Hogwarts Management System

Paulo Pinto 103234 Bruna Melo 103453 Tiago Carvalho 104142

Web Semantica
Departamento de Electrónica, Telecomunicações e Informática,
Universidade de Aveiro

April 16, 2024

Contents

Abstract

The report details the development of a web-based application for managing the Hogwarts School of Witchcraft and Wizardry, utilizing Django and Tailwind CSS. The project's initial phase involved transforming data from CSV files into RDF format using XML, enabling the use of SPARQL queries within a GraphDB database for semantic data interaction. It highlights the implementation of three distinct user roles — students, professors, and headmasters — each with tailored access levels to the school's ecosystem. The application features role-based dashboards and dynamic data visualization, supporting functionalities like authentication, course management, and personal information updates. Operations such as data insertion, updates, and deletions are conducted through SPARQL queries, underpinning the system's functionality. It also emphasizes the application's effectiveness in educational data management through semantic web technologies, providing a flexible and powerful platform for Hogwarts School's administrative needs. The developed code, dataset, and additional resources are in a public GitHub repository.

1 Introduction

This project was designed to be a web-based application made with *Django* and *Tailwind* to manage the School of Hogwarts. Initially constructed from a collection of CSV files, our data was transformed into an RDF format using XML, and it's now on a *GraphDB* database. This setup allows us to employ *SPARQL* queries to interact with the data, integrating it within our Python environment through the *SPARQLWrapper* and *rdflib* modules.

We have three types of users, **students**, **professors**, and **headmasters**, that will have different access levels that respect their roles within the school's ecosystem, ensuring that users can only access information pertinent to their responsibilities and interests.

2 Data

2.1 Relational Model

A relational dataset was used as a primary data source for the implemented project, which included data from several entities from the Hogwarts School of Witchcraft and Wizardry, available in the *csv* format. However, since some entries were incomplete, a lot of the data was manually inserted in the relational model.

- **Courses:** Courses available in the Hogwarts School. These are characterized by *name*, *type*, that is, if they are *Core* or *Elective*, corresponding year, *Attending Year* and the course's professor, *ProfessorId*
- **Wizard:** Registers personal information for a person in the Hogwarts School. It can be either a student, professor, or headmaster. Stores information such as the *gender*, *species*, *blood-type*, *eye color*, *HouseId* which references one of the four houses of the Hogwarts School, *wand* description and the *Patronus*.
- **Student:** Stores the student's information. It is mapped to one wizard through *wizardId*, one school, *schoolId*, and the school year they are currently at, *SchoolYear*.
- **Professor:** Stores professor's information. It is mapped to one wizard through *wizardId* and one school through *SchoolId*.
- **Headmaster:** Stores information about the headmaster, including its reference to the wizard entity through *wizardId* and its start date as a headmaster, *start_date*.
- **School:** Stores information about Hogwarts School, includes its *name*, *location*, date of creation *DateOfCreation*, and its reference to the headmaster through the identifier *headmasterId*.
- **Skills:** Registers information about the student's capabilities. It's represented through *name*.
- **Spell:** Corresponds to the spells taught in professors' courses. They are characterized by *name*, *incantation*, *type* which denotes the type of spell, *effect*, and *light* when casting the spell.
- **House:** Stores the information on the four Hogwarts' houses, including the *name* of the house, the animal which represents the *symbol*, a description of the *location* and the reference of the professor in charge *ProfessorInCharge*.
- **Accounts:** To create sessions in the project and allow students, professors, and the headmaster to register in the platform, an additional table was created, which stores information of the *nmec* which is the user's identification number in the system, *password* and the reference to the person through *wizardId*.

The relational dataset also includes two tables that map the several spells to different courses, *RelationCourseSpell* and several skills to other wizards, *RelationWizardSkill*.

2.2 RDF conversion

After analyzing the available data, a *python* script was used to convert the relational model to a non-relational one, which is based on the notion of triples.

Using the XML/RDF syntax, a script was developed that stores a *description* for each table entity. Besides storing the table attributes the entity belonged to in the relational model, a predicate *_type* field was also added to indicate the table it originally belonged to, with the literal being a string that corresponds to the table name.

Additionally, for the tables that map several entities, the following adaptations were performed:

- For the wizard entity, using *RelationWizardSkill*, several skills were added to a singular wizard. Therefore, a single wizard could have none, one, or several skills associated.
- For the courses, since different courses teach different spells, the table *RelationCourseSpell* was used to map all the spells of a course to the latter, using the *teaches_spell* predicate. Therefore, each course is mapped to one, none, or several spells.
- Since there was no mapping in the acquired dataset for the learned courses or currently learning courses for students, a default configuration was made, in which depending on the student's school year and the course's attending year, the student is considered to be learning all the courses which have the same attending year as the student's school year and has learned all the courses which attending year is inferior to the student's school year. For example, if a student is in grade 3, it is considered that it is learning, through the *is_learning* predicate of the grade 3 courses, and has learned, defined through the *learned* predicate, the courses corresponding to the 1 and 2 grades.

Using these configurations on the converter script, an output file *data.rdf* is generated. Upon uploading the new RDF dataset on the *triplestore* Graph DB, it is possible to observe the connection between the graph nodes.

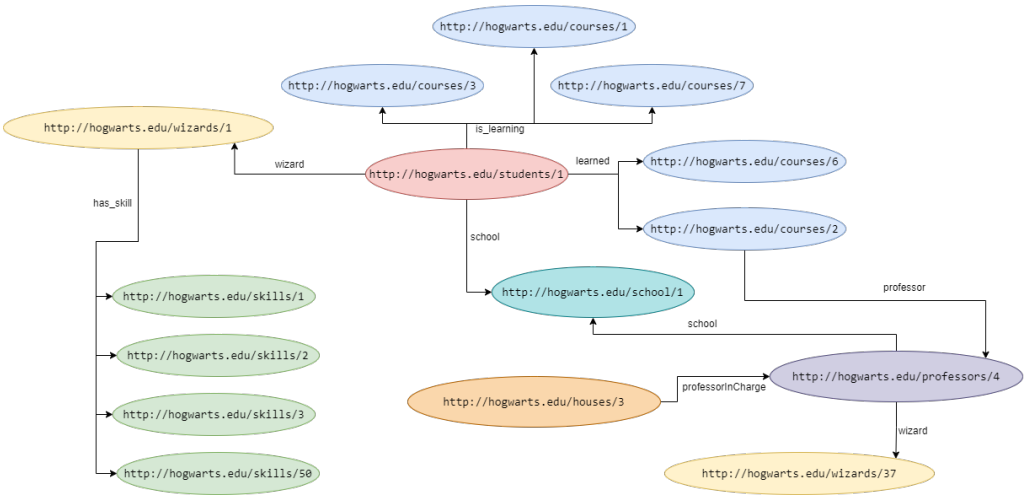

Figure 1: Graph scheme

In Figure 1, it is possible to observe a small scheme part of the created graph, with some, but not all entities.

3 Data operations

3.1 Data visualization queries

To present the information to students, professors, and the headmaster, queries to the GraphDB database were made to obtain, update, delete, and insert relevant information.

This allows users to be presented with only relevant information as well as not allowing other user's sensitive information to not be displayed.

3.1.1 Insert Queries

The following queries correspond to queries that insert data in the graph database.

Insert an existing spell to a certain course, that is, add a spell to the list of spells that are taught under one certain course.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

INSERT {{
  ?course hogwarts:teaches_spell ?spell
}}
WHERE {{
    ?spell hogwarts:_type "spell" .
    ?spell hogwarts:id "{spell_id}" .
    ?course hogwarts:_type "course" .
    ?course hogwarts:id "{course_id}"
}}
```

Listing 1: Add Spell to Course

Add an existing student to an existing course. The operation searches the course through its id, while also searching the student by its id as well, and then appends the designated student the predicate *is_learning* to a course, to designate that the student is now learning a certain course.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

INSERT {{
  ?student hogwarts:is_learning ?course
}}
WHERE {{
    ?course hogwarts:_type "course" .
    ?course hogwarts:id "{course_id}" .
    ?student hogwarts:_type "student" .
    ?student hogwarts:id "{student_id}" .
}}
```

Listing 2: Add Student to Course

A student can register in the Hogwarts School through the registration page. To access the data and operate through it in the future, it is necessary to fill the database with all the information provided by the registering user. While the gender, species, blood type, eye color, house. wand and patronus are not obligatory information and can be null, the name, student number, and password are mandatory parameters that are necessary. The *max_wizard_id*, *max_account_id*, *max_student_id* are calculated through another query and represent the maximum non-occupied id, to guarantee that no repeated ids are being inserted.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

    INSERT DATA {{
    GRAPH <http://hogwarts.edu/ontology#> {{

            <http://hogwarts.edu/wizards/{max_wizard_id}>
                hogwarts:id "{max_wizard_id}" ;
                hogwarts:_type "wizard" ;
                hogwarts:name "{name}" ;
                {gender}
                {species}
                {blood_type}
                {eye_color}
                {house}
                {wand}
                {patronus} .

            <http://hogwarts.edu/accounts/{max_account_id}>
                hogwarts:id "{max_account_id}";
                hogwarts:_type "account" ;
                hogwarts:number "{nmec}" ;
                hogwarts:password "{password}" ;
                hogwarts:wizard <http://hogwarts.edu/wizards/{max_wizard_id}> .


            <http://hogwarts.edu/students/{max_student_id}>
                hogwarts:id "{max_student_id}" ;
                hogwarts:_type "student" ;
                hogwarts:school <http://hogwarts.edu/schools/1> ;
                hogwarts:school_year "1" ;
                hogwarts:wizard <http://hogwarts.edu/wizards/{max_wizard_id}> .
    }}
}}
```

Listing 3: Add New Wizard

3.1.2 Update Queries

The following queries are used to update triples in the GraphDB database.

Change the professor in charge of a course. The query deletes the triple that defines the professor assigned to teach a certain course and assigns a new one based on the provided professor's identification number.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

DELETE {{
  ?course hogwarts:professor ?old_professor .
}}
INSERT {{
  ?course hogwarts:professor ?new_professor .
}}
WHERE {{
    ?course hogwarts:_type "course" .
    ?course hogwarts:id "{course_id}" .
    ?course hogwarts:professor ?old_professor .
    ?new_professor hogwarts:_type "professor" .
    ?new_professor hogwarts:id "{professor_id}" .
}}
```

Listing 4: Update Course Professor

The following query updates the state of a student who is learning a course, passing from the *is_learning* predicate to the *learned* one. This is used to indicate that a student has passed a course.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

DELETE {{
  ?student hogwarts:is_learning ?course .
}}
INSERT {{
  ?student hogwarts:learned ?course .
}}
WHERE {{
    ?course hogwarts:_type "course" .
    ?course hogwarts:id "{course_id}" .
    ?student hogwarts:_type "student" .
    ?student hogwarts:id "{student_id}" .
    ?student hogwarts:is_learning ?course
}}
```

Listing 5: Update Learning Status

The following query updates a wizard's information, with the *query_delete* and *query_insert* being updated depending on the parameters passed in the *python* function. For example, if a wizard only wants to update their name, the variables passed only contain the triples referring to the wizard's name.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

DELETE {{
    {query_delete}
}}
INSERT {{
    {query_insert}
}}
WHERE {{
    ?wizard hogwarts:_type "wizard" .
    ?wizard hogwarts:id "{wizard_id}" .
    {query_delete}
}}
```

Listing 6: Update Wizard Information

3.1.3 Delete Queries

The following queries remove information that is no longer true in the database, based on the user's preferences.

The following query removes a spell from the list of taught spells in a course.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

DELETE {{
  ?course hogwarts:teaches_spell ?spell
}}
WHERE {{
    ?spell hogwarts:_type "spell" .
    ?spell hogwarts:id "{spell_id}" .
    ?course hogwarts:_type "course" .
    ?course hogwarts:id "{course_id}" .
    ?student hogwarts:teaches_spell ?spell
}}
```

Listing 7: Remove Spell from Course

To indicate that a student has not passed a course, the *is_learning* connection from the student to the course is removed, meaning that the student who was currently learning a course has not obtained sufficient marks.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

DELETE {{
  ?student hogwarts:is_learning ?course
}}
WHERE {{
    ?course hogwarts:_type "course" .
    ?course hogwarts:id "{course_id}" .
    ?student hogwarts:_type "student" .
    ?student hogwarts:id "{student_id}" .
    ?student hogwarts:is_learning ?course
}}
```

Listing 8: Remove Student from Course

3.1.4 Read Queries

This section regards queries that are used to present information to the user.

As mentioned above, the acquired relational model was parsed to the RDF format using the *hogwarts:_type* predicate and a string literal which denotes the table name. Therefore, to obtain all occurrences in the table, the following query was created, which returns a list of URIs of the type specified.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?data
WHERE {{
    ?data hogwarts:_type "{type}" .
}}
```

Listing 9: Get Entity by Type

The following query gets all wizards that correspond to professors and then returns a list of identification numbers and corresponding names of professors that have identification numbers different from the provided one. This is used, for example, when switching the professor of a course.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?name ?wizardId
WHERE {{
    ?professor hogwarts:_type "professor" .
    ?professor hogwarts:wizard ?wizard .
    ?wizard hogwarts:id ?wizardId .
    ?wizard hogwarts:name ?name

    FILTER(?wizardId != "{professor_id}")
}}
```

Listing 10: Get All Teachers not Teaching Course

The following query is used to obtain all references of courses taught by a professor, that is, get a list of course URIs taught by a professor referenced by its own URI.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?courses
WHERE {{
    ?courses hogwarts:_type "course" .
    ?courses hogwarts:professor <{user_uri}> .
}}
```

Listing 11: Get Courses URI by Professor URI

In order to obtain more information about a certain entity with its URI, a `DESCRIBE` command was used, which denotes all triples where the provided URI is present.

```
PREFIX hogwarts: <http://hogwarts.edu/>
DESCRIBE <{uri}>
```

Listing 12: Get Entity by URI

When providing a user with their personal information, only the reference to their house is provided. Therefore, to provide the user with more information, the following query was made, which returns the name of the house based on the *houseId* variable.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX hogwarts: <http://hogwarts.edu/ontology#>
SELECT
    ?name
WHERE {{
    <http://hogwarts.edu/houses/{houseId}> hogwarts:name ?name .
}}
```

Listing 13: Get House Name

The following query counts the number of entries which correspond to courses and therefore returns the number of courses existent in the database.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT (COUNT(?course) AS ?count)
WHERE {{
    ?course hogwarts:_type "course" .
}}
```

Listing 14: Get Number of Courses

Similarly to the above, the following query counts the number of entries in the "spell" category, meaning, the output is the number of existing registered spells.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT (COUNT(?spells) AS ?count)
WHERE {{
    ?spells hogwarts:_type "spell" .
}}
```

Listing 15: Get Number of Spells

To relieve some computational effort when managing information, instead of using a query to get all professor information in an instance where no other professor information is necessary, the following query returns only the name of the presented professor through their identification number.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX hogwarts: <http://hogwarts.edu/ontology#>
SELECT
    ?name
WHERE {{
    <{professor_uri}> hogwarts:wizard ?wizard .
    ?wizard hogwarts:name ?name
}}
```

Listing 16: Get Professor Name

When logging in to the platform, to infer the role of the wizard that has just authenticated, that is, if they correspond to a student, professor, or headmaster, the following query was made, which returns the URI of the entity that has authenticated, its identification number in the associated type and the designated type. For example, if a wizard has an ID equal to `5`, for the student with an ID equal to `2` that corresponds to the wizard with ID equal to `5`, the output would be, for the URI, to be equal to `<http://hogwarts.edu/students/2>`, the returned ID would be `2` and the type returned would be "student".

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?type ?role ?type_id
WHERE {{
    ?wizard hogwarts:_type "wizard" .
    ?wizard hogwarts:id "{wizard_id}" .
    OPTIONAL {{
        ?role hogwarts:wizard ?wizard .
        ?role hogwarts:_type ?type .
        ?role hogwarts:id ?type_id .
        FILTER (?type IN ("headmaster", "professor", "student"))
    }}
}}
```

Listing 17: Get Role Info by Wizard ID

The following query returns the name of the school using its describing URI.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX hogwarts: <http://hogwarts.edu/ontology#>
SELECT
    ?name
WHERE {{
    <{school_uri}> hogwarts:name ?name .
}}
```

Listing 18: Get School Name

To add spells to a course, it is convenient for users to not be provided with spells that are already taught in the course. Therefore the following query returns all the spells that are not taught in the course so that the end user, in this case, the headmaster, can select it from the list and put it on the course that it is teaching.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?spellId ?name
WHERE {{
    ?spell hogwarts:_type "spell" .
    ?spell hogwarts:id ?spellId .
    ?spell hogwarts:name ?name .

    FILTER NOT EXISTS {{
        <http://hogwarts.edu/courses/{course_id}> hogwarts:teaches_spell ?spell .
    }}
}}
```

Listing 19: Get Spells not Taught in Course

The query returns a list of students that are currently enrolled in a certain course, that is, the students that have the *is_learning* predicate associated with the designated course. Since the *learned* predicate corresponds to the courses that the students have already been approved for, they do not count as courses they are currently enrolled in.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?student
WHERE {{
    ?course hogwarts:_type "course" .
    ?course hogwarts:id "{course_id}" .
    ?student hogwarts:_type "student" .
    ?student hogwarts:is_learning ?course .
}}
```

Listing 20: Get Students Enrolled in Course

This query regards the problem mentioned above, in which only the students currently enrolled in a course are returned. In this query, only the students who have completed a certain course are returned.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?student
WHERE {{
    ?course hogwarts:_type "course" .
    ?course hogwarts:id "{course_id}" .
    ?student hogwarts:_type "student" .
    ?student hogwarts:learned ?course .
}}
```

Listing 21: Get Students that Completed the Course

The following query returns the students that are not, or have ever been, enrolled in a certain course, that is, it returns the list of students that are not returned by the two previous queries. The query searches all the available students and filters the ones that do not have the predicate *is_learning* or *learned* where the object is the course.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?wizardId ?name
WHERE {{
    ?student hogwarts:_type "student" .
    ?student hogwarts:id ?wizardId .
    ?student hogwarts:wizard ?wizard .
    ?wizard hogwarts:name ?name .

    FILTER NOT EXISTS {{
        ?student hogwarts:is_learning <http://hogwarts.edu/courses/{course_uri}> .
    }}
    FILTER NOT EXISTS {{
        ?student hogwarts:learned <http://hogwarts.edu/courses/{course_uri}> .
    }}
}}
```

Listing 22: Get Students not Enrolled in Course

The query returns the number of students that have currently not finished a certain course.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?id (COUNT(?student) AS ?numberOfStudents)
WHERE {{
    ?student hogwarts:_type "student" ;
             hogwarts:is_learning ?course .
    ?course hogwarts:id ?id
}}
GROUP BY ?id
```

Listing 23: Get Number of Students Learning a Course

The following query returns the number of students that have successfully completed a certain course.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?id (COUNT(?student) AS ?numberOfStudents)
WHERE {{
    ?student hogwarts:_type "student" ;
             hogwarts:learned ?course .
    ?course hogwarts:id ?id
}}
GROUP BY ?id
```

Listing 24: Get Number of Students that have Completed a Course

To obtain some statistics about the total number of students each year, the following query was implemented, which returns the school year and the count of the number of students for each.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT ?school_year (COUNT(?student) AS ?numberOfStudents)
WHERE {{
    ?student hogwarts:_type "student" ;
             hogwarts:school_year ?school_year .
}}
GROUP BY ?school_year
```

Listing 25: Get Number of Students in each School Year

The following query returns the house identification number based on the house name it is given, performing a case-insensitive search.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT
    ?houseId
WHERE {{
    ?house hogwarts:_type "house" .
    ?house hogwarts:name ?houseName .
    FILTER(LCASE(?houseName) = LCASE("{house}")) .
    ?house hogwarts:id ?houseId .
}}
```

Listing 26: Get House Id Based on House Name

The following query searches all identification numbers for all wizards, students, and accounts and returns the maximum identification number found. This is used when registering a user and ensuring that no identification numbers are overlapped.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT
    (MAX(xsd:integer(?wizardId)) AS ?nextWizardId)
    (MAX(xsd:integer(?studentId)) AS ?nextStudentId)
    (MAX(xsd:integer(?accountId)) AS ?nextAccountId)
WHERE {{
    ?wizard hogwarts:_type "wizard" .
    ?wizard hogwarts:id ?wizardId .
    ?student hogwarts:_type "student" .
    ?student hogwarts:id ?studentId .
    ?account hogwarts:_type "account" .
    ?account hogwarts:id ?accountId .
}}
```

Listing 27: Get Max IDs

The login query returns the wizard for a selected password and student number. If the credentials are not correct, no wizard is returned, and therefore, the login attempt is unsuccessful.

```
PREFIX hogwarts: <http://hogwarts.edu/ontology#>

SELECT
?wizardId
WHERE
{{
    ?account hogwarts:_type "account" .
    ?account hogwarts:number {nmec_literal} .
    ?account hogwarts:password {password_literal} .
    ?account hogwarts:wizard ?wizard .
    ?wizard hogwarts:id ?wizardId
}}
```

Listing 28: Login Query

4 App Functionalities

The developed project is divided into four main components:

- **Authentication:** For users to register or log in to the platform, a dedicated page was created that enables users who already have an account to access their dashboard and a registration page that allows the creation of a student. The registration page does not allow the registration of professors or other headmasters.
- **Student view:** When logged in as a student the user is redirected to the Students Dashboard. Here he is capable of seeing and editing information.
- **Professor view:** When logged in as a professor, the user is redirected to the Professor Dashboard, which displays information about themselves as well as information about the course they teach (that is, if they teach any course) as well as the spells in that course.
- **Headmaster view:** When logged in as a headmaster, the user is redirected to the Headmaster Dashboard, which not only displays information about themselves, but also well as display information about existing courses, all students, and two statistical graphs, one that displays the number of spells per course and another that displays information about the number of students per school year.

4.1 Authentication

When the user first runs the program, they will be presented with the login page, where they can insert their student number and password to access the dashboard based on their role as a student, professor, or headmaster.

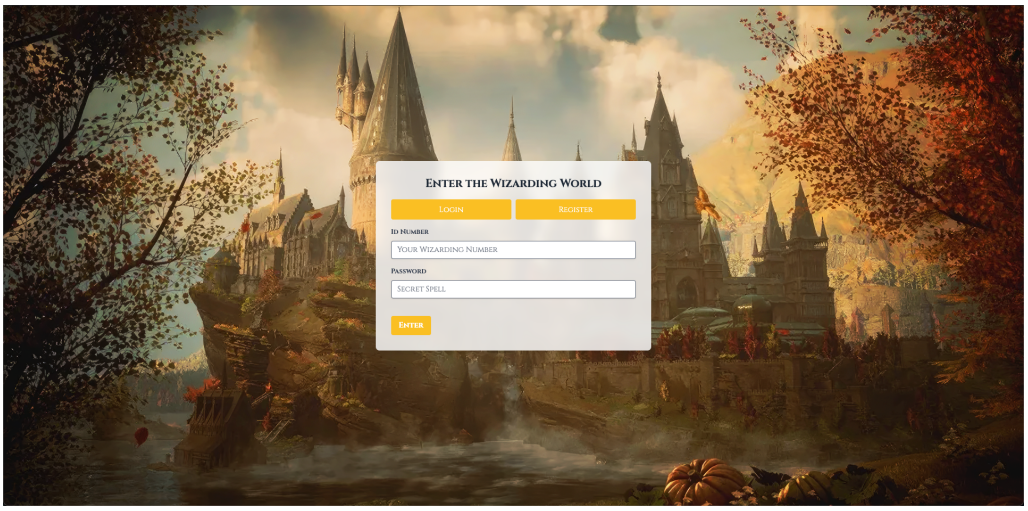

Figure 2: Login

For users that do not have an account, they can instead register on the platform, by inserting their personal information in the displayed form, including their student number, password, full name, eye color, patronus and wand. Additionally, the user can select one of the four available houses, gender, blood-type and species. However, when the user registers and uses the created account to login, the information displayed in the dashboard will be very little, since by default the student has no courses upon creation, and is dependent on the headmaster adding courses to visualize information in the dashboard.

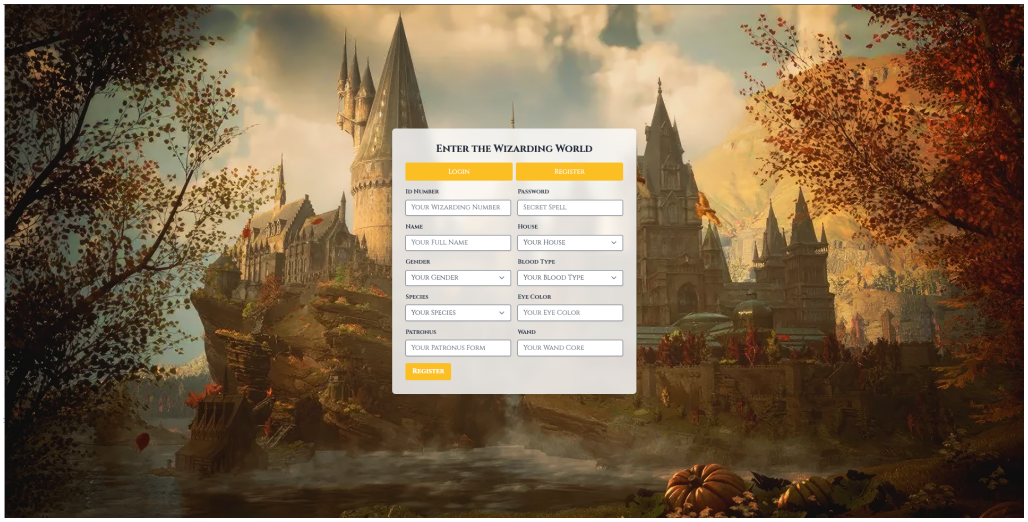

Figure 3: Register

4.2 Student Dashboard

Figure 4 demonstrates the student dashboard where the logged student can visualize their personal information, as well as the name of the school they are currently enrolled in, the acquired skills' name, and the name of the corresponding house.

Additionally, it is also possible to infer about the courses the student is currently enrolled in, as well as information about the courses that were already learned.

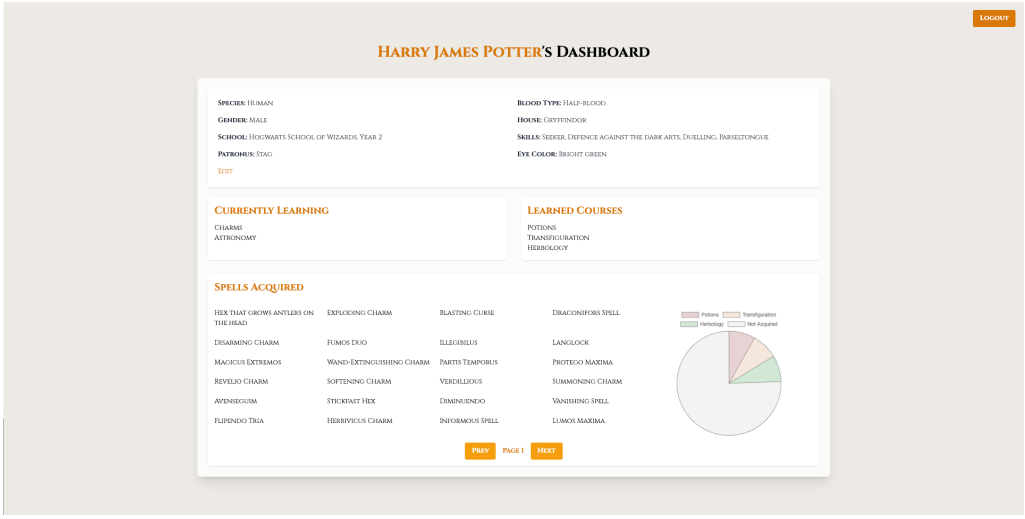

Figure 4: Student Dashboard

Figure 5 demonstrates the spell details. Upon pressing a spell from the list of acquired spells, a modal displays detailed information about the one selected, including its intended effect, incantation, and light color.

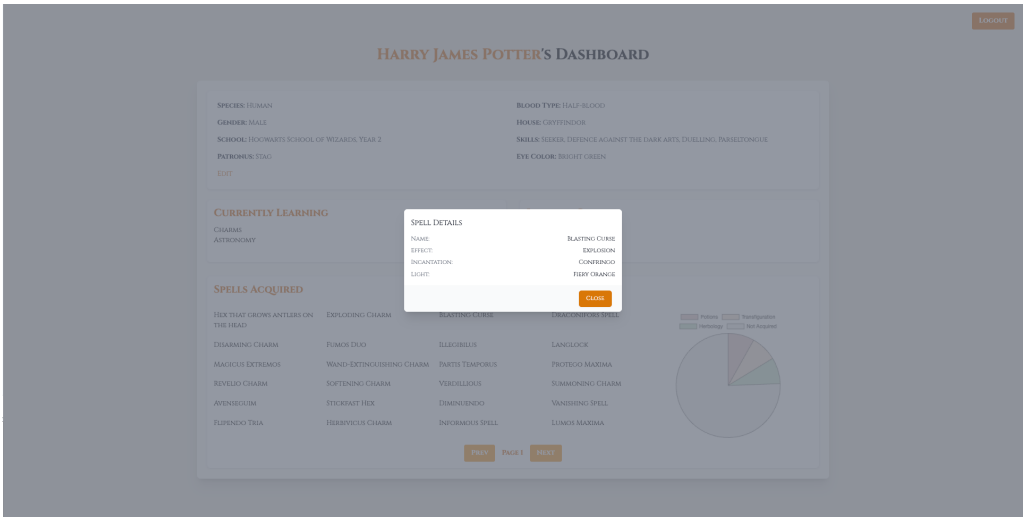

Figure 5: Spell Details in Student Dashboard

The view in Figure 6 is obtained upon pressing one of the courses, whether they are currently learning or have already learned it. It displays information about all the spells, and their name, effect, and type, as well as display the name of the professor in charge, if available.

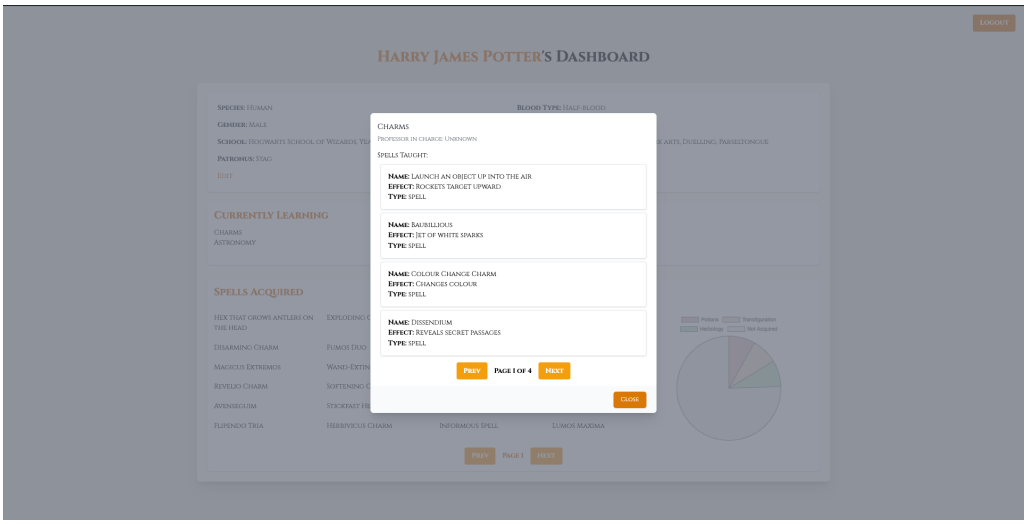

Figure 6: Student Course Details

To update a student's information, the student can use the `edit` button and update their name, gender, blood type, species eye color, patrons, and wand. For the blood type, species, and gender, the user can only specify from a certain range of options, while the name, eye color, patrons, and wand can be specified however the user wishes.

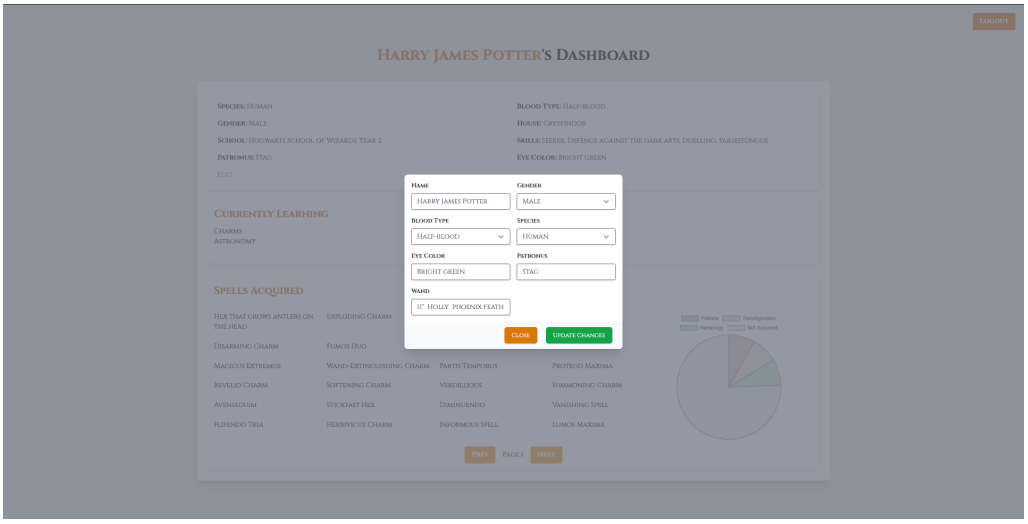

Figure 7: Student's Information Edit

4.3 Professors Dashboard

When the user authenticates as a professor, a different dashboard will be presented. While the professor can visualize their information in the same way a student does, there are no learned or learning courses, since logically they can only teach a course, but not attend it. However, for the courses they teach, they can visualize the courses they currently teach, as well as the students enrolled in each course they teach, as can be visualized in Figure 8.

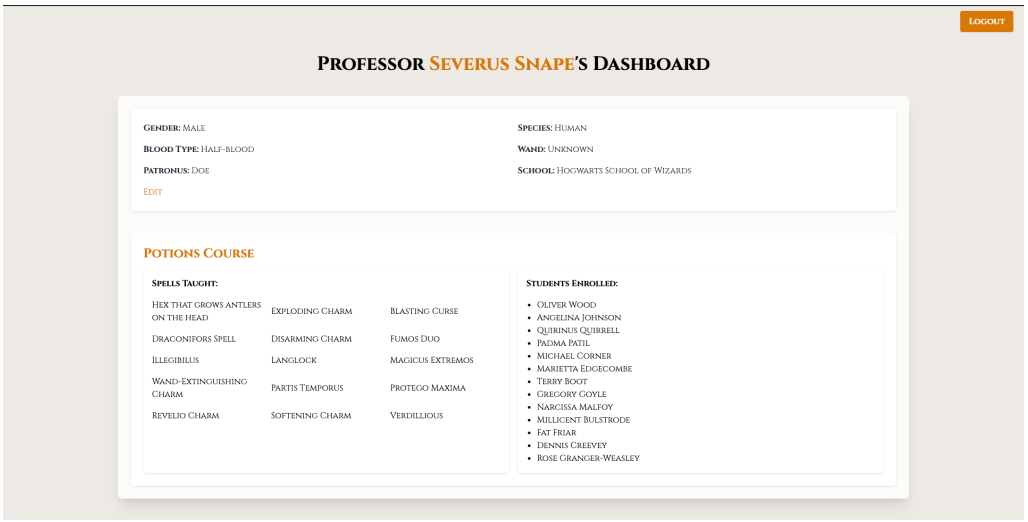

Figure 8: Professor Dashboard

Similarly to students, the professor can also change their personal information, including name, gender, blood type, species, eye color, patronus, and wand, as can be viewed in Figure 9.

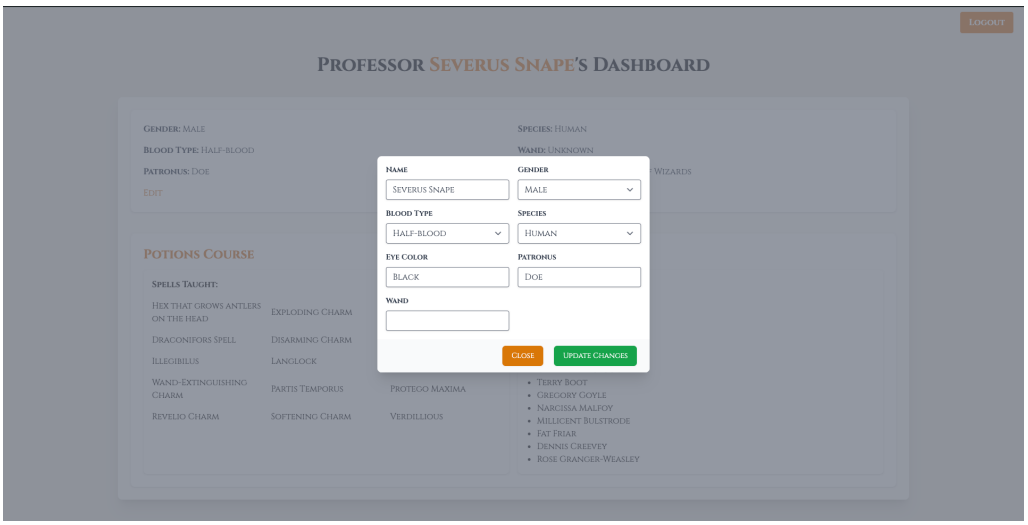

Figure 9: Professor information Edit

In Figure 10, it is possible to infer the presented view upon selecting a spell from the list of spells from the taught course. The professor can view its details including name, effect, incantation, and light color.

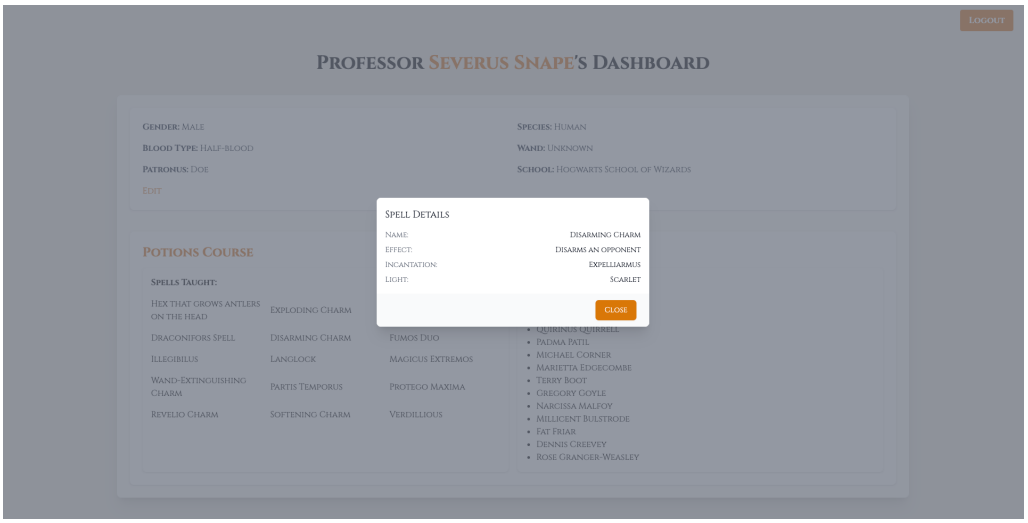

Figure 10: Spell Details

Upon pressing a student from the list of students that attend a course, a new model appears that displays detailed information about the selected student.

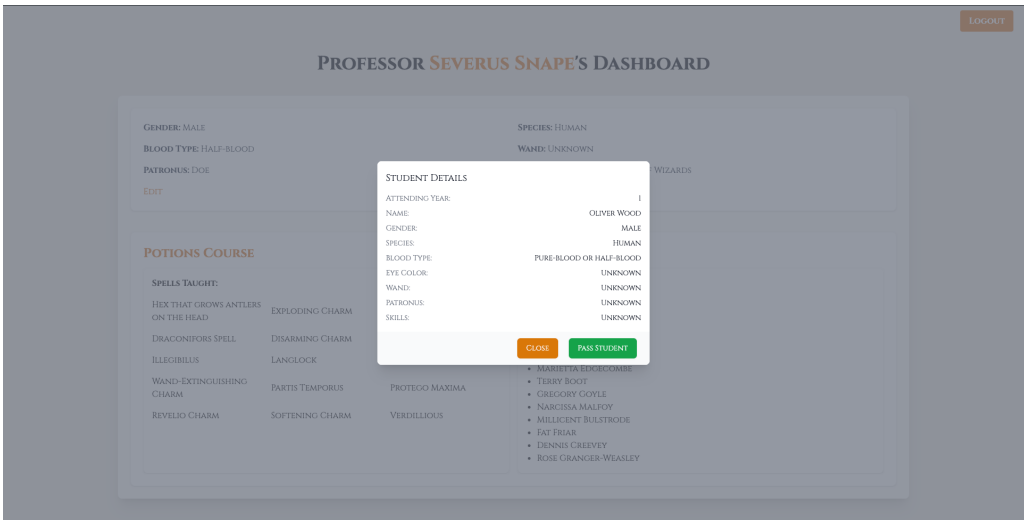

Figure 11: Student Details

4.4 Headmaster Dashboard

When the user authenticates as a headmaster, a different dashboard from both the student and the professor will be presented. The headmaster, being the wizard with the most authority in the Hogwarts School, has access to several elements. In their dashboard, they can visualize all the registered courses, as can be visualized in Figure 12.

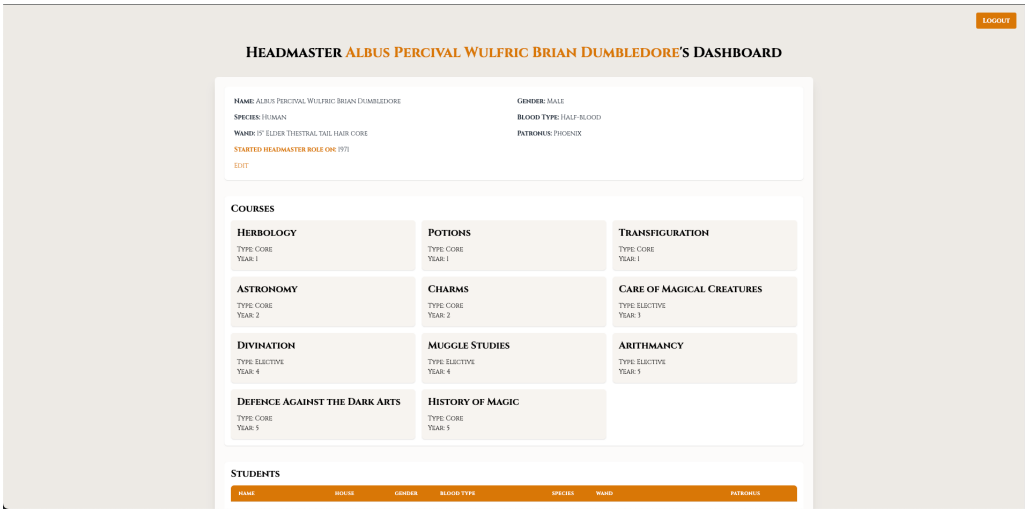

Figure 12: Headmaster Top Dashboard

Additionally, as can be viewed in Figure 13, the headmaster can also visualize all the students and their respective information, including the house they belong to, their gender, blood type, species, wand, and patronus, even though they cannot change that personal information.

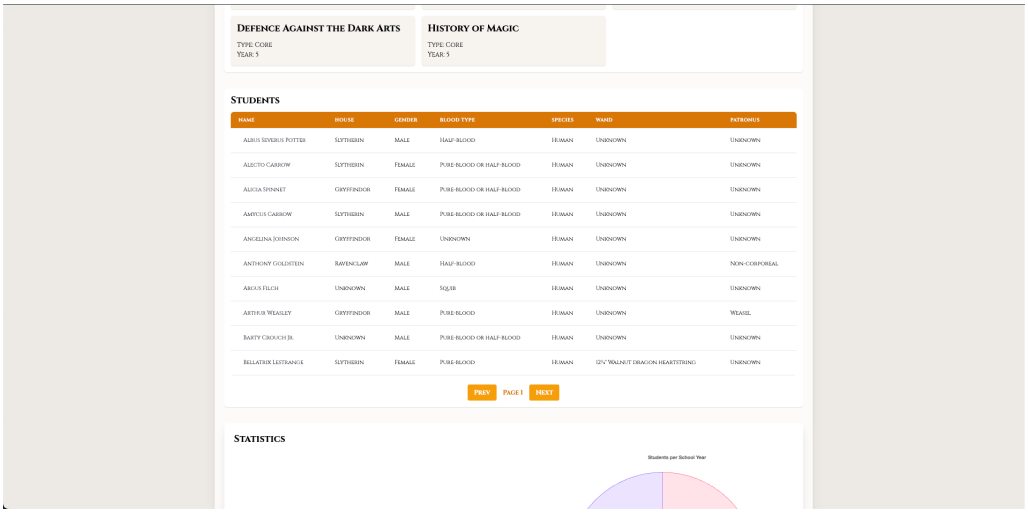

Figure 13: Headmaster Middle Dashboard

In the last part of the dashboard, the headmaster has access to two different charts, the left one is a bar chart that tells us the number of spells per type.

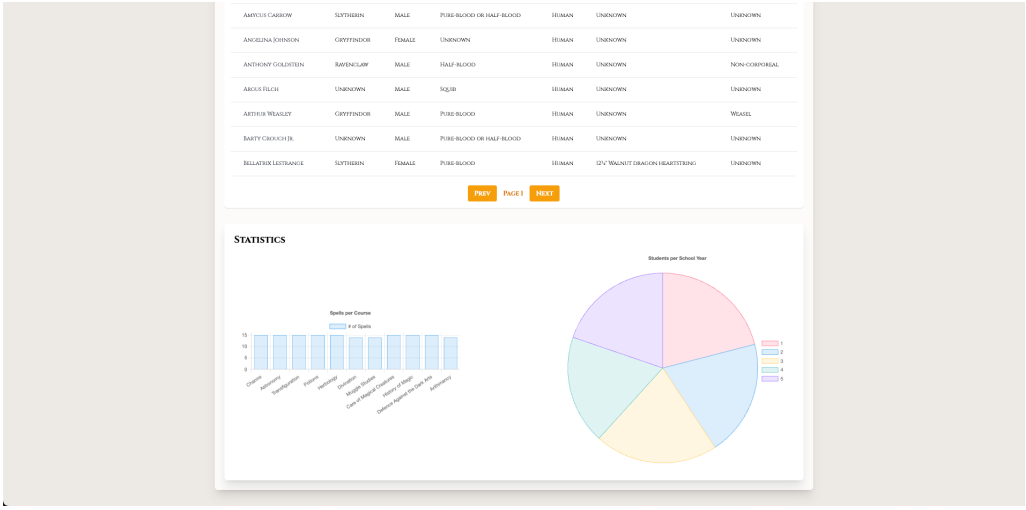

Figure 14: Headmaster Bottom Dashboard

Additionally, the headmaster can also edit their personal information, in the same way a student and a professor can, as can be viewed in Figure 15.

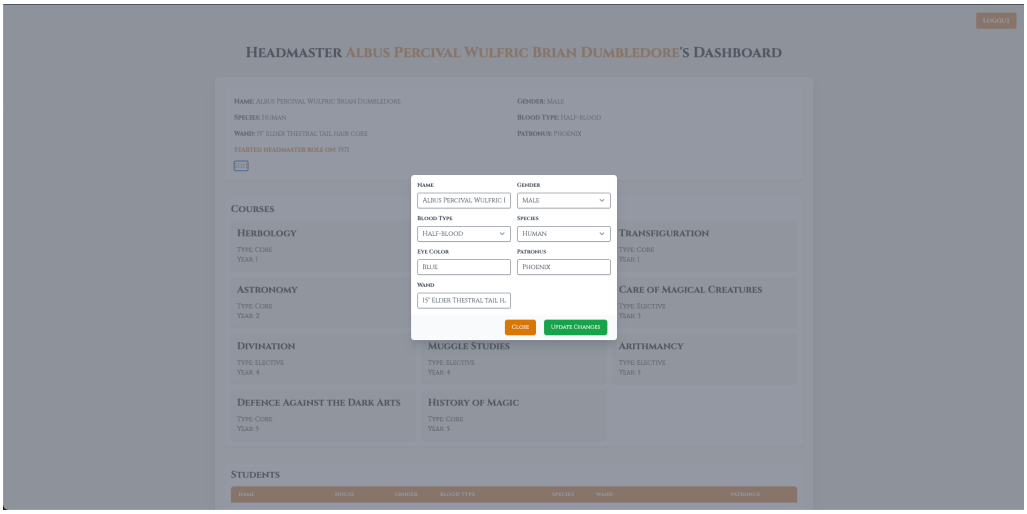

Figure 15: Headmaster's Information Edit

In this part it's possible to see some information about a selected course, being also able to press a button to see details about that course.

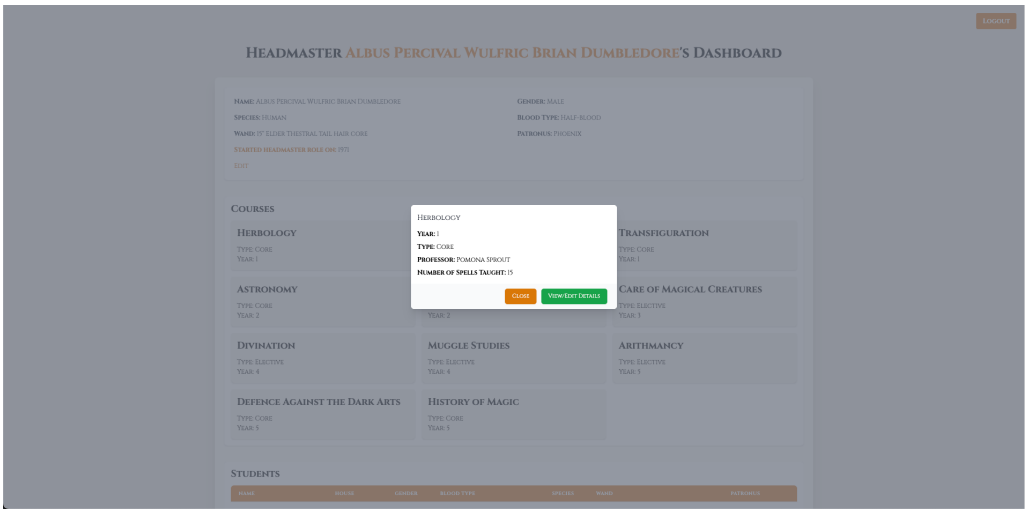

Figure 16: Course Info

In order to obtain more information about a course, the headmaster can view its details by selecting one course from the list of registered courses, which opens a new page with the course's main details, as well as the spells taught in that course and a list of student enrolled, as can be viewed in Figure 17.

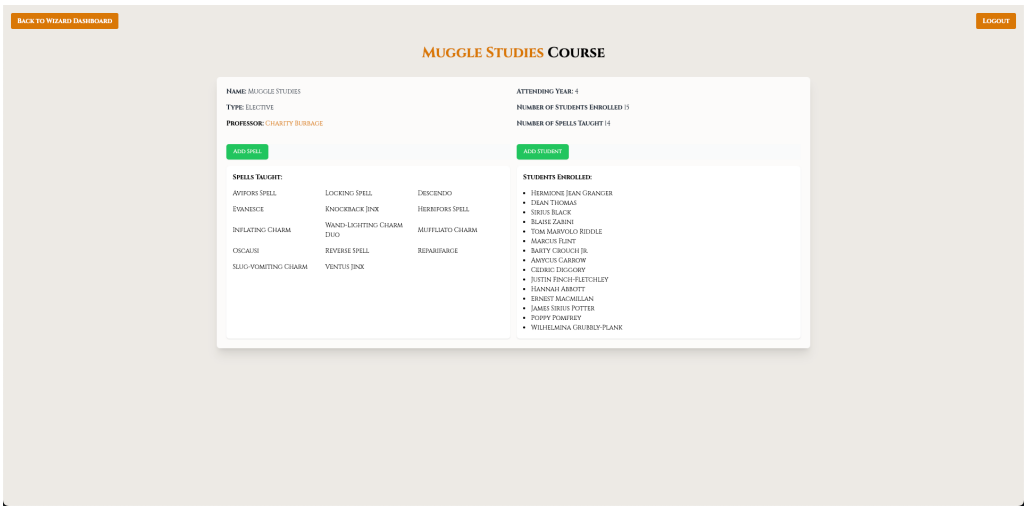

Figure 17: Course Details

Here is possible to see the details about the teacher responsible for the course and is also able to press a button to change the responsibility to a new teacher.

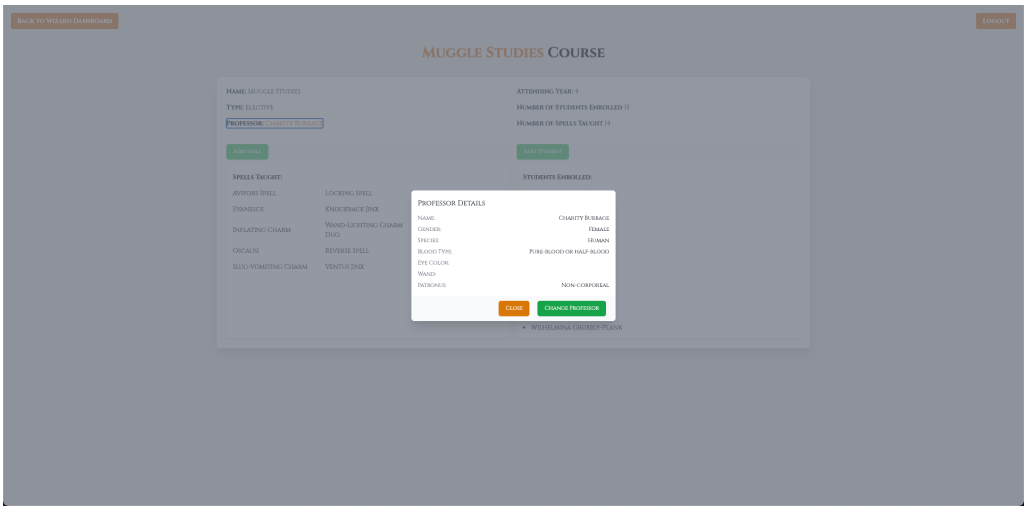

Figure 18: Professor Details

As can be viewed in Figure 19, The headmaster can also add a spell to a certain course, that is, require the learning of a spell for a certain course, from a list of spells that are not taught in that course, to not have duplicate entries.

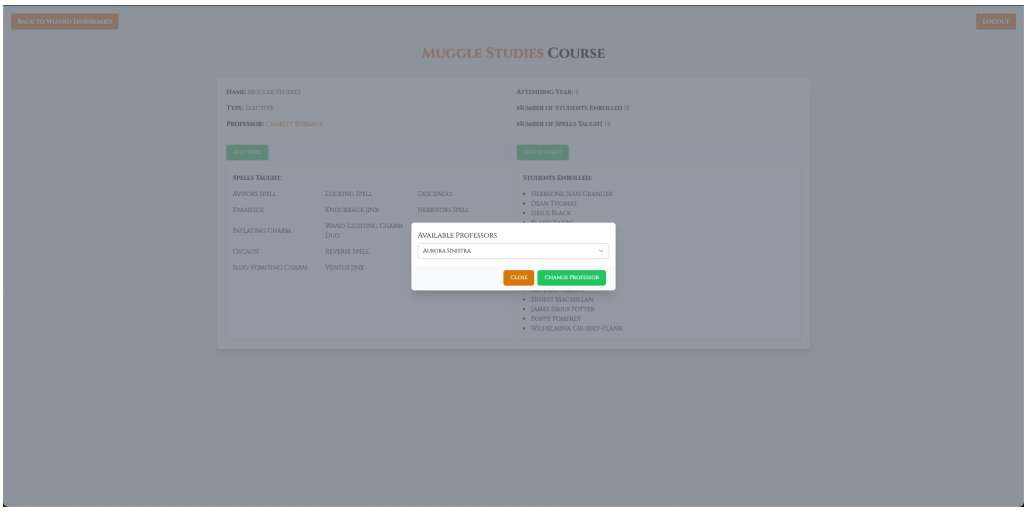

Figure 19: Course Professor Assignment

In this figure, the headmaster can add a new spell to a course, so it is taught in it.

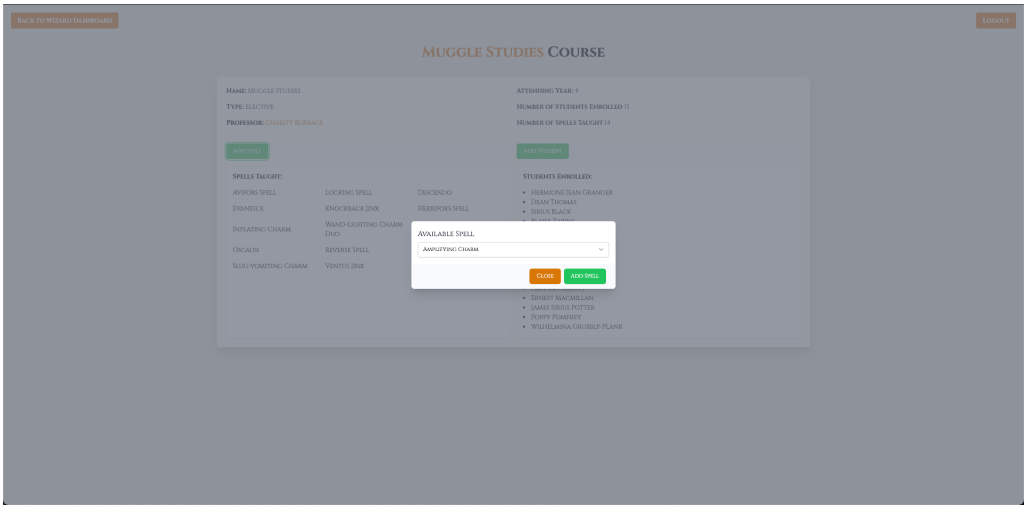

Figure 20: Add Spell to Course

For this part, it's possible to see spell details and remove them from the course.

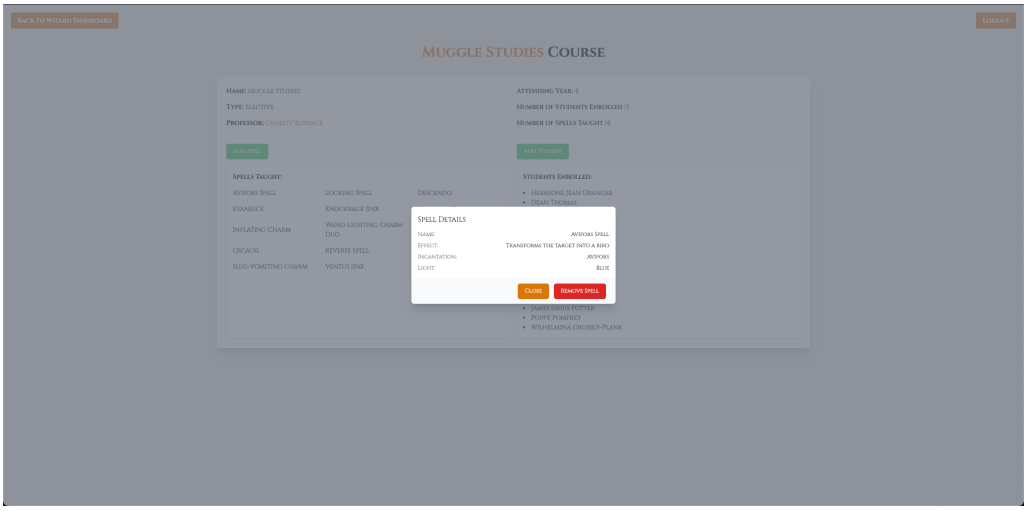

Figure 21: Spell Details

Additionally, a headmaster can also enroll a student in a certain course, from a list of students who have not yet learned or are not learning a course, as can be viewed in Figure 22.

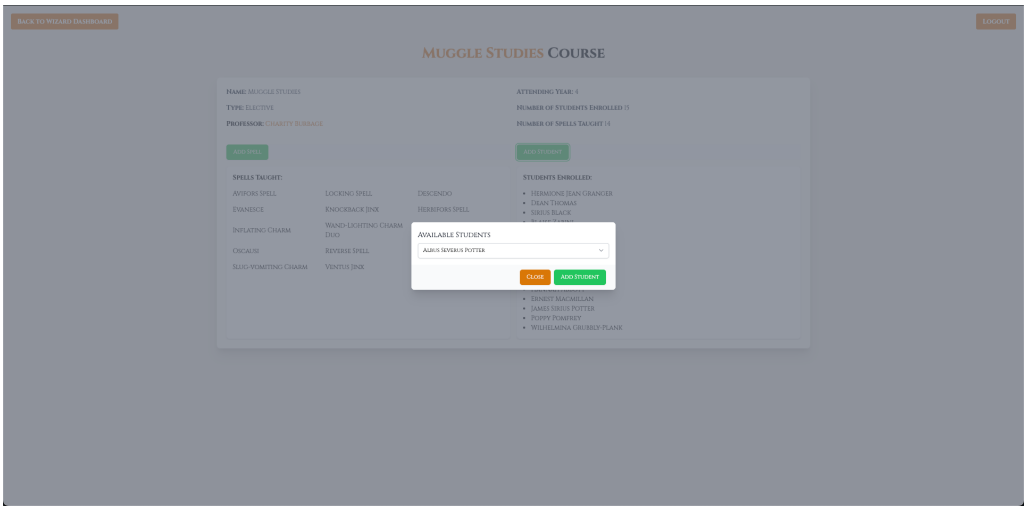

Figure 22: Add Student to Course

Finally, the headmaster can view detailed information about a student who is currently enrolled in a course, with the option to remove it from the course.

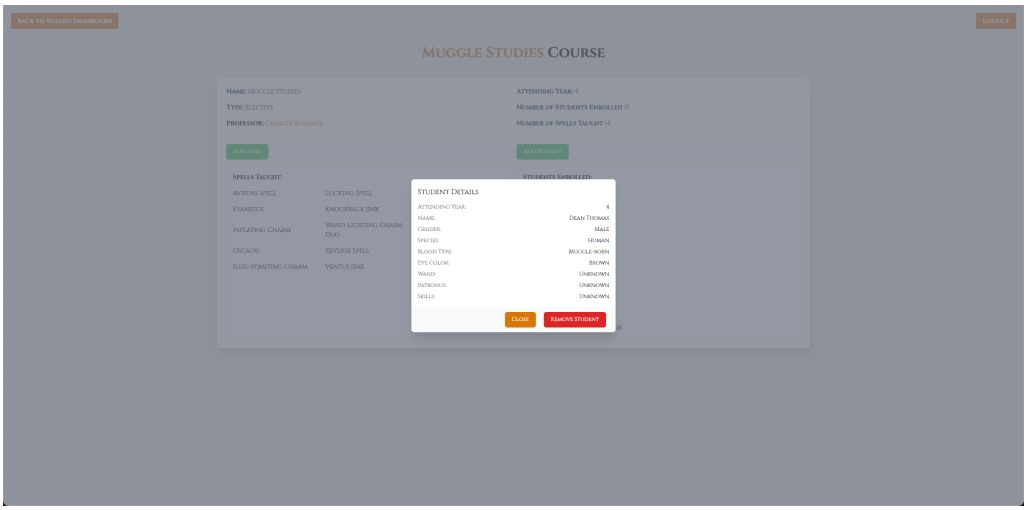

Figure 23: Student Details

5 How to Run

5.1 Installing the project requirements

To first be able to run the application, the user must install the module requirements. In other to contain the installations, we highly recommend the use of Python virtual environments.

The user can install the modules with the guidance of the following steps:

1. Create a virtual environment, by running one of the following commands:

   ```
   python<version> -m venv <virtual-environment-name>

   python -v venv venv
   ```

   Alternatively, some other libraries might be used to generate the virtual environment.

   ```
   virtualenv <virtual-environment-name>

   virtualenv venv
   ```

2. Enter the virtual environment, by running this command:

   ```
   source venv/bin/activate
   # or
   ./venv/scripts/activate
   ```

3. Install the requirement modules:

   ```
   python manage.py tailwind install
   ```

5.2 Running the application

To run the application, the user must ensure that the GraphDB database running. To run the application the user should follow these instructions:

1. Have GraphDB running and the *dataset data/data_.rdf* imported in a repository named `ws_repository`.

2. Create a .env file inside the root folder with the following configuration:

   ```
   REPO_URL=http://127.0.0.1:7200/repositories/ws_project
   REPO_URL_UPDATE=http://127.0.0.1:7200/repositories/ws_project/statements
   ```

3. Open two separate terminals from the root folder.

4. In the first terminal, run this command, responsible for launching the application:

   ```
   cd webproj
   python manage.py runserver
   ```

5. In the other terminal, run this command, which will load the Tailwind CSS styles:

   ```
   cd webproj
   python manage.py tailwind start
   ```

Permission	Id Number	Password
Student	104142	Pass.Muito.Forte123
Professor	103234	Pass.Muito.Forte123
Headmaster	103453	Pass.Muito.Forte123

6 Conclusion

The report outlines the development of a web application for managing the Hogwarts School of Witchcraft and Wizardry using Django. The project began by converting CSV data into RDF format, enabling SPARQL queries within a GraphDB database. This conversion allowed for a semantic web approach, ensuring interconnected and meaningful data manipulation.

The application integrates three user roles: students, professors, and headmasters, with access levels tailored to their specific needs. Features include authentication, role-based dashboards, and dynamic data visualization. The data model encompasses entities such as Courses, Wizards, Schools, Skills, Spells, and Houses, alongside accounts for session management. Operations such as data insertion, updates, and deletions are facilitated through SPARQL queries, making the application a powerful tool for educational data management.

In conclusion, this project successfully demonstrates the application of semantic web technologies to manage educational data. By converting relational data to RDF and leveraging a graph database, the application provides a flexible and powerful platform for managing the Hogwarts School. The implementation of role-based access controls and tailored dashboards enhances user experience, making it a comprehensive tool for all stakeholders within the school.

This report outlines the project's approach from data transformation to application functionalities and running instructions, showcasing the potential of semantic web technologies in educational data management.

7 Appendix

The developed code is publicly available on our GitHub repository. It also contains the dataset and data conversion methods.

As another result of this project, we created a PowerPoint presentation detailing the theme, data, data conversion, application functionalities and other details. The presentation is also included in the repository as WS TP1 Presentation.pdf.

Additionally, we took the opportunity to record our final product and created a video presentation showcasing the application functionalities and some elements used to interact with the application. The video is available in Youtube.