Assignment 2

Topicus Team 1

Umair Aamir Mirza (s2928558)

Martin Demirev (s2965046)

Condu Alexandru-Stefan (s2769549)

Narendra Setty (s2944200)

Alexandru Lungu(s3006301)

Teodor Pintilie (s2920344)

Introduction:

This report contains the progress related to the project in relation to the second SPRINT review. The contents of this report include:

- Use Case Diagrams.
- UML Class Diagram (provided as a hyperlink due to the size being too large to display).
- SQL Database Schema.

The Use Case diagrams were used to model the user stories into one space where the system requirements could be determined with ease. They enable the development of the system with the abstraction from technical details, rather, a stronger focus on exactly how a user will interact with the system.

The UML class diagram is to show what objects exist in the system, and how they can be best modeled to facilitate their interactions. Such a diagram enables the development of efficient code. In the case of our project, the UML class diagram was almost identical to how the database was structured as well (apart from the database containing the constraints as well). Some entities, such as User Requests and Server Responses (seen in the Payload folder of the project) were not included on the diagram as they are reserved for inter-communication between different parts of the system.

Finally, the SQL database schema is provided as well. It contains the necessary constraints for adding values, but the check statements were omitted from the schema itself to facilitate the insertion of testing data. The check

statements are defined already, which will facilitate their insertion, but were not added for now, for the convenience of testing.

Use Case Diagrams

The following use cases have been extracted from the Functional Requirements in the form of User Stories of the project.

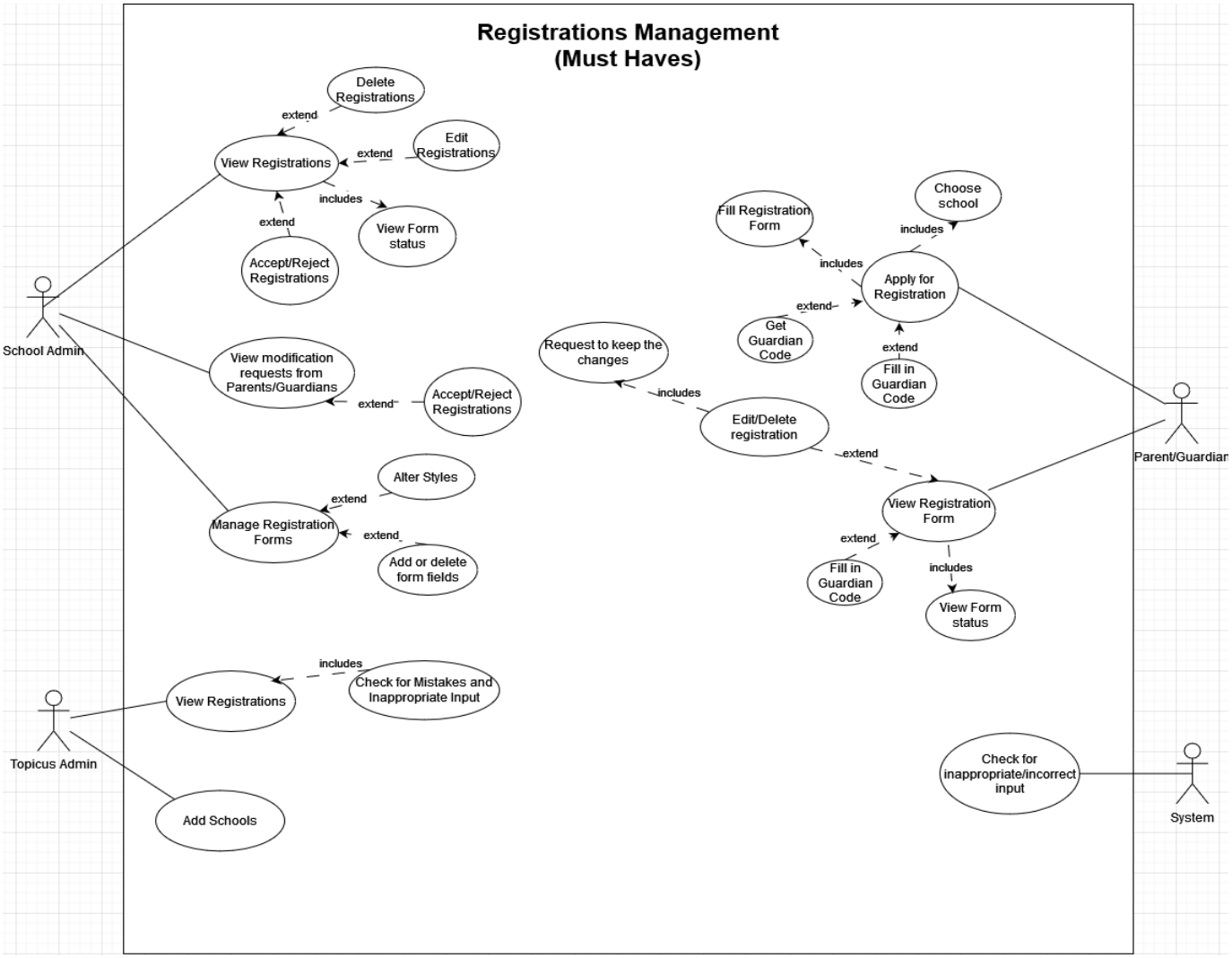

Figure 1: Registrations Management (must have-s)

Figure 1 represents the Use Cases covering all must-have requirements related to Registrations Management. Parents or Guardians (will be called Guardians to make it shorter) can apply for registration only after choosing a school and filling in the registration form fields with appropriate data. If they apply without having an account, they will receive their Guardian Code. In the case when they apply multiple times without signing up, they have to fill in their Guardian Code every time.

A Guardian can view all registrations they have made with their statuses. If they want to view them without having an account, they have to fill in their Guardian Code first. They can Edit or Delete registrations only if they are logged in and by requesting to perform such a modification to a School Admin.

School Admins can view all registrations sent to their school and their statuses. They can accept or reject any registration and also edit or delete the accepted ones. All requests sent from parents for modifications are visible to the School Admins and they can accept or reject them. School Admins can also manage registration forms by changing their stylization and adding or removing fields.

Topicus Admins' role is to check if registrations have appropriate input or not and add schools to the system so that they can use the platform to create their registration forms. The system also has to check if inputs are appropriate.

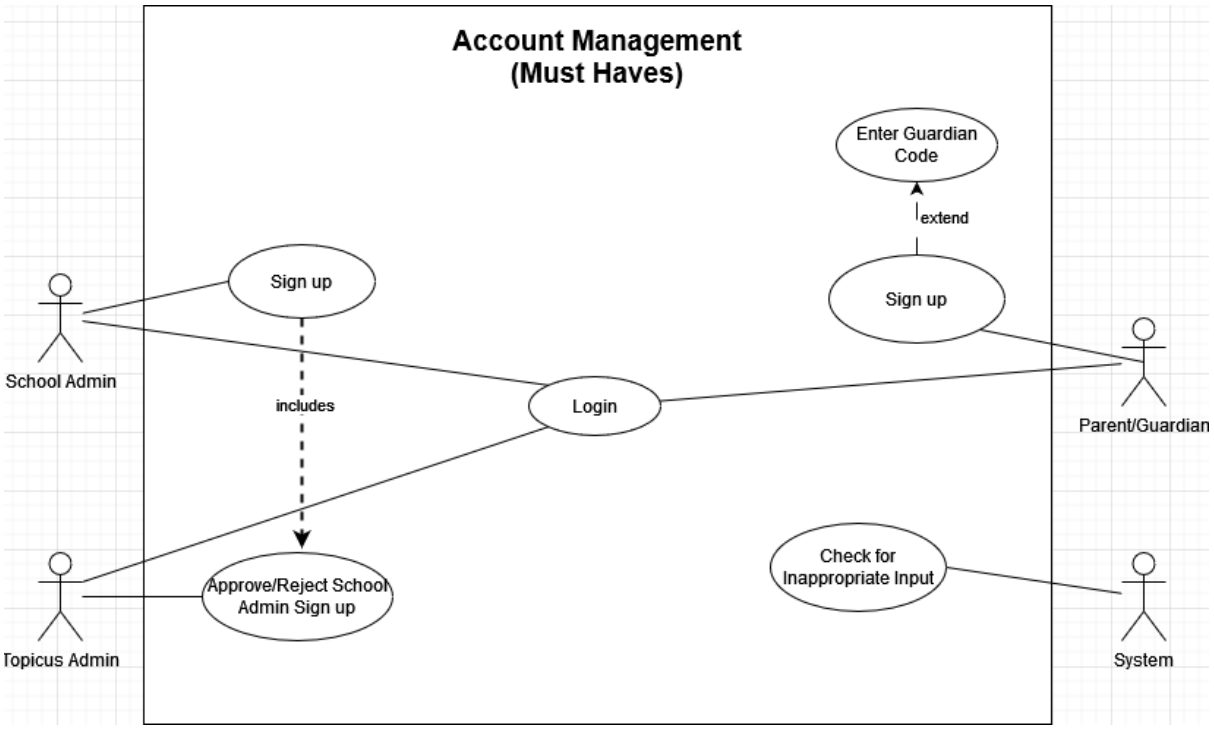

Figure 2: Account Management (must have-s)

Figure 2 represents the use cases covering all the must-have requirements related to Accounts Management. The guardians of the child should be able to create an account by filling in the necessary details. However, if a registration has been made before they created an account they would have to fill in their

Guardian Code which they would have acquired when submitting their registration. The School Admin should also have to sign up to create an account and their account request would be handled by the Topicus Admin who can either accept or reject their requests. The Topicus System should be able to check the input fields for inappropriate or incorrect data. Finally, the guardians, school admins, and topicus admins should be able to log in to access their respective accounts.

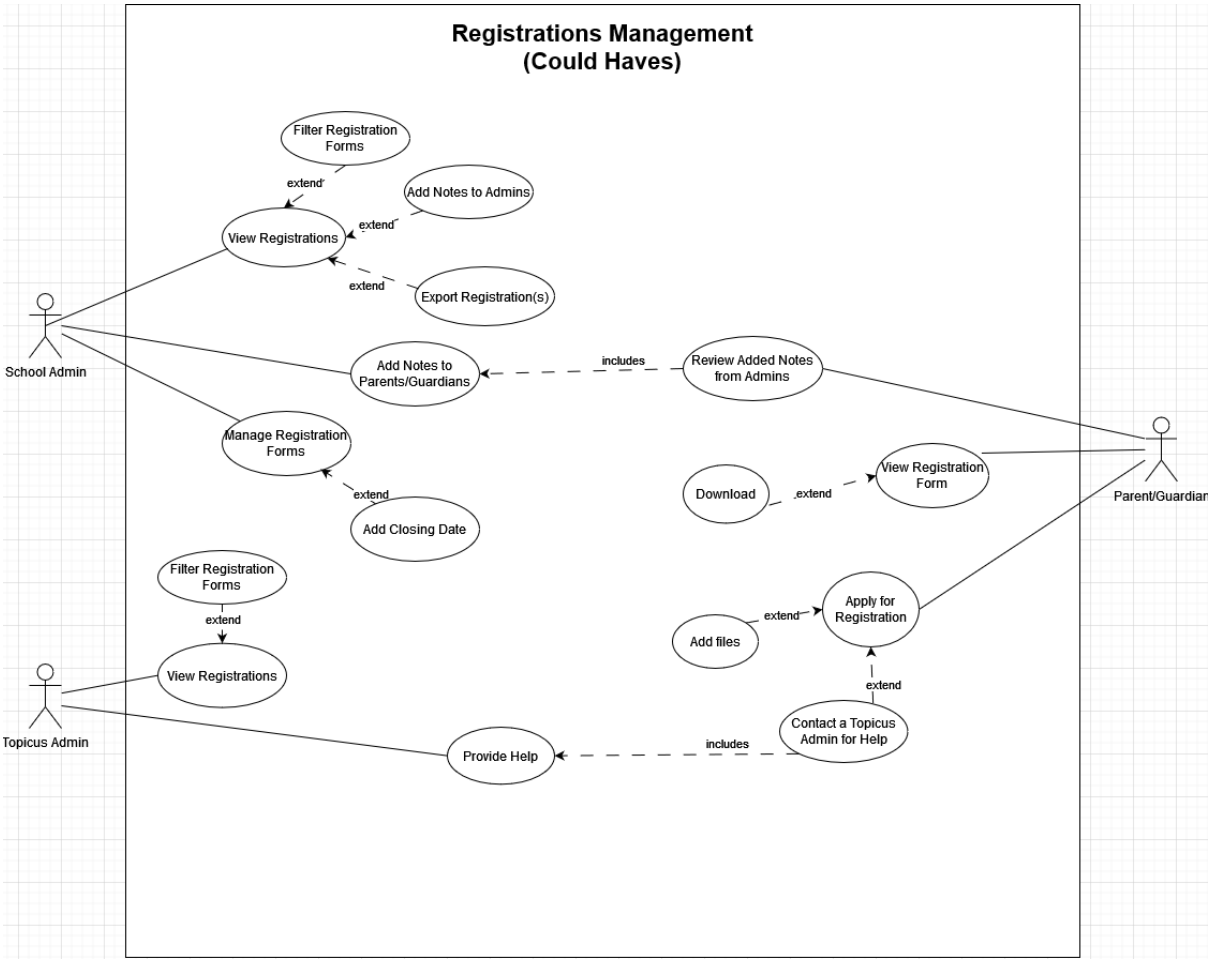

Figure 3: Registrations Management (could have-s)

Figure 3 represents the use cases covering all the could-have requirements related to Registrations Management. In addition to the previously stated features of registrations management, the Guardian could have the ability to add files when applying for registration and also be able to download the registration forms. In case of any difficulties, they could contact the Topicus Admin for support who will be able to provide assistance. Both the Topicus Admin and the School Admin can filter through the registration forms. The School Admin can add notes in the registrations to other School Admins as well as the guardians. The Guardians will then be able to review these notes to see if there are any

important updates. The School Admin can also export the registrations to other applications or devices. Lastly, a closing date can be set for the registrations by the School Admins to make the forms inaccessible after that specific date.

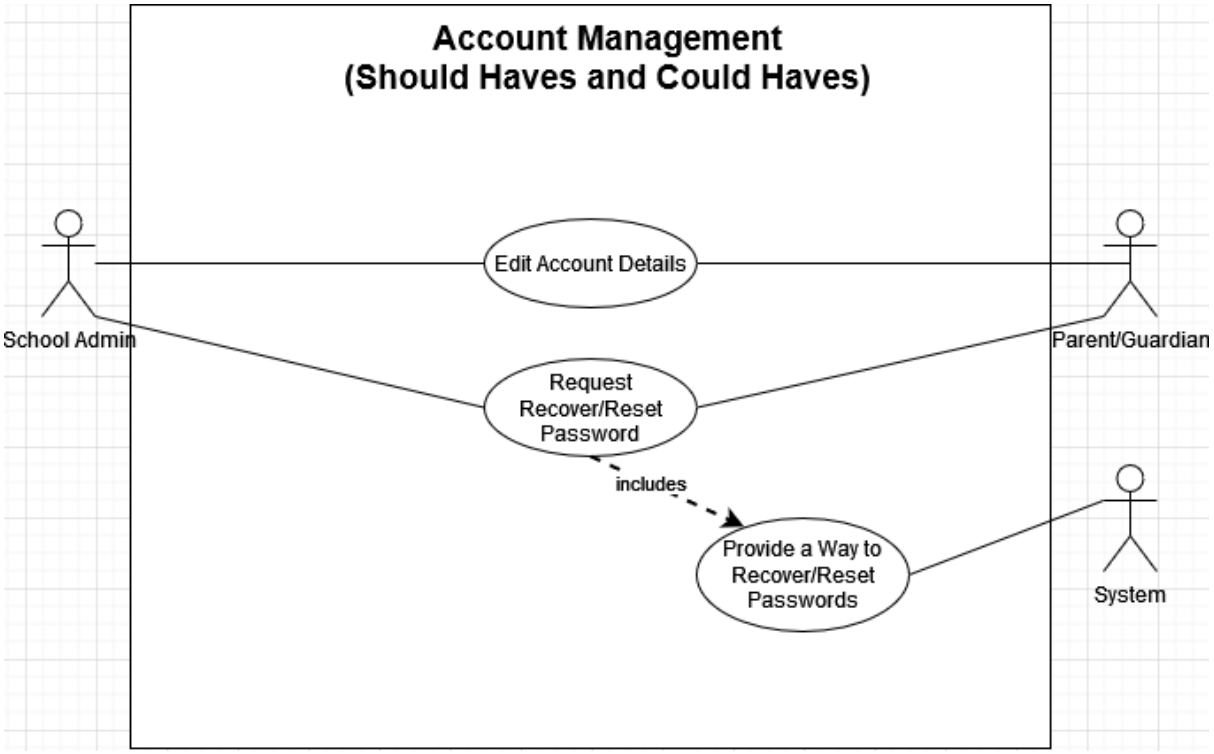

Figure 4: Account Management (could and should have-s)

Figure 4 illustrates the could and should have-s related to Account Management. Both School Admins and Guardians can edit their account details. Moreover, they can recover or reset passwords only after the system has provided them with a way to perform that.

UML Class Diagram:

In the folder accessible from the hyperlink above (or from this link: https://drive.google.com/drive/folders/17zy8TfIM5OjniJQZitz3Pi6GFPNNj3Q2), the class diagram is visible. Though it seems extremely large, many of the specified relations were done to ensure that no level of interaction necessary for the end-user is lost during the conversion to both the database schema and to the code itself.

The first design choice was that different user groups all have access to a System_User account (or may not, due to guardians being able to make registrations without accounts). This model works efficiently to facilitate the

guardians without accounts, as it stores all the necessary identification information on the guardian themself. Of this data, the most critical piece is the Guardian_ID itself, a unique identifier (UUID) for a guardian to use to later link an existing System_User account to their already-recorded data. This can be seen on the sign-up page of the prototype. The other design choice relates to the foreign keys of Address_ID (shown later in the schema). The system will require guardians to fill out two items before accessing the registration form (if they do not have an account already), which are the information in the Guardian table and Address table. In this way, the system will prepare the Address before the Guardian to respect the association. Users can always change their address details later.

Moving further to the Guardian_Child table, this was created to ensure that Children (if required by the client) can have multiple guardians as well. If not, a simple association between the Guardian and Child will be sufficient. The reason for the 1 to * association is due to the fact that a Child entity is not created immediately upon the creation of a Guardian.

The Child, Previous_Education, Medical_Information, and Document (along with subclasses) are data containers that were created after research was conducted on existing registration forms in the Netherlands. Schools will be allowed to select these as predefined options or choose to include similar components in their own registration forms. However, whether this functionality is added to the final product depends on how the rest of the development for the application goes.

The Data_Field is linked to the Registration and to the Registration_Form_Component from the Registration_Form from where it is derived, and the section of that form as well. The same principle applies to the Registration_Form as well, the Registration_Form_Component and Section are linked to the Registration_Form. An important distinction is that a Registration_Form is an uneditable form that a school generates and stores so that it can be used to create instances of a Registration.

One note, the FK of Section_ID and Component_ID on the Data_Field is kept because, after the start date of the form, no deletions or changes can be made to the data to prevent data loss.

Database Schema:

The database schema in SQL is shown below:

```
CREATE TABLE T_System_User (
account_id uuid UNIQUE NOT NULL,
email VARCHAR(255) NOT NULL UNIQUE,
password VARCHAR(255) NOT NULL,
PRIMARY KEY(account_id)
);

CREATE TABLE T_Topicus_Administrator(
admin_id uuid NOT NULL UNIQUE,
surname VARCHAR(255),
phone_number VARCHAR(255),
date_of_birth Date,
given_names VARCHAR(255),
employee_id INT UNIQUE,
PRIMARY KEY(admin_id),
CONSTRAINT fk_topicus FOREIGN KEY(admin_id) REFERENCES
t_system_user(account_id)
);

CREATE TABLE T_Address(
address_id uuid NOT NULL UNIQUE,
postal_code VARCHAR(6) NOT NULL,
street_name VARCHAR(50) NOT NULL,
street_number INT NOT NULL,
city VARCHAR(50) NOT NULL,
country VARCHAR(200) NOT NULL,
phone_number VARCHAR(255) NOT NULL,
PRIMARY KEY(address_id)
);

CREATE TABLE T_Guardian(
guardian_id uuid UNIQUE NOT NULL,
account_id uuid UNIQUE,
address_id uuid UNIQUE,
nationality VARCHAR(100),
surname VARCHAR(100),
phone_number VARCHAR(100),
date_of_birth DATE,
given_names VARCHAR(150),
occupation VARCHAR(150),
company_name VARCHAR(100),
PRIMARY KEY(guardian_id),
```

```
CONSTRAINT fk_guardian_for_account FOREIGN KEY (account_id) REFERENCES
t_system_user(account_id),
CONSTRAINT fk_guardian_for_address FOREIGN KEY (address_id) REFERENCES
t_address(address_id)
);

CREATE TABLE T_Child(
child_id uuid UNIQUE NOT NULL,
surname VARCHAR(100) NOT NULL,
given_names VARCHAR(100) NOT NULL,
preferred_name VARCHAR(100) NOT NULL,
date_of_birth DATE NOT NULL,
BSN INT NOT NULL NOT NULL,
Nationality VARCHAR(100) NOT NULL,
Languages VARCHAR(255) NOT NULL,
PRIMARY KEY(child_id),
);

CREATE TABLE T_Previous_Education(
prev_id uuid UNIQUE NOT NULL,
child_id uuid NOT NULL,
address VARCHAR(255),
institution_name VARCHAR(255),
type VARCHAR(50),
phone_number VARCHAR(100),
duration INT,
description VARCHAR(250),
PRIMARY KEY (prev_id),
CONSTRAINT fk_education_of_child FOREIGN KEY (child_id) REFERENCES
t_child(child_id)
);

CREATE TABLE T_Medical_Information(
child_id uuid NOT NULL UNIQUE,
doctor_name VARCHAR(200),
phone_number VARCHAR(100),
insurance_number VARCHAR(20),
vaccinations VARCHAR(255),
allergies VARCHAR(255),
medicine_required VARCHAR(255),
chronic_illness VARCHAR(255),
PRIMARY KEY(child_id),
CONSTRAINT fk_medicine_child FOREIGN KEY (child_id) REFERENCES t_child(child_id)
);

CREATE TABLE t_School(
school_id uuid UNIQUE NOT NULL,
address_id uuid NOT NULL UNIQUE,
```

```
school_name VARCHAR(150),
school_type VARCHAR(100),
phone_number VARCHAR(100),
email VARCHAR(150),
PRIMARY KEY(school_id),
CONSTRAINT fk_school_address FOREIGN KEY (address_id) REFERENCES
t_address(address_id)
);

CREATE TABLE T_Registration_Form(
registration_form_id uuid UNIQUE NOT NULL,
school_id uuid NOT NULL,
title VARCHAR(50) NOT NULL,
description VARCHAR(255) NOT NULL,
year INT NOT NULL,
form_style VARCHAR(255),
education_type VARCHAR(100),
isDeleted boolean NOT NULL,
isActive boolean NOT NULL,
startDate Date NOT NULL,
PRIMARY KEY(registration_form_id),
CONSTRAINT fk_regform_school FOREIGN KEY (school_id) REFERENCES
t_school(school_id)
);

CREATE TABLE T_Section(
section_id uuid UNIQUE NOT NULL,
registration_form_id uuid NOT NULL,
position_number INT NOT NULL,
title VARCHAR(255) NOT NULL,
PRIMARY KEY(section_id),
CONSTRAINT fk_section_form FOREIGN KEY (registration_form_id) REFERENCES
T_Registration_Form(registration_form_id)
);

CREATE TABLE T_Registration_Form_Component(
component_id uuid UNIQUE NOT NULL,
registration_form_id uuid NOT NULL,
section_id uuid NOT NULL,
position_number INT NOT NULL,
title VARCHAR(255) NOT NULL,
PRIMARY KEY(component_id),
CONSTRAINT fk_comp_form FOREIGN KEY (registration_form_id) REFERENCES
T_Registration_Form(registration_form_id),
CONSTRAINT fk_comp_section FOREIGN KEY (section_id) REFERENCES
T_Section(section_id)
);
```

```
CREATE TABLE T_School_Administrator(
sa_id uuid NOT NULL UNIQUE,
surname VARCHAR(255),
phone_number VARCHAR(255),
date_of_birth VARCHAR(255),
given_names VARCHAR(255),
school_id INT NOT NULL,
employee_id VARCHAR(255),
PRIMARY KEY(sa_id),
CONSTRAINT fk_sadmin FOREIGN KEY(sa_id) REFERENCES
t_system_user(account_id),
CONSTRAINT fk_admin_school FOREIGN KEY (school_id) REFERENCES
t_School(school_id)
);

CREATE TABLE T_Registration(
registration_id uuid UNIQUE NOT NULL,
guardian_id uuid  NOT NULL,
child_id uuid NOT NULL,
school_id uuid NOT NULL,
registration_form_id uuid NOT NULL,
status VARCHAR(100),
PRIMARY KEY(registration_id),
CONSTRAINT fk_reg_guard FOREIGN KEY (guardian_id) REFERENCES
T_Guardian(guardian_id),
CONSTRAINT fk_reg_child FOREIGN KEY (child_id) REFERENCES T_Child(child_id),
CONSTRAINT fk_reg_school FOREIGN KEY (school_id) REFERENCES
T_School(school_id),
CONSTRAINT fk_reg_form FOREIGN KEY (registration_form_id) REFERENCES
T_Registration_Form(registration_form_id),
CONSTRAINT chk_valid_status CHECK (status IN ('ACCEPTED', 'REJECTED',
'UNDER_REVIEW', 'SUBMITTED', 'AWAITING_SUBMISSION',
'MODIFICATIONS_ALLOWED'));
);

CREATE TABLE T_Data_Field(
field_id uuid NOT NULL,
registration_id uuid NOT NULL,
component_id uuid,
title VARCHAR(255),
content VARCHAR(255),
PRIMARY KEY(field_id),
CONSTRAINT fk_field_reg FOREIGN KEY (registration_id) REFERENCES
T_Registration(registration_id),
CONSTRAINT fk_field_component FOREIGN KEY (component_id) REFERENCES
T_registration_form_component(component_id)
);
```

```
CREATE TABLE T_Modification(
mod_id uuid NOT NULL,
sa_id uuid NOT NULL,
reg_id uuid NOT NULL,
description VARCHAR(255),
date DATE,
PRIMARY KEY(mod_id),
CONSTRAINT fk_mod_admin FOREIGN KEY (sa_id) REFERENCES
T_school_administrator(sa_id),
CONSTRAINT fk_mod_reg FOREIGN KEY (reg_id) REFERENCES
T_registration(registration_id)
);

CREATE TABLE T_Guardian_Child(
guardian_id uuid NOT NULL,
child_id uuid NOT NULL,
CONSTRAINT fk_gc_guardian FOREIGN KEY (guardian_id) REFERENCES
T_Guardian(guardian_id),
CONSTRAINT fk_gc_child FOREIGN KEY (child_id) REFERENCES T_Child(child_id),
CONSTRAINT chk_gc_count CHECK ((SELECT COUNT(gc.child_id) FROM
T_Guardian_Child AS gc WHERE gc.child_id = child_id GROUP BY gc.child_id) <= 2)
);
```

As stated, the constraints required have been specified, but the checks have been omitted so far (they will be added later). The SQL Schema was designed based on the class diagram, as said earlier, they were incredibly similar, and the only technicality to specify is the use of UUID instead of an auto-increment integer. This is because the data insertion is managed by the back-end, and it will ensure that privacy at resource endpoints is kept even if the authentication validation is not successful in some cases, as the details cannot be guessed using a UUID.

Conclusion:

Moving forward, the plan for development is already in motion. Reflecting upon the second SPRINT Review, we have reallocated the work to ensure that we are on track. The communication between the back-end and front-end is the largest area of focus for now, and an added focus on security during the SPRINT. One slight technical issue we were not able to address was the fact that the database driver does not work on the first request to the database, but works flawlessly on all other requests afterwards. We were not able to find enough documentation to fix it and will request assistance the next chance we get.

Appendix A: Class Diagram