



Nisotak Author Tool
2020

1 Introduction

Nisotak Author Tool was developed to offer a simple and easy way for content creators to add and update lessons, that will then be served to the learner interface. The goal was to develop it in a manner that allows the authors to create these lessons in a flexible way. This was done so by dividing the lesson in different components, which can be added and ordered as the author wishes. Currently, these components are:

- Content pages: these pages allow the users to create, delete and order content components, which can be:
 - Text/Story: the users can include text in an indigenous language, as well as its translation to English. Accompanying the story the users can include, if they wish, an audio and an image file.
 - Phrases: this component allows the users to break a phrase into morphemes and include the translations in English. The users can change the order, add and delete morphemes. Besides that, they can include an audio and an image file.
- Multiple choice questions: this component allows the users to create questions to test the student knowledge on the content. The users can include instructions in an indigenous language and its translation. They can also select an audio file to accompany the instructions. They can include as many answer options as they wish and they should select which is the correct one. An audio file can be added to accompany each answer option.

This tool is being developed following a microservices architecture which facilitates the maintainability of the code as well as the development and deployment of new features. The idea is that over time, other components are developed to allow content creators to be creative, offering different lesson types to the learners. The backend is being implemented in Node.js and the frontend in React.js.

The next chapters we provide an overview of the tool. In Chapter 2 we present the access types available. In Chapter 3 we show the user interface and provide a description of the features available. In Chapter 4 we included an overview of the microservices developed, including the endpoints available in each of them.

2 Access types

As explained in the introduction, this tool is meant to allow content creators to add and update lessons that will be served to the learner interface. What each content creator can access in the tool will depend on the access type they have. Currently the application offers three types of access for content creators:

- Admin: users with this access type can see, edit and delete all the lessons in the platform, including lessons created by other users. Moreover, they can manage the access to the tool. This means they can change the access types from other users and they can approve access requests to the tool. It is important to highlight that the first user ever to register in the Author Tool - when deploying a new instance or resetting the database, for example - will be automatically set as an admin.
- Editor: users with this access type can see, edit and delete all the lessons in the platform, including lessons created by other users.
- Author: users with this access type can see, edit and delete all the lessons they created themselves.

Besides the access types described above, a fourth type was created to allow the learner tool - or other authorized people - to perform get requests to the lessons API: the API reader access type. Users with this access type cannot edit and delete lessons. It is important to highlight that the users from the learner interface do not need to be registered in the author tool. The two applications share a JWT secret, so the learner interface generates the token which is necessary to perform the get requests on the author tool. This access type can also be used to allow reading access to the API for other purposes (e.g. if the admin wants to give reading access for developers that want to build applications using the database). In this case, the person can use the register page to request access and the admins can approve the user access normally, as they would do for other access types. However, after being approved by an admin as a reader, users cannot access the author tool application.

