 relations we have are from Step 2, the weak entities. For this step, we decided to use the foreign key approach. For both of these relations, we would see the **student** as the “T” in our relationship, and the **grades** or **vaccination\_records** as the “S” in this relationship. Since the primary key (**“studentID”**) is already in both **grades** and **vaccination\_records**, we don’t need to complete this step.

**grades**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| studentID | gradeReportID | first\_grade | second\_grade | final\_grade | teacherID |

**vaccination\_records**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| studentID | recordID | first\_vaccine | first\_vaccine\_date | second\_vaccine | second\_vaccine\_date | third\_vaccine | third\_vaccine\_date | health\_workerID |

Just to reiterate, here the studentID in grades would point to studentID in student, and the studentID in vaccination\_records would point to studentID in student as well.

***Step 4: Mapping of Binary 1:N relationships.***

For this step, we identified 1:N relationships between these following entity relations:

**health\_worker\_student\_relationship**

Primary key of **health\_worker** **(health\_workerID)** is transferred to **student** as **health\_workerID** in student.

**student**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| studentID | username | password | teacherID | health\_workerID | sex | age |

**edit\_grades**

Primary key of the **teacher (teacherID)** is transferred to **grades** as **teacherID** in grades.

**grades**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| studentID | gradeReportID | first\_grade | second\_grade | final\_grade | teacherID |

**edit\_vaccination\_record**

Primary key of **health\_worker (health\_workerID)** is transferred to **vaccination\_records** as **health\_workerID** in vaccination\_records.

**vaccination\_records**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| studentID | recordID | first\_vaccine | first\_vaccine\_date | second\_vaccine | second\_vaccine\_date | third\_vaccine | third\_vaccine\_date | health\_workerID |

***Step 5: Mapping of Binary M:N relationships.***

We only have one M:N relationship, between teacher and student, called teacher\_student\_relationship. We will create a new relation for this:

**teacher\_student\_relationship**

|  |  |  |
| --- | --- | --- |
| teacherID | studentID | till\_academic\_year |

Here, academic\_year would denote when this relation is no longer valid. For example, if till\_academic\_year is 2023, then this relationship is valid until 2023, and will become invalid after 2023.

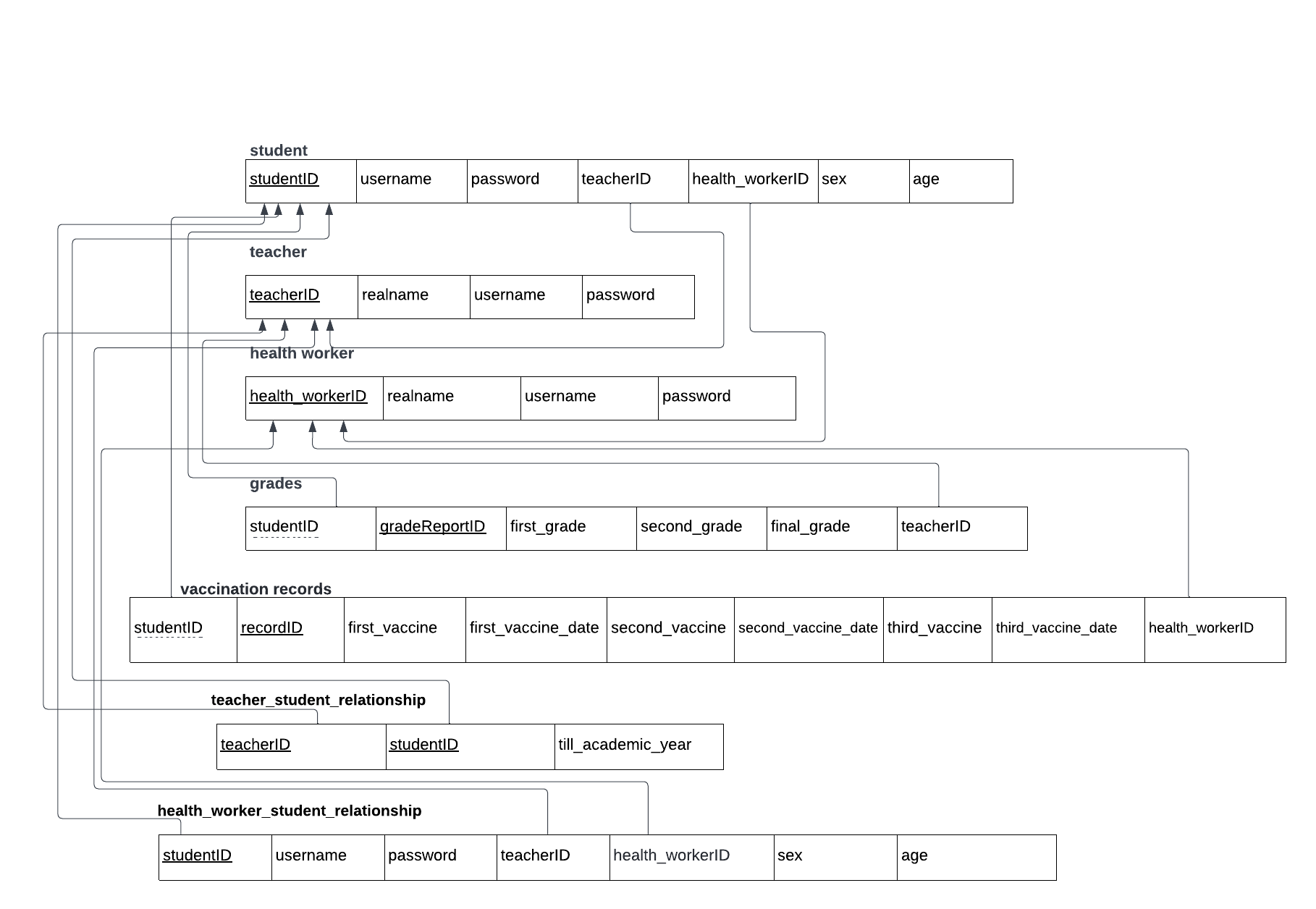
***Step 6: Mapping multi-varied attributes.***

We don’t currently have any multi-varied attributes.

***Step 7: Mapping of “N-ary” relationship types.***

We don’t have any N-ary relationships in our ER diagram.

***Step 8 and 9: Mapping summary***



|  |  |
| --- | --- |
| **Relation name** | **ER Diagram components** |
| student | E(student)+R(teacher\_student\_relationship)  +R(health\_worker\_student\_relationship) |
| teacher | E(teacher)+R(edit\_grades)+R(teacher\_student\_relationship) |
| health\_worker | E(health\_worker)+R(edit\_vaccination\_records)  +R(health\_worker\_student\_relationship) |
| grades | E(grades)+R(edit\_grades) |
| vaccination\_records | E(vaccination\_records)+R(edit\_vaccination\_record) |

**Schema of database**

*student* relation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key Type** | **Justification** | **Data Type** | **Description** | **Example** | **Default** | **Action on FK** |
| studentID | Primary key | It is an unique identifier for each student | INTEGER(20) | Student’s ID | “166300” |  |  |
| username | Unique key | It is an unique identifier for each student user | VARCHAR(30) | Student’s username to log in the system | “josh1889” |  |  |
| password | Unique key | It is an unique identifier for each student user | VARCHAR(20) | Student’s password to log in the system | “123Abc” |  |  |
| teacherID | Foreign Key | Refers to teacher.teacherID | INTEGER(20) | ID for the corresponding teacher of each student | “255660” |  | CASCADE |
| health\_workerID | Foreign key | Refers to healthworker.healthworkerID | INTEGER(20) | ID for the corresponding health worker of each student | “399980” |  | CASCADE |
| sex |  |  | VARCHAR(20) | Student’s sex | “female”, “male”, “transgender”, … | Can be set to “null” |  |
| age |  |  | INTEGER(3) | Student’s age | “17” | Can be set to “null” |  |

*teacher* relation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key Type** | **Justification** | **Data Type** | **Description** | **Example** | **Default** | **Action on FK** |
| teacherID | Primary key | It is an unique identifier for each teacher user | INTEGER(20) | Teacher’s ID | “255660” |  |  |
| realname |  |  | VARCHAR(30) | Teacher’s real name | “Josh Allen” |  |  |
| username | Unique key | It is an unique identifier for each teacher user | VARCHAR(30) | Teacher’s username to log in the system | “josh1889” |  |  |
| password | Unique key | It is an unique identifier for each teacher user | VARCHAR(20) | Teacher’s password to log in the system | “123Abc” |  |  |

*health\_worker* relation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key Type** | **Justification** | **Data Type** | **Description** | **Example** | **Default** | **Action on FK** |
| health\_workerID | Primary key | It is an unique identifier for each health worker user | INTEGER(20) | Health worker’s ID | “399980” |  |  |
| realname |  |  | VARCHAR(30) | Health worker’s real name | “David Wang” |  |  |
| username | Unique key | It is an unique identifier for each health worker user | VARCHAR(30) | Health worker’s username to log in the system | “David11” |  |  |
| password | Unique key | It is an unique identifier for each health worker user | VARCHAR(20) | Health worker’s password to log in the system | “123Abc” |  |  |

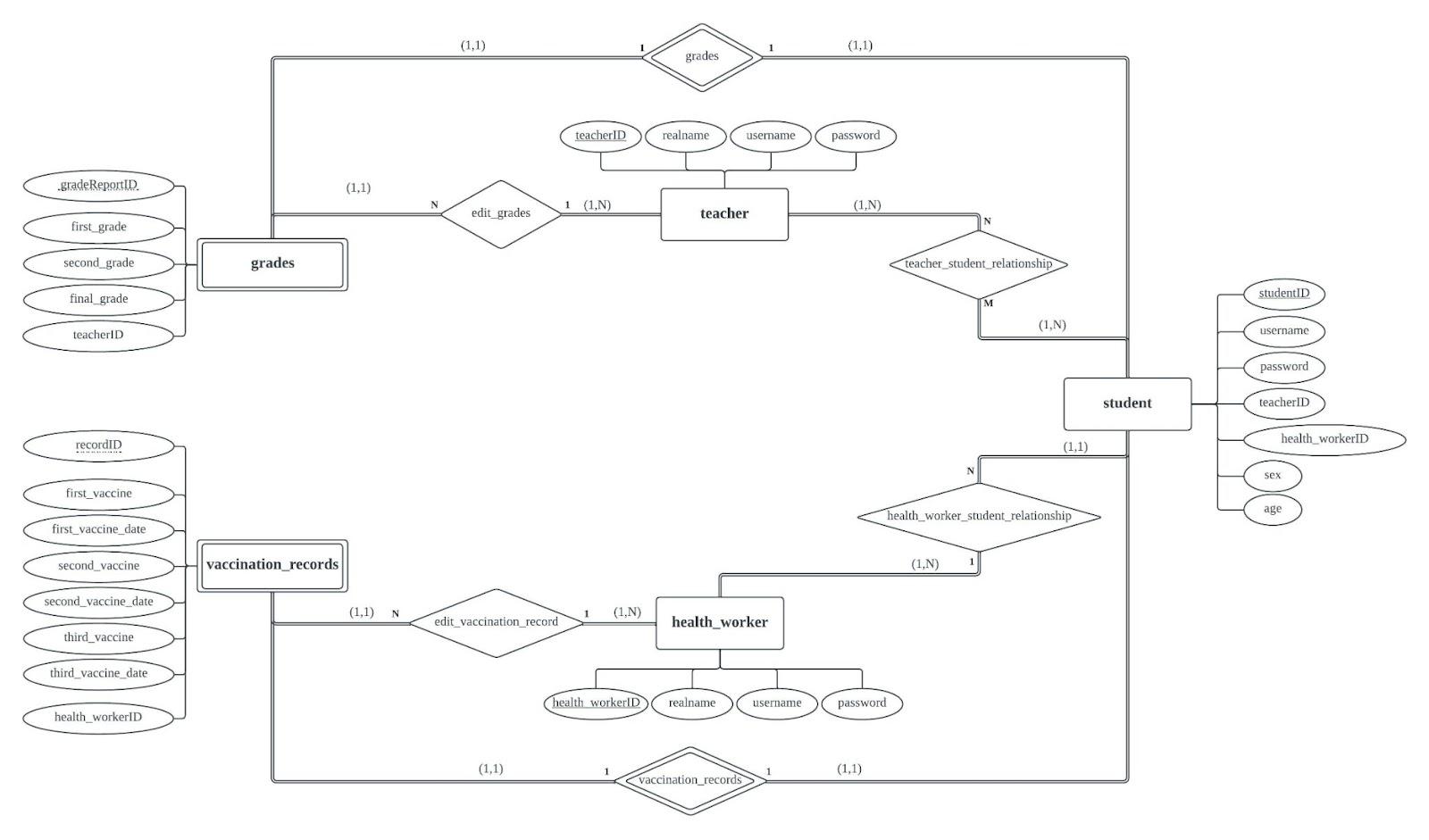
*grades* relation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key Type** | **Justification** | **Data Type** | **Description** | **Example** | **Default** | **Action on FK** |
| gradeReportID | Partial Key | Since *grades* is the weak entity of *student*, gradeReportID goes with student.studentID as the primary key of *grades* | INTEGER(10) | Student’ grade report ID | “1” |  |  |
| first\_grade |  |  | FLOAT | Student’s grade for the first exam | “10.5” | Can be set to “null” |  |
| second\_grade |  |  | FLOAT | Student’s grade for the second exam | “11.7” | Can be set to “null” |  |
| final\_grade |  |  | FLOAT | Student’s grade for the final exam | “12.8” | Can be set to “null” |  |
| teacherID | Foreign Key | Refers to teacher.teacherID | INTEGER(10) | ID for the corresponding teacher of each student | “255660” |  | CASCADE |

*vaccination\_records* relation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key Type** | **Justification** | **Data Type** | **Description** | **Example** | **Default** | **Action on FK** |
| recordID | Partial key | Since *vaccination\_records* is the weak entity of *student*, recordID goes with student.studentID as the primary key of *vaccination\_record* | INTEGER(10) | ID for student’s vaccine status | “1” |  |  |
| first\_vaccine |  |  | VARCHAR(20) | Status of student’s first vaccination including the vaccine’s name if vaccinated or “none” for not vaccinated | “Moderna” , “Pfizer” or “ None” for not vaccinated | Can be set to “null” |  |
| first\_vaccine\_date |  |  | DATE | Date to receive the first vaccine | “2022-03-02” | Can be set to “null” |  |
| second\_vaccine |  |  | VARCHAR(20) | Status of student’s second vaccination including the vaccine’s name if vaccinated or “none” for not vaccinated | “Moderna” , “Pfizer” or “ None” for not vaccinated | Can be set to “null” |  |
| second\_vaccine\_date |  |  | DATE | Date to receive the second vaccine | “2022-03-02” | Can be set to “null” |  |
| third\_vaccine |  |  | VARCHAR(20) | Status of student’s thor vaccination including the vaccine’s name if vaccinated or “none” for not vaccinated | “Moderna” , “Pfizer” or “ None” for not vaccinated | Can be set to “null” |  |
| third\_vaccine\_date |  |  | DATE | Date to receive the third vaccine | “2022-03-02” | Can be set to “null” |  |
| health\_workerID | Foreign Key | Refers to healthwork.healthworkID | INTEGER(10) | ID for the corresponding health worker of each student | “399980” |  | CASCADE |

**2.3** **BCNF Normalization**



**Figure 2.** ER Diagram

As seen in Figure 2, we have five relational schemas in this database:

1. (*studentID, username, realname, password, teacherID, health\_workerID, sex, age*)
2. (*teacherID, realname, username, password*)
3. (*health\_workerID, realname, username, password*)
4. (*studentID, gradeReportID, first\_grade, second\_grade, final\_grade, teacherID*)
5. (*studentID, recordID, first\_vaccine, first\_vaccine\_date, second\_vaccine, second\_vaccine\_date, third\_vaccine, third\_vaccine\_date, health\_workerID*)

The key for each schema is underlined. Because relational schema 4 is a weak entity, so the primary key is the combination of {*studentID*, *gradeReportID*} in which *studentID* is the primary key of the owner entity **student** it is related to, and the partial key of this weak entity is *gradeReportID*. Similarly, relational schema 5 is also a weak entity. The primary key is the combination of {*studentID*, *reportID*} in which *studentID*is the primary key of its owner entity **student**, and the partial key of this weak entity is *reportID*.

**BCNF Normalization for**

We come up with the candidate keys and functional dependencies based on the definitions of key and functional dependency, along with each attributes’ real-world constraints. Based on the definition of key, we selected all the attributes that can be used to uniquely identify the attributes in this particular relation. Therefore, *studentID* is the primary key, *username* and *password* are the secondary keys. According to the definition of the functional dependencies, X→Y holds if whenever two tuples have the same value for X, they must have the same value for Y (Zhupa, 2022). So each candidate key functionally determines all attributes in . The results are shown as below.

(*studentID, username, password, realname, teacherID, health\_workerID, sex, age*)

Prime attributes:

*studentID* (primary key)*, username* (candidate key)*, password* (candidate key)

Non prime attributes:

*teacherID* (foreign key)*, health\_workerID* (foreign key), *realname, sex*, *age*

Original FDs for :

FD1: *studentID →* {*username, password, teacherID, health\_workerID, realname, sex, age*}

FD2: *username →* {*studentID, password, teacherID, health\_workerID, realname, sex, age*}

FD3: *password →* {*studentID, username, teacherID, health\_workerID, realname, sex, age*}

To satisfy BCNF normalization, first we checked whether all the functional dependencies are in minimal cover. We found our original set of functional dependencies is not minimal because there are multiple attributes on the right hand side (RHS). So we transformed the original functional dependencies into canonical form, and checked whether the transformed form is not redundant and without extraneous attributes. The results can be seen below.

New FDs for :

FD1: *studentID → username*

FD2: *studentID → password*

FD3: *studentID → teacherID*

FD4: *studentID → health\_workerID*

FD5: *studentID → sex*

FD6: *studentID → age*

FD7: *studentID → realname*

FD8: *username → studentID*

FD9: *username → password*

FD10: *username → teacherID*

FD11: *username → health\_workerID*

FD12: *username → sex*

FD13: *username → age*

FD14: *username → realname*

FD15: *password → studentID*

FD16: *password → username*

FD17: *password → teacherID*

FD18: *password → health\_workerID*

FD19: *password → sex*

FD20: *password → age*

FD21: *password → realname*

Subsequently, we evaluated whether the new set of functional dependencies satisfy either of conditions: a) X→Y is a trivial functional dependency; or b) X is a superkey for schema (Zhupa, 2022). We found that in all cases from the new functional dependencies, X is a superkey for , and there are no violations. Therefore, we concluded that all the resultant relations from is in the Boyce-Codd normal form.

**BCNF Normalization for**

Based on the definition of key and the meaning of each attribute from the real world, we selected *teacherID*, *username* and *password* as keys which can uniquely identify each record.

(*teacherID, realname, username, password*)

Prime attributes:

*teacherID* (primary key)*, username* (candidate key)*, password (candidate key)*

Non prime attribute:

*realname*

According to the definition of the functional dependencies, X→Y holds if whenever two tuples have the same value for X, they must have the same value for Y (Zhupa, 2022). Each candidate key functionally determines all attributes in as demonstrated below.

Original FDs for :

FD1: *teacherID →* {*username, password, realname*}

FD2: *username →* {*teacherID, password, realname*}

FD3: *password →* {*teacherID, username, realname*}

To check whether the original functional dependencies satisfy BCNF normalization, first we evaluate whether all the functional dependencies are in minimal cover. We found our original set of functional dependencies can be further transformed into canonical form to meet the requirement of minimal cover. It means that there will be only one attribute on the right hand side (RHS). The results shown below are in canonical form with no redundancy or extraneous attributes.

New FDs for :

FD1: *teacherID → username*

FD2: *teacherID → password*

FD3: *teacherID → realname*

FD4: *username → teacherID*

FD5: *username → password*

FD6: *username → realname*

FD7: *password → teacherID*

FD8: *password → username*

FD9: *password → realname*

Then, we examined whether the new set of functional dependencies meet either of conditions: a) X→Y is a trivial functional dependency; or b) X is a superkey for schema (Zhupa, 2022). We found all the Xs in the above 9 functional dependencies are superkey and there is no violation. Therefore, we concluded that all the resultant relations are in the Boyce-Codd normal form.

**BCNF Normalization for**

We chose *health\_workerID*, *username* and *password* as keys since they can uniquely identify each record.

(*health\_workerID, realname, username, password*)

Prime attributes:

*health\_workerID* (primary key)*, username* (candidate key)*, password (candidate key)*

Non prime attribute:

*realname*

Since each candidate key functionally determines all attributes in based on the definition of the functional dependencies, the original FDs are as follows.

Original FDs for :

FD1: *health\_workerID →* {*username, password, realname*}

FD2: *username →* {*health\_workerID, password, realname*}

FD3: *password →* {*health\_workerID, username, realname*}

To satisfy BCNF normalization, first we converted the original functional dependencies into canonical form with no redundancy or extraneous attributes. In order to do so, we made sure each functional dependency has only one attribute on its right-hand side.

New FDs for :

FD1: *health\_workerID → username*

FD2: *health\_workerID → password*

FD3: *health\_workerID → realname*

FD4: *username → health\_workerID*

FD5: *username → password*

FD6: *username → realname*

FD7: *password → health\_workerID*

FD8: *password → username*

FD9: *password → realname*

Second, we checked whether the new version of functional dependencies meet either of conditions: a) X→Y is a trivial functional dependency; or b) X is a superkey for schema (Zhupa, 2022). We found each X in the above 9 functional dependencies is a super key, so there is no violation of Boyce-Codd normal form. Therefore, we concluded that all the resultant relations in schema are in Boyce-Codd normal form.

**BCNF Normalization for**

According to the definition of key and the meaning of each attribute from the mini-world, we identified *studentID* and *gradeReportID* together as the whole key, *teacherID* as the foreign key.

(*studentID, gradeReportID, first\_grade, second\_grade, final\_grade, teacherID*)

Prime attributes:

*studentID* combined with *gradeReportID* (primary key)

Non prime attributes:

*first\_grade*, *second\_grade*, *final\_grade*, *teacherID* (foreign key)

We created the original version of functional dependency based on the definition of the functional dependencies (i.e., X→Y holds if whenever two tuples have the same value for X, they must have the same value for Y, Zhupa, 2022). Here, the key functionally determines all attributes in .

Original FD for :

{*studentID, gradeReportID*} *→* {*first\_grade, second\_grade, final\_grade, teacherID*}

To check whether the original functional dependency satisfies BCNF normalization, we first examined whether it is in minimal cover. Since the original version is not in canonical form, we transformed it so that only one attribute exists on the right hand side. The new set is in canonical form with no redundancy or extraneous attributes.

New FDs for :

FD1: {*studentID, gradeReportID*} *→ first\_grade*

FD2: {*studentID, gradeReportID*}  *→ second\_grade,*

FD3: {*studentID, gradeReportID*} *→ final\_grade*

FD4: {*studentID, gradeReportID*} *→ teacherID*

Next, we examined whether the new set of functional dependencies meet either of the two conditions: a) X→Y is a trivial functional dependency; or b) X is a superkey for schema (Zhupa, 2022). We found whenever a X→Y holds in , X is a super key, so there is no violation in Boyce-Codd normal form and thus we can conclude it is in the Boyce-Codd normal form.

**BCNF Normalization for**

According to the definition of key and the meaning of each attribute from the real world, we decided to choose *studentID* combined with *recordID* as the whole key which can uniquely identify each record.

(*studentID, recordID, first\_vaccine, first\_vaccine\_date, second\_vaccine, second\_vaccine\_date, third\_vaccine, third\_vaccine\_date, health\_workerID*)

Prime attributes:

*studentID* combined with *recordID* (primary key)

Non prime attribute:

*first\_vaccine, first\_vaccine\_date, second\_vaccine, second\_vaccine\_date, third\_vaccine, third\_vaccine\_date, health\_workerID* (foreign key)

We identified one functional dependency based on the definition of the functional dependencies which states that X→Y holds if whenever two tuples have the same value for X, they must have the same value for Y (Zhupa, 2022). The whole key functionally determines all attributes in as shown below.

Original FD for :

{*studentID, recordID*} *→* {*first\_vaccine, first\_vaccine\_date, second\_vaccine, second\_vaccine\_date, third\_vaccine, third\_vaccine\_date, health\_workerID*}

To check whether this functional dependency satisfies BCNF normalization, we examined whether it is in minimal cover. Since the right-hand side has multiple attributes, we transformed it into canonical form with no redundancy or extraneous attributes, and came up with 7 new functional dependencies.

New FDs for :

FD1: {*studentID, recordID*} *→ first\_vaccine*

FD2: {*studentID, recordID*} *→ first\_vaccine\_date*

FD3: {*studentID, recordID*} *→ second\_vaccine*

FD4: {*studentID, recordID*} *→ second\_vaccine\_date*

FD5: {*studentID, recordID*} *→ third\_vaccine*

FD6: {*studentID, recordID*} *→ third\_vaccine\_date*

FD7: {*studentID, recordID*} *→ health\_workerID*

Secondly, we checked whether the new set of functional dependencies meet either of conditions: a) X→Y is a trivial functional dependency; or b) X is a superkey for schema (Zhupa, 2022). We found all the Xs on the left-hand side are superkeys and there is no violation. Thus, we concluded that all the resultant relations are in the Boyce-Codd normal form.

Now, all the relational schemas have been transformed into Boyce-Codd normal form.

**3 Full-stack web app deployment**

**3.1 Web Front End Development**

For this project, we start by defining the functionalities of the application.

It’s obvious that we would need a ***login screen***. For the login screen, the application would transmit the username and hashed password to the backend, and the backend would authenticate.

After that, the student would enter the ***student dashboard***, where the student could easily view all their grades, information and vaccination records. For teachers and health workers, they would enter the ***teachers dashboard*** and ***health-workers dashboard***, where they could see all the students they are currently supervising.

For students, everything they need to see is on their dashboard. For teachers and health-workers, they could select studentID and student names to go into the ***grade-edit page*** and ***vaccination-edit page*** to edit the information.

In total, we would need 6 different front end views for this application. To develop those views, we consider the common usage scenarios: how would a user use this application? In a modern environment, it’s crucial for our design to be mobile-first, since we anticipate that most of our users will use our application on mobile devices.

Because of the scope of this project, we didn’t opt for a fully separated front-end back-end structure, since there’s not likely to be room for further development.

10 queries are needed, as follow:

1. Login
   1. Given username and password, return usertype and userID (studentID, teacherID or health\_workerID)

Possible usertype: “teacher”, “student”, “health\_worker” and “none”, where “none” represents that cannot find this user in the database (no match.)

1. Dashboards

(Selects)

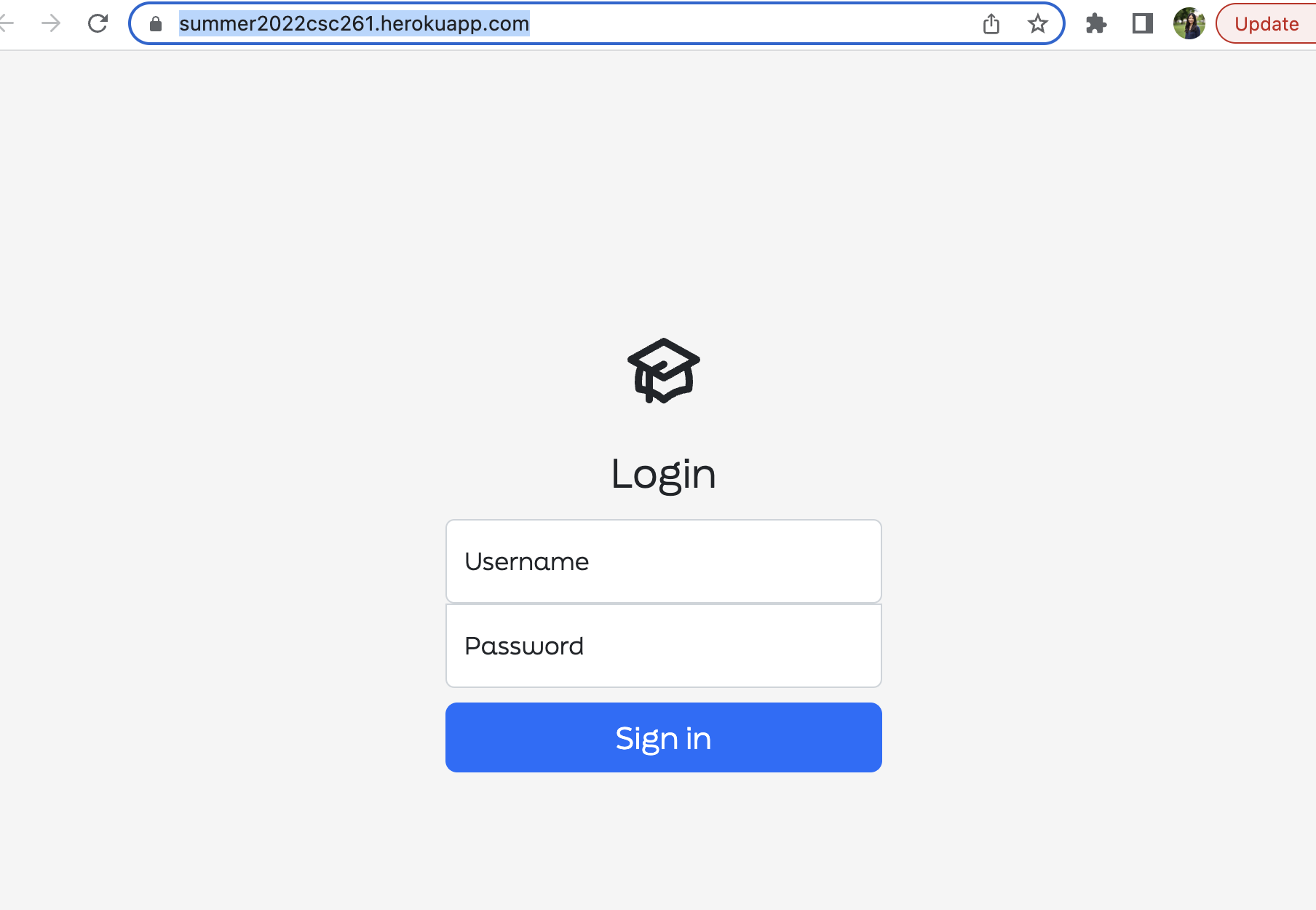
* 1. Given studentID, return its associated recordID and gradeReportID, associated teacherID, associated health\_workerID, sex, age and realname.’
  2. Given teacherID, return teacher’s real name.
  3. Given healthworkerID, return healthworker’s real name.
  4. Given teacherID, return all associated studentID.
  5. Given healthworkerID, return all associated studentID.
  6. Given studentID, return all elements in that grade report.
  7. Given studentID, return all elements in that vaccination record.

(Updates)

* 1. Given gradeReportID and all elements in that grade report, update that grade report.
  2. Given recordID and all elements in that recordID, update that record.

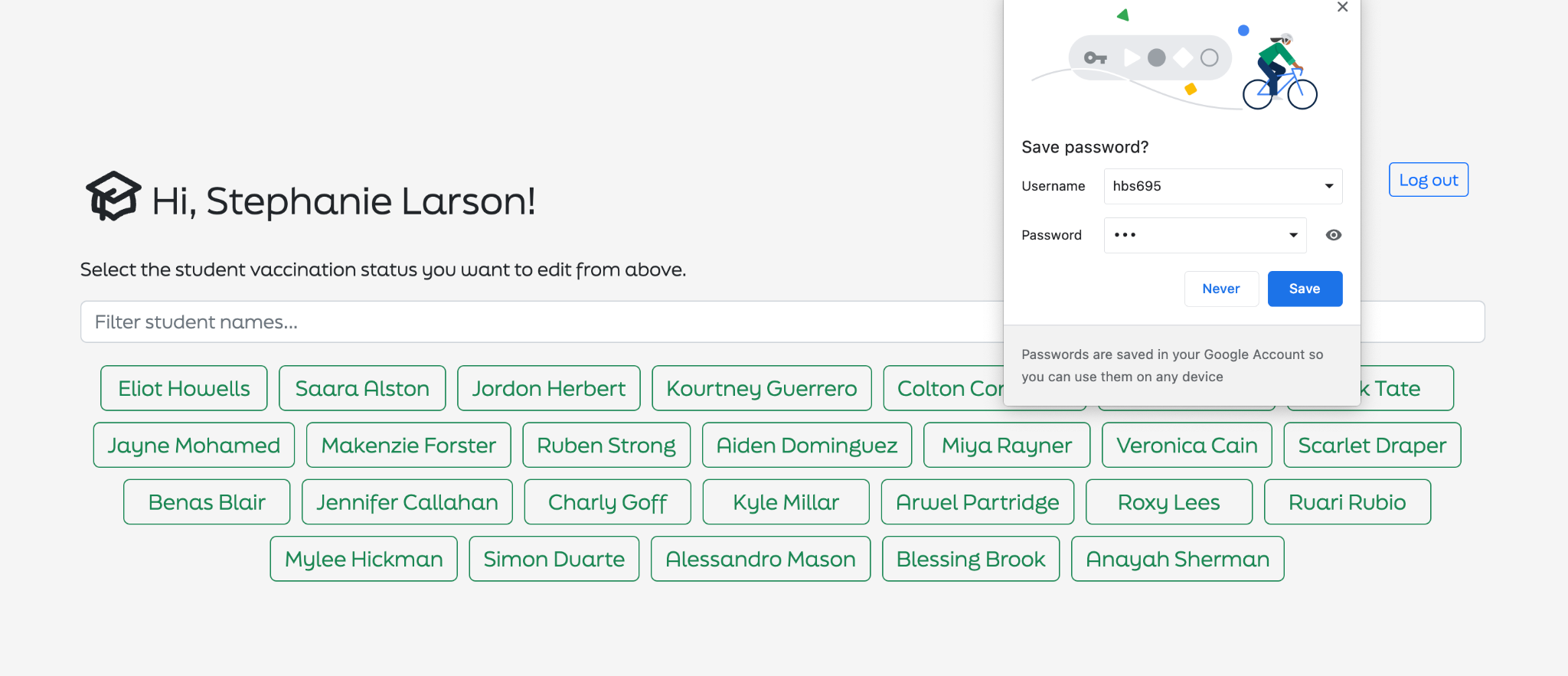
For the purpose of this milestone, we assume that we don’t need to delete or insert new records, as the database would be created in advance. Before submitting the SQL query, PHP backend would make sure that all inputs are sanitized to stop potential malicious input.

1. Sample web pages and descriptions



**Figure 3.** Login page.

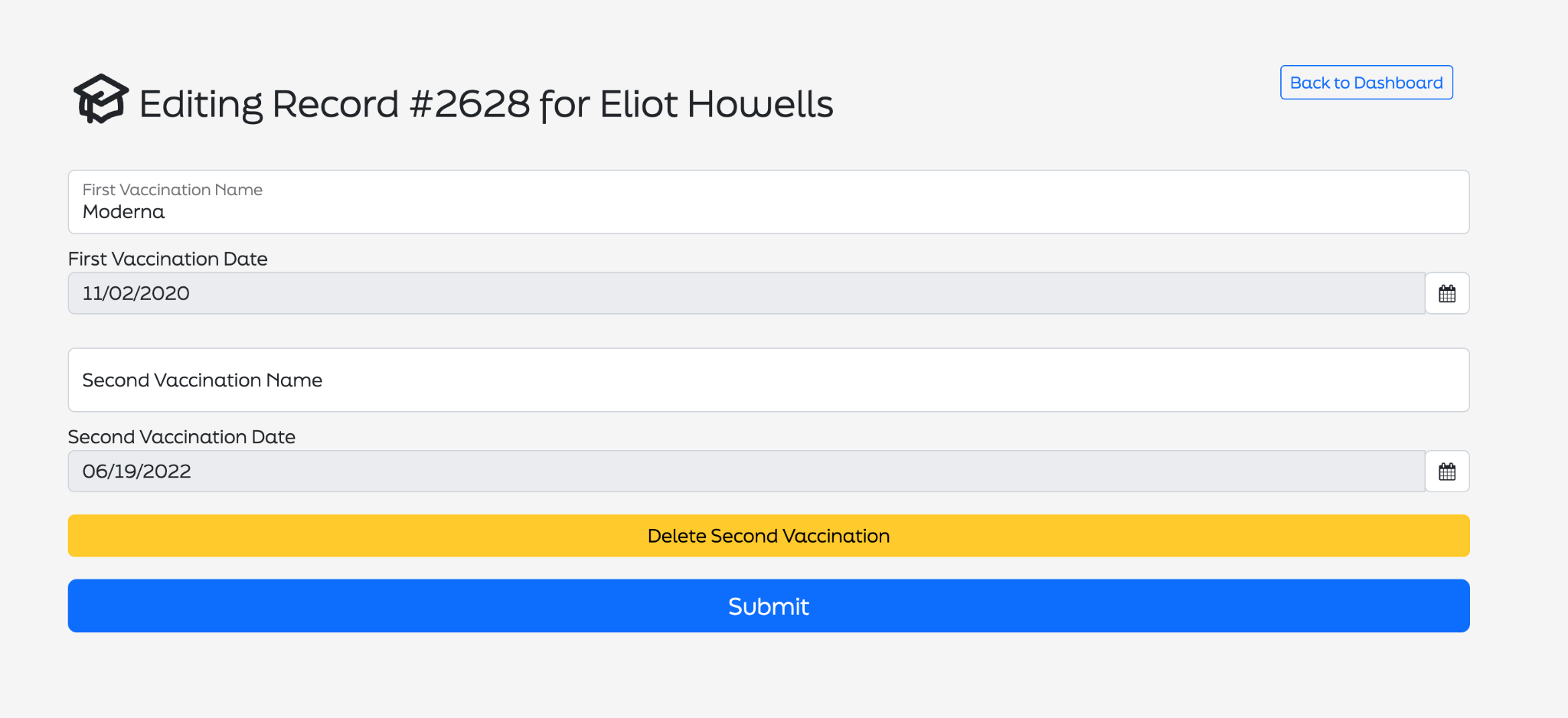
Users should enter both username and password to sign in the system.



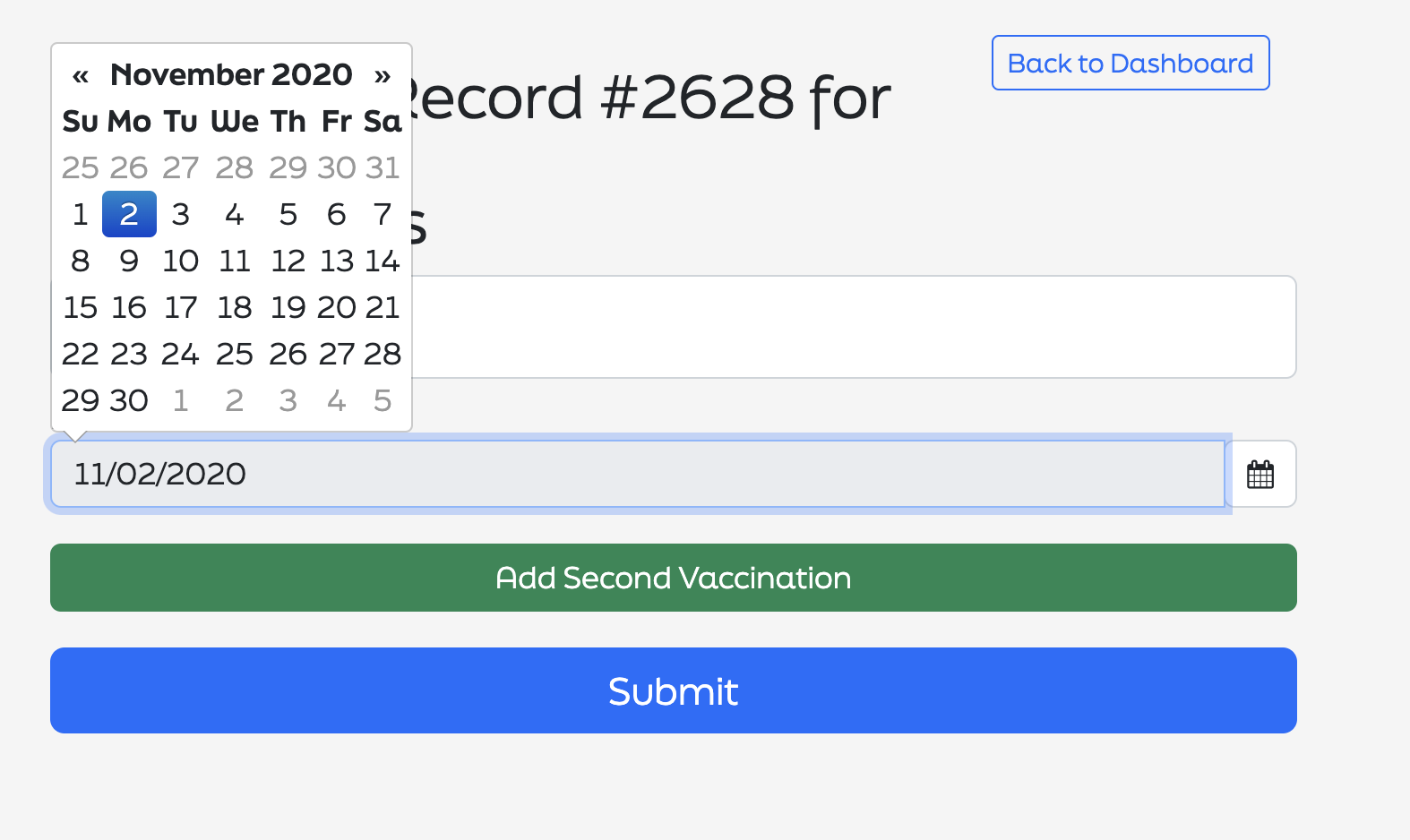
**Figure 4.** Name selection front page after logging in as a health-worker

Users can save their username and password for future convenience. Users can either type in the student's name or select one of the green bars. Users can also log out when necessary.

**Figure 5.** Vaccination editing page

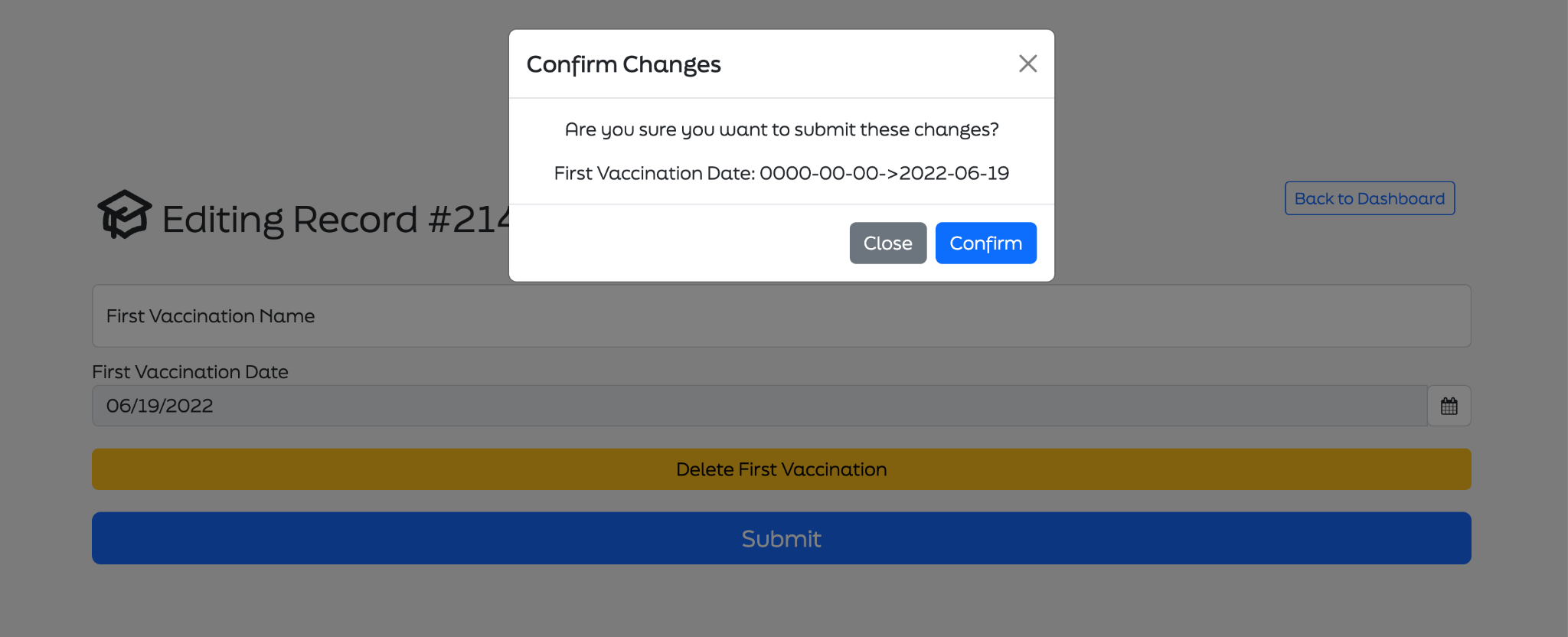


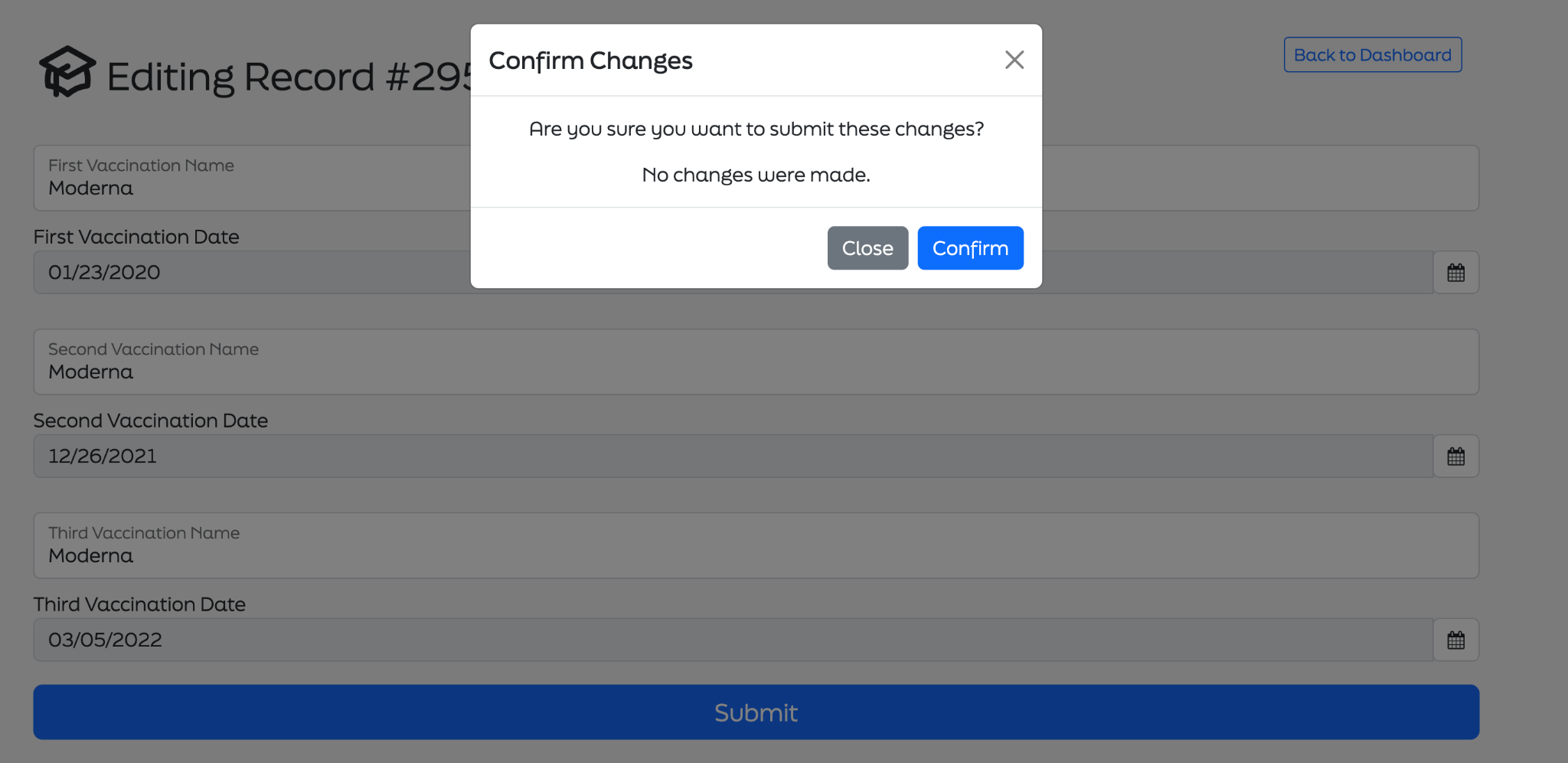
Imagine a user selects the vaccination record for a student named Eliot Howells, he/she can update this student’s first or second vaccination’s name and date, then submit the form. Users can always click “Back to Dashboard” and return to the previous front page.



**Figure 6.** Vaccination editing date page

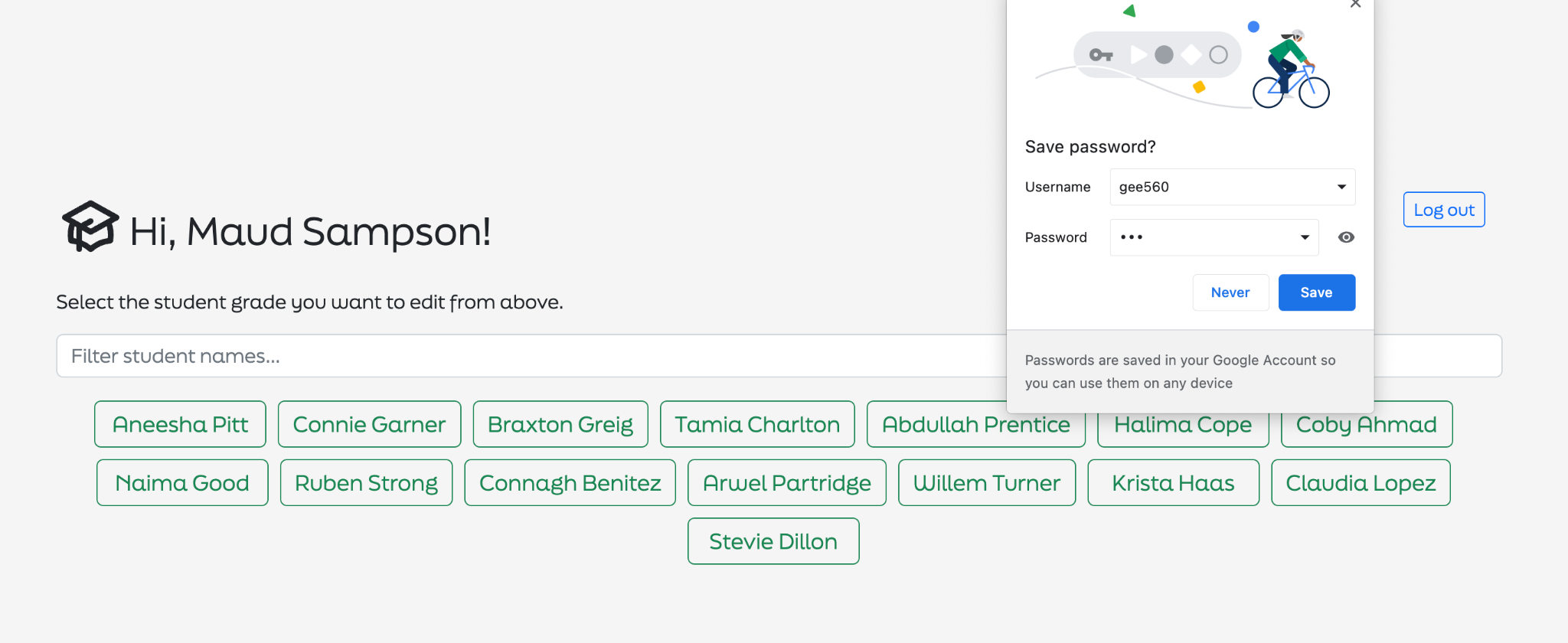
A calendar will appear when users select the date.





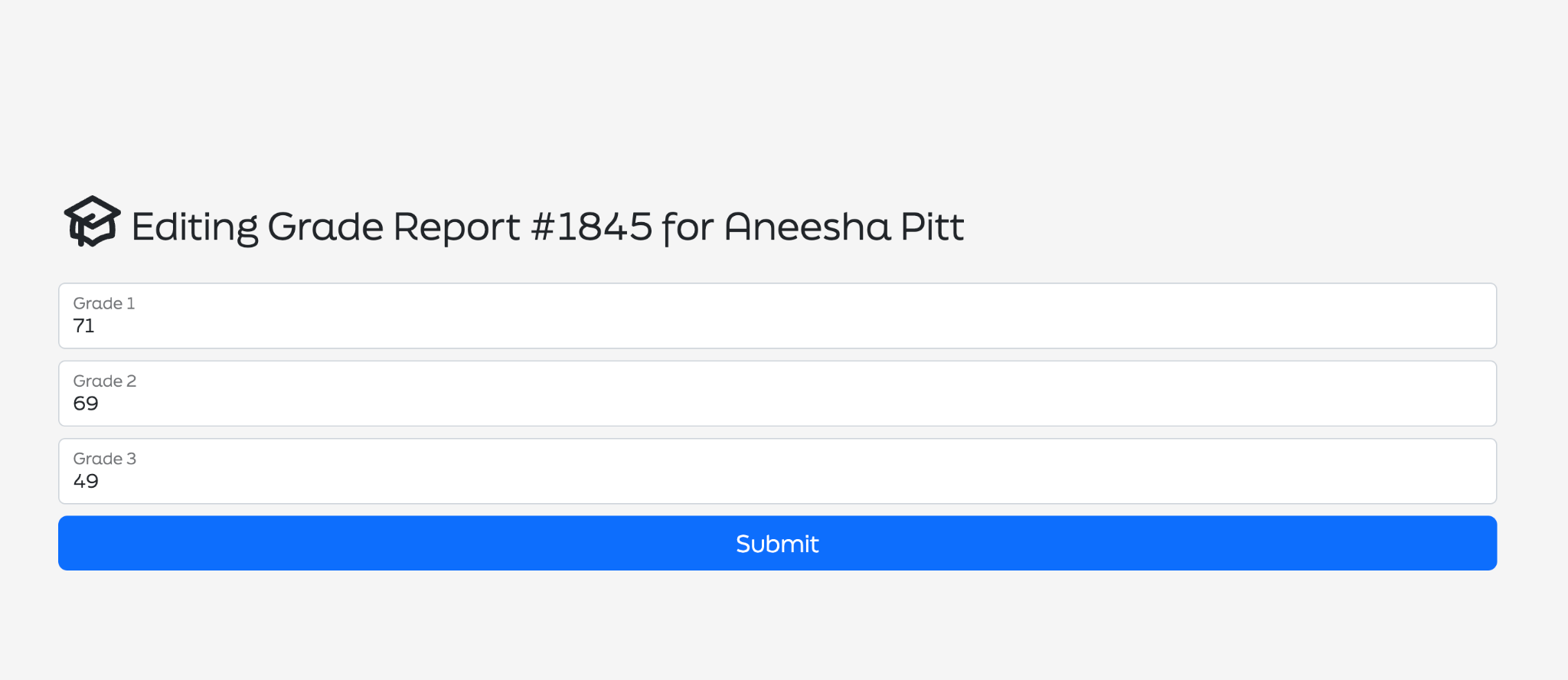
**Figure 7 and Figure 8.** Vaccination confirmation page

The system will ask the users to confirm changes when they update the information on the vaccination page. Users can also click “Close” if they decide to make no changes at this time.



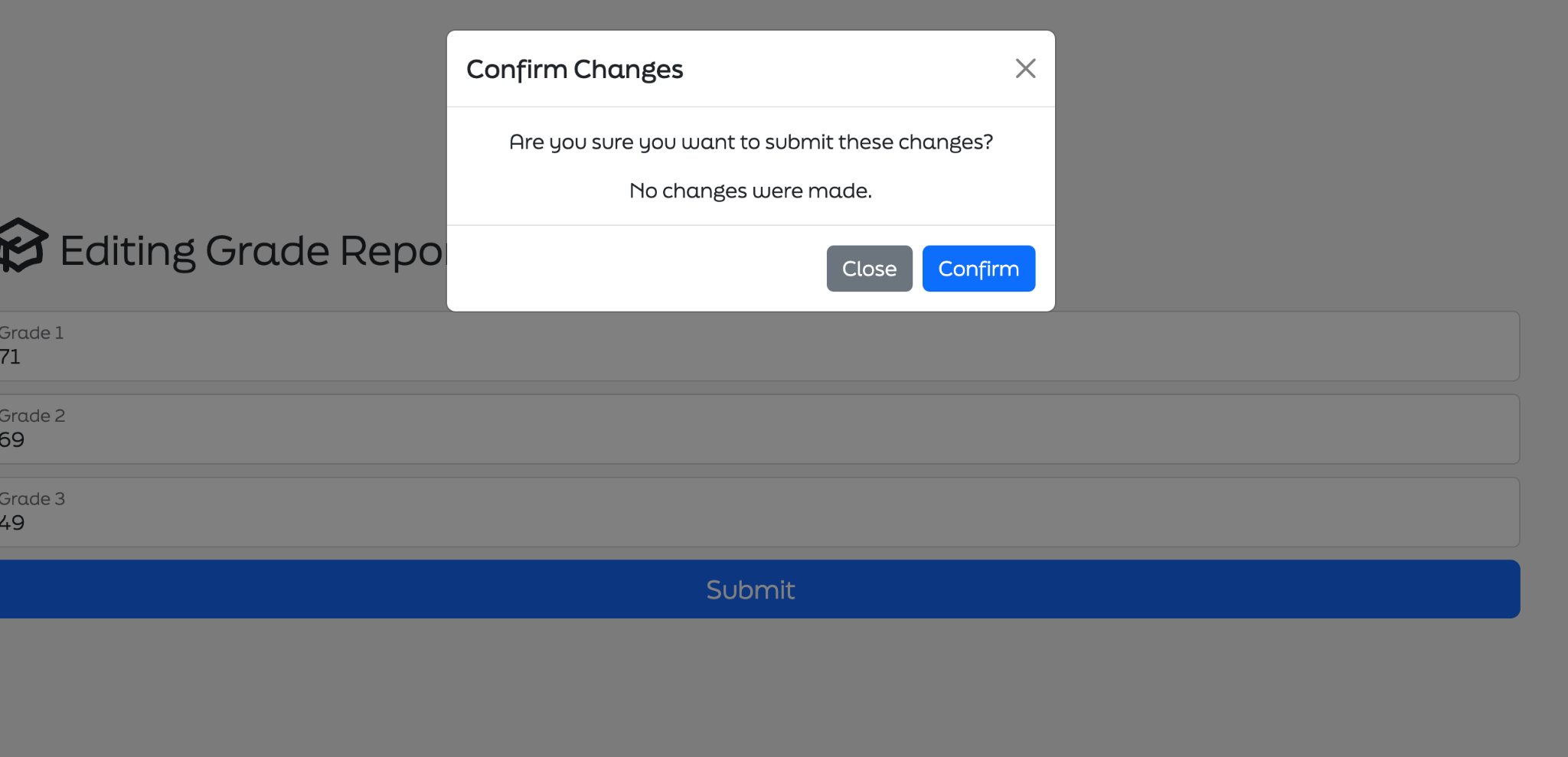
**Figure 9.** Name selection page after logging in as a teacher

Users can save their username and password for future convenience. Users can either type in the student's name or select one of the green bars.

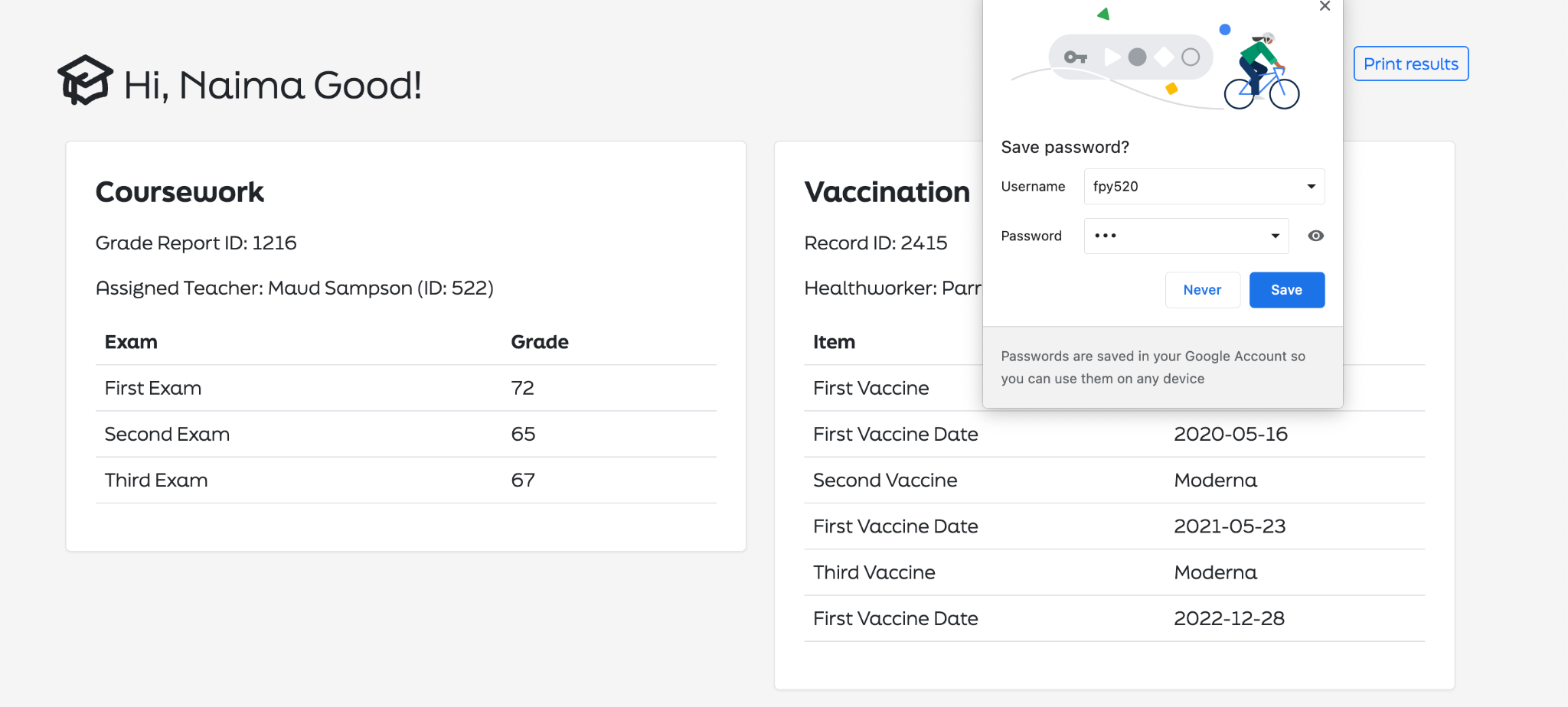


**Figure 10.** Grade editing page.

Users can update Grade 1, Grade 2 and Grade 3 during three different periods of schooling. Users can click “Submit” when they complete this step.

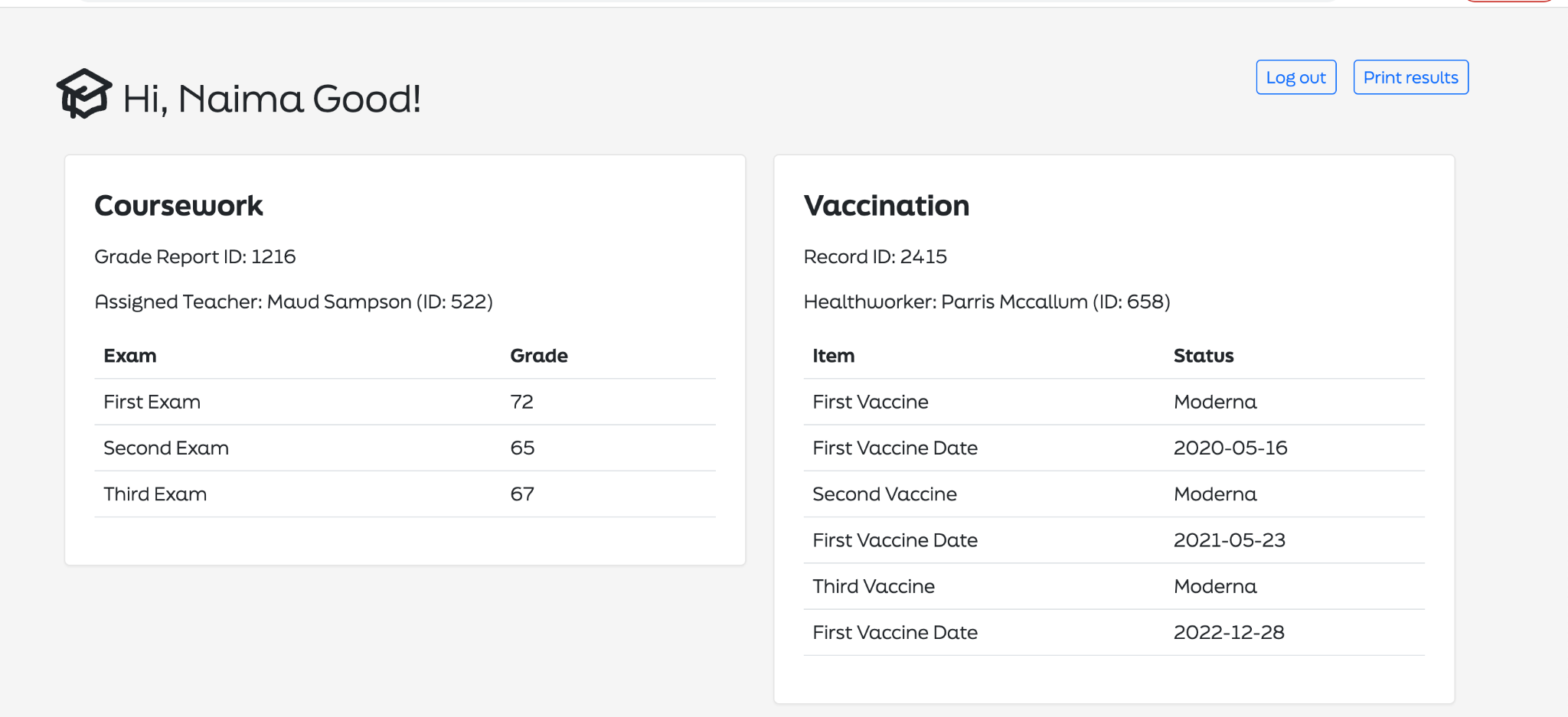


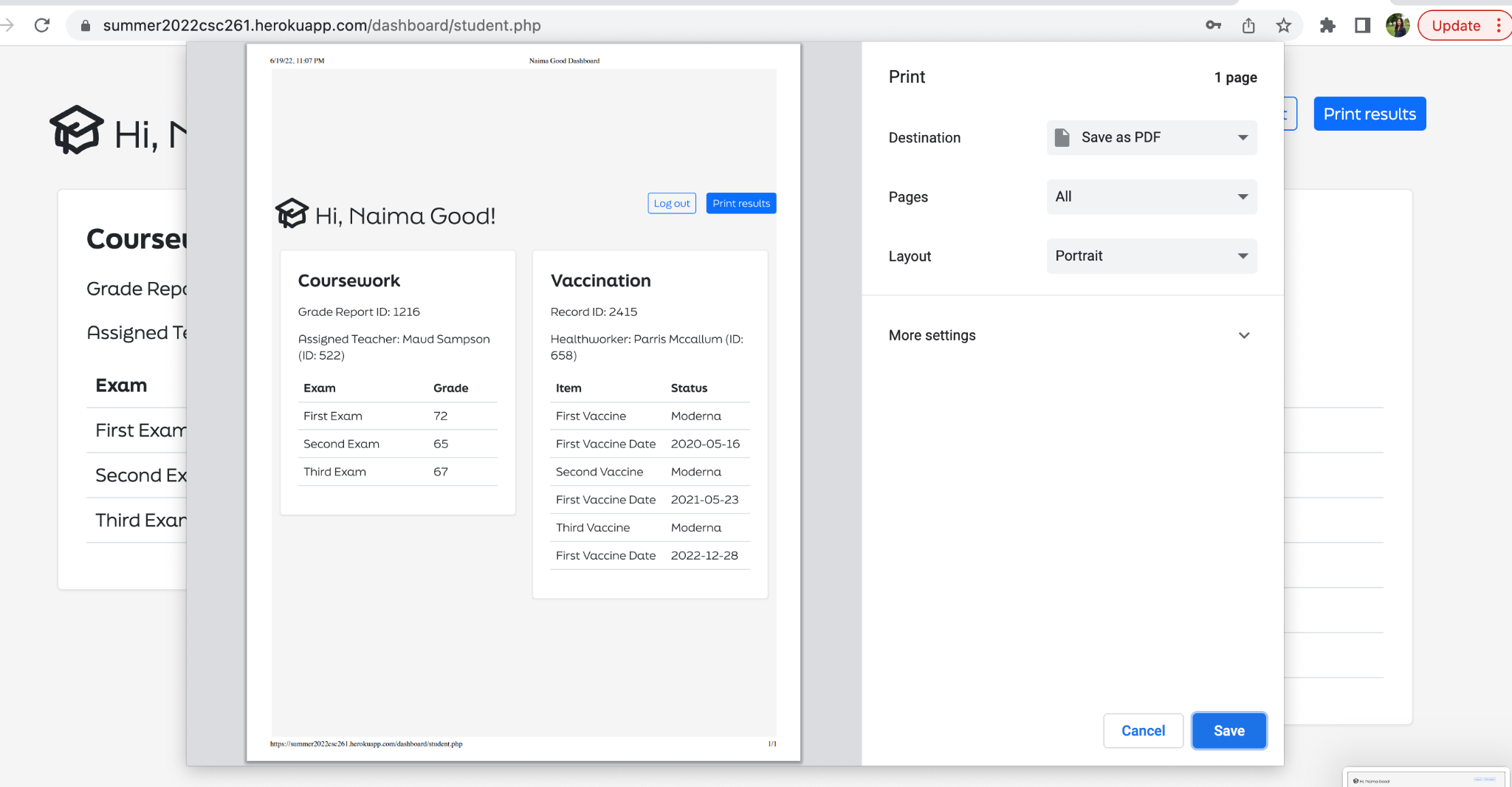
**Figure 11.** Grade confirmation page

The system will ask users to confirm changes when they complete updating the information on the grade report page. Users can also click “Close” if they decide to make no changes at this time.

**Figure 12**. Dashboard after logging in as a student

Users can save their username and password for future convenience. Student users are only able to browse their academic achievements on coursework, and vaccination related information. However, they are not allowed to edit the information.





**Figure 13 and 14**. Dashboard’s additional functions after logging in as a student

Users can choose to log out the webpage by clicking “Log out”. They can also choose to save the results as PDF or print them out by clicking on the “Print results” button.

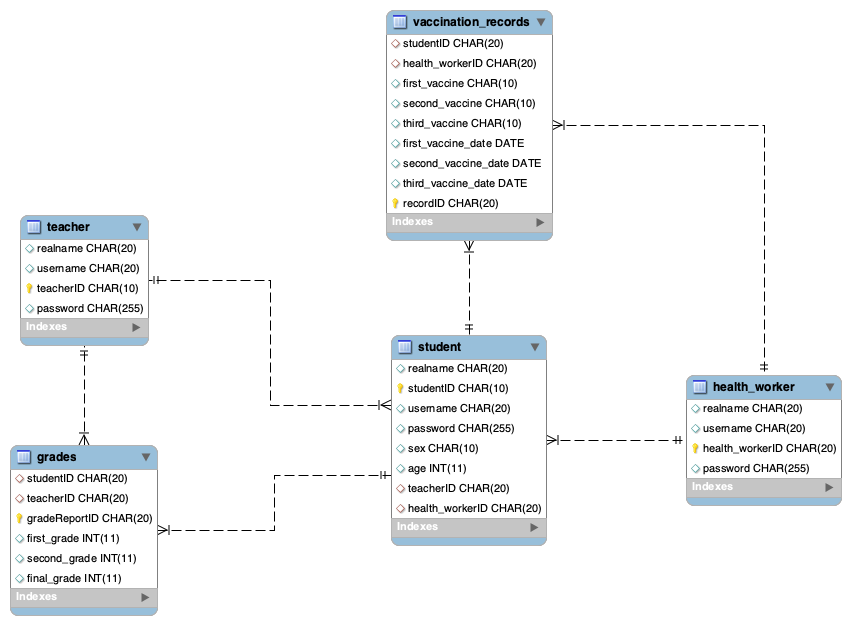
The front end is built using the Bootstrap framework, with jQuery to help with additional scripts. Web font is loaded in via Adobe Fonts service, and the logo is an SVG hardcoded into the page. Animations are coded using jQuery Animation API. The front-end is mobile-friendly and PWA-ready, so users could add it to their phone app drawer as a mobile application.

When considering which functionalities to implement in the front end, and which in the back end, we decided to separate based on data. When an operation involves dealing with SQL queries or session data, or when the data is too sensitive, the logic resides in PHP where the user would have no way of altering (login/logout, getting/updating data from SQL, etc.;) When a particular part of logic is more about how data is presented, we write it in JavaScript (vaccine date/name/display parsing, filtering student names, for instance.) This architecture also helps with future expansion, as the front end and back end could be separated easily once needed.

**3.2 SQL queries to create and load relations**

To test the function of our database, five “.csv” files are generated by the data simulation process in R. The five files are the tables in the database, including “student”, “teacher”, “health\_worker”, “grades” and “vaccination\_records”. For each kind of user, the real name, ID, username and password are randomly generated. To be specific, IDs are three-digit random numbers and the username contains three random letters and three random numbers. The password is designed in the format of “three letters” + “csc261”. Following best practices in database security when dealing with login information, we salted and hashed the password in order to protect ourselves against rainbow table attacks. The password stored in our database is calculated using md5(password\_pre + ‘csc261’). The vaccination records are set to be with 0.8 probability to be vaccinated for each shot.

***Create.sql*** and ***load.sql*** files are developed by MySQLWorkbench to create the database. According to the relations defined in previous work, five tables are created to store the information about the three types of users and the grades and vaccination records of the students. Meanwhile, primary keys and the foreign keys are defined after loading the data. The ER diagram by reversing engineering the database is in Figure 13.



**Figure 15.** EER diagram of the database

**3.3 PHP Backend**

The website is hosted on heroku, at <https://summer2022csc261.herokuapp.com/>. The database for this project is ClearDB, an add-on provided by heroku. All SQL queries needed are stored in ***sql-queries.php*** and called at the beginning of each page. It exposes functions as APIs for other parts of the PHP program to call. We only used wildcard queries (SELECT \*) when absolutely necessary to prevent causing too much workload to the server.

Users are expected to login at ***index.php***, then get redirected to ***dashboard/redirect.php***, where the user’s identity would be checked. Then, the user would be redirected to its respective dashboard, where based on the user's permission, the user could perform tasks. Users could also log out at ***logout.php***.

The login functionality is implemented by using PHP session APIs. Every time a user visits one particular page, the session status would be checked again to ensure that the user is in fact logged in. When a user tries to login, the input password would be computed using md5(input\_password + ‘csc261’), then compared against the stored password to see if such a user exists.

In order to optimize PHP performance, we used JSON when an array needs to be returned instead of XML. This also helped us send data directly to JavaScript for on-page processing. Also, to reduce server load, all resources are loaded via CDN whenever possible. HTTPS security comes by default with Heroku deployment. The end result is a site that could work perfectly on the limitations of the heroku free tier virtual server.

**Appendix A: Database Examples**

***Users***

**PK** userID (int) example: 1

username(string) example: “josh1889”

password(string) example: “1234567890”

type(string) example: “parent”, “teacher”, “healthworker”, “administrator”

***Student Info***

**PK** studentID (string) example: “166300”

userID (int) example: 1

sex (character-string) example: “F” = female, “M” = male

age (int) example: “17”

live in (character-string) example: “U” = urban, “R” = rural

guardian (character-string) example: “mother”, “father”

reason to choose this school (character-string)example: “course”, “home”,

“reputation”, “other”

class absences (int) example: 0, 6, 10

family educational support (boolean) example: “no”, “yes”

extra educational support (boolean) example: “no”, “yes”

extra paid class (boolean) example: “no”, “yes”

extra curricular activities (boolean) example: “no”, “yes”

attended nursery school (boolean) example: “no”, “yes”

wants to take higher education (boolean) example: “no”, “yes”

has internet access at home (boolean) example: “no”, “yes”

***Teacher Info***

**PK** userID (int) example: 1

name(string) example: “Josh Allen”

subject(string) example: “math” or “portuguese”

***Health Workers Info***

**PK** userID (int) example: 1

name(string) example: “Josh Allen”

***Math Scores***

**PK** studentID (string) example: “166300”

firstGrade (int, between 0 and 20) example: 10

secondGrade (int, between 0 and 20) example: 10

finalGrade (int, between 0 and 20) example: 10

lastUpdated (datetime)

***Portuguese Scores***

**PK** studentID (string) example: “166300”

firstGrade (int, between 0 and 20) example: 10

secondGrade (int, between 0 and 20) example: 10

finalGrade (int, between 0 and 20) example: 10

lastUpdated (datetime)

***Vaccination Status***

**PK** studentID (string) example: “166300”

firstVaccine (string) example: “Moderna” or “” (for not vaccinated)

firstVaccineDate (datetime)

secondVaccine (string) example: “Moderna” or “” (for not vaccinated)

secondVaccineDate (datetime)

boosterVaccine (string) example: “Moderna” or “” (for not vaccinated)

boosterVaccineDate (datetime)

These three are linked to the student info database.