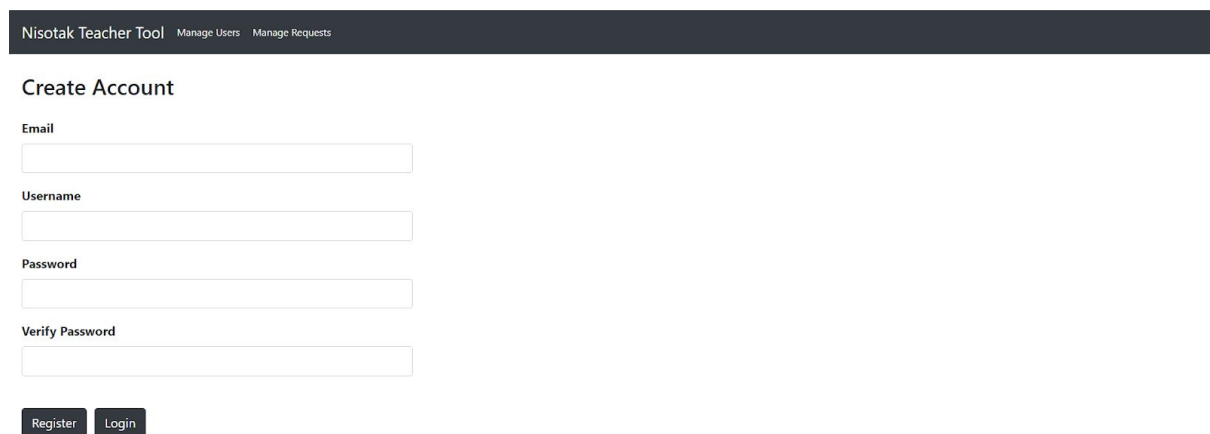
3 The interface

In this chapter we will provide an overview of Nisotak Author Tool's application. The implementation of the interface was done using React.js and can be found in the *author-tool-app* folder in the root of the Github project. The next sections in this chapter will describe each page from the application.

3.1 Register Page

Any person that wants to access the system needs to request access in the register page, shown in figure 3.1. The person provides email, username and password. If the email was previously registered in the system, the request will not be created, as the email is the unique identifier of the users. The username, on the other hand, does not need to be unique.

Once the user submits the request, the user needs to wait for the admin approval. Only after that the user will be able to log in the system.



Nisotak Teacher Tool Manage Users Manage Requests

Create Account

Email

Username

Password

Verify Password

Register Login

Figure 3.1 Register page

3.2 Login Page

The user needs to log in to gain access to the system. To log in, the user needs to provide email and password in the login page (figure 3.2). The author tool sends a request to the server and receives a JWT, which will enable the user to gain access to the other features in the system, to the extent their access type permits. The token expires every two hour. After that time, the user needs to login again. As mentioned in the previous chapter, users with read only access cannot access the application. If they try to use the login page to access the system, their request will be denied. Only Admins, Authors and Editors are able to log in.

Nisotak Teacher Tool
Manage Users
Manage Requests

Login

Email

Password

Login
Register

Figure 3.2 Login page

3.3 Home page

When the user logs in the platform, they see the home page (Figure 3.3). This page exhibits a list of all the existing lessons the user has access too - and the option to edit/delete them -, as well as a button that allows the user to add a new lesson. The table containing the lessons can be filtered by term and the columns can be organized in alphabetical order, ascending or descending.

It is important to note that deleting a lesson automatically deletes all the content associated with that lesson - content pages, multiple choice questions, texts/stories and phrases - and all the media related to them.

Nisotak Teacher Tool
Manage Users
Manage Requests

Home

Lessons list

Add Lesson

Filter table by term:

Title (Dialect)	Title (English)	Dialect	Options
ānisi kaki-isi-wiyasiwātamiḥk ka-siskōtahēciḥ awāsisak	Planning an Outdoor Field Trip (nēhiyawī)	nēhiyawī (Cree Y-dialect)	Edit Delete
Yā āt'ēēh abinī. Tj' nahgōō k'ad. Hasht'e' ī dīlīnēēh.	Good morning. Let's get ready. (Diné Bizaad)	Diné Bizaad (Navajo)	Edit Delete
Tf'óó'góó Áhoot'ēhigīi Binaḥjī' Ohoo'aah, Nihinaagóó Áhoot'ēhigīi Bina'niltin, Nahasdzāán Bikāa'gi líná Holónigīi Bina'niltin, Tf'óó'góó Binaḥjī' Ihwiidoo'āhigīi, Tf'óó'góó Naanish Binaḥjī' Ihwiidoo'āhigīi líná Baa Ákwiniidzin	Planning an Outdoor Field Trip (Diné Bizaad)	Diné Bizaad (Navajo)	Edit Delete
K'é	Kinship / Greetings. (Diné Bizaad)	Diné Bizaad (Navajo)	Edit Delete

Figure 3.1 Home page

3.4 Lesson page

The lesson page (Figure 3.4) allows the users to input:

- Information about the language:
 - Dialect
 - Knowledge source
 - Location

- Lesson details:
 - Level
 - Difficulty
 - Title (Indigenous language and English)
 - Topic (Indigenous language and English)
 - Description (Indigenous language and English)

Besides that, an audio file can be included to accompany the lesson description. In this page, the user also controls what we call lesson flow. The lesson flow is composed of content pages and multiple choice questions. The users can add as many of each as they wish and, in the lesson page, they inform the order they want these components to be displayed to the learner on the learner interface.

Nisotak Teacher Tool
Manage Users
Manage Requests

Home / Lesson

Lesson Page

Created by: m.koole@usask.ca

About the Language

Location:

Ministikwan

Dialect:

nēhiyawī (Cree Y-dialect)

Knowledge source:

Kevin Lewis

Lesson Details

Level:

Intermediate

Difficulty:

-

Title (Indigenous Language):

ānisi kaki-isi-wiyasiwātamiḥ ka-siskōtahēcik awāsisak

Title (English):

Planning an Outdoor Field Trip (nēhiyawī)

Topic (Indigenous Language):

wayawitamiḥ

Topic (English):

Outdoor activities

Description (Indigenous Language):

tānisi kā-ititwān "nimiwyēyihṭēn kā-(pimohṭēyān, sōskwāhtahikiyān, atoskēyān)". mistahi kikwaya kāki-itohcikāhtēwa wayawēhtimiḥ.

Description (English):

How to say "I like doing [something]". Many different kinds of activities that can be done outside.

Audio

Current file: description-nimiwyeyihten-ka.mp3

Download
Replace
Delete

Lesson Flow

The lesson flow is composed by content pages and questions, which can be combined and ordered freely.

New Content Page
New Question

Display Order	Name	Page Type	Options
1	nimiwyēyihṭēn kā-	Content Page	Edit Delete
2	sōniskwātahikē	Multiple Choice Question	Edit Delete

Save Lesson

Figure 3.4 Lesson page

3.5 Content page

The content page allows the users to add as many Text/Story and Phrases components as they wish to be part of that content page component. To do so, they can click on the buttons “Add Text/Story” and “Add Phrase”. When they do, the new component will be created and the user is redirected to the component page.

All the components included in the content page are displayed in a table. Using the buttons in the last column of the table, the user can opt to edit or delete the components. The user can order these components according to their desire, using the first column, indicating in which order this content should be presented to the learners in the Learner application. The application does not check if the numbers are unique, so it is important that the user pays attention to not include duplicate numbers.

Last, the users can enter a name to be displayed in the lesson table page, to help identify it, as the lesson can have more than one content page.

Display Order	Name	Type	Options
1	nimiywéyhtén ká- (ácimowinis)	Text/Story	Edit Delete
2	nimiywéyhtén ká-kwaskwépíckéyán	Phrase	Edit Delete
3	nimiywéyhtén ká-sisáwohtéyán	Phrase	Edit Delete
4	nimiywéyhtén ká-máciyán	Phrase	Edit Delete
5	nimiywéyhtén ká-sóniskwátahikéyán	Phrase	Edit Delete

Name (to be displayed on the lesson page table)

nimiywéyhtén ká-

Figure 3.5 Content page

3.6 Text/Story Content page

The Text/Story Content page allows the users to include text in an indigenous language and in English. The user can also opt to include an audio and an image file to accompany the text/story in the learner interface. The author can give a name to the content, which will be displayed in the table from the content page in which the text/story is included.

Nisotak Teacher Tool
Manage Users
Manage Requests

Home / Lesson / Content Page / Text/Story Content

Text/Story Content

Text/Story:
Indigenous Language

nimiwëyihthên kâ- (âcimowinîs)
awâsisak nêtê okiskinwahamâtowikamohk itwêwak nohtê-itohtamohk kîkway wayawêhtimiik.
êkosi awa okiskinwahamâkêw kwêcimêw. kâhkiyaw pâpîtos itwêwak!
Sue: nimiwëyihthên kâ- kwâkwêpicikîyân.

English

I like (story).
The kids at the school said they would like to go on an outdoor field trip. So, the teacher asked them what they would like to do. Everyone had a different idea!
Sue: I like fishing.

Audio
Current file: *planning_outdoor_field_trip.mp3*

Download Replace Delete

Image
Current file: *pack_for_fieldtrip.jpg*

Download Replace Delete

Text/Story name:

nimiwëyihthên kâ- (âcimowinîs)

Save Text/Story

Figure 3.6 Text/Story content page

3.7 Phrase Content page

The Phrase Content page, allows the authors to include phrasen broken into morphemes. For each morpheme the author can include its meaning in English. The position of the morphemes can be edited by the author, in case a mistake is made. Each phrase can also be accompanied by an audio and image file. The author can give a name to the content, which will be displayed in the table from the content page in which the phrase is included

Nisotak Teacher Tool
Manage Users
Manage Requests

Home / Lesson / Content Page / Phrase

Phrase

Morphemes

Position	Language: Indigenous	Language: English	Options
1	ni	I	Delete
2	miywëyihthên	like	Delete
3	kâ-	conjunct	Delete
4	kwaskwêpicikê	fish	Delete
5	yân	me	Delete

Add Morpheme

Phrase Media
Audio

Choose File No file chosen

Image

Choose File No file chosen

Phrase Name

nimiwëyihthên kâ-kwaskwêpicikêyân

Figure 3.7 Phrase content page

3.8 User Management page

This page can only be accessed by the system admins. All the users from the teacher interface are displayed in a table. The admins can use this page to change the access type of the users or delete them (i.e. revoke their access), by clicking on the appropriate button that appears in the column “Options”, seen in Figure 3.8.

It is important to note that deleting a user does not delete the content created by the user. This was done to enable users to recover access in case they lose their password. At the moment, the system does not have a way for the user to change or recover the password. The workaround is:

1. The admin deletes the user;
2. The user requests a new access using the same email used before;
3. The admin approves the access.

As the content is not deleted when the users are deleted, once these three steps are executed the users regain access to the content previously created by them.

If the admin wants to delete the user and the content created by that user, the lessons need to be manually deleted by either the editor or the admin.

E-mail	Username	Role	Options
lucascamargo755@gmail.com	lucasCamargo	admin	<button>Assign API Reader</button> <button>Assign Admin</button> <button>Assign Editor</button> <button>Delete User</button> <button>Assign Author</button>
barbaraabur@gmail.com	Barbara	admin	<button>Assign API Reader</button> <button>Assign Admin</button> <button>Assign Editor</button> <button>Delete User</button> <button>Assign Author</button>
m.koole@usask.ca	mkoollead	admin	<button>Assign API Reader</button> <button>Assign Admin</button> <button>Assign Editor</button> <button>Delete User</button> <button>Assign Author</button>
nikiron@usask.ca	kiron	editor	<button>Assign API Reader</button> <button>Assign Admin</button> <button>Assign Editor</button> <button>Delete User</button> <button>Assign Author</button>
gene80@gmail.com	gene80	editor	<button>Assign API Reader</button> <button>Assign Admin</button> <button>Assign Editor</button> <button>Delete User</button> <button>Assign Author</button>

Figure 3.8 User management page

3.9 Access Requests page

The Access Requests page displays a table containing all the open access requests to the teacher tool, as shown in Figure 3.9. The admin can use this page to approve the access, granting the appropriate user type.

E-mail	Username	Options
frtherez@yahoo.com	Mwhitstone	<button>Approve as API reader</button> <button>Approve as Editor</button> <button>Approve as Author</button> <button>Approve as Admin</button>

Figure 3.9 Access requests page

4 Microservices

Six different microservices are currently part of Nisotak Author Tool's backend. They are:

1. Access control.
2. Multiple Choice Question.
3. Content Page.
4. Text Content.
5. Phrase Content.
6. Lesson.

These services can be found in the *Microservices* folder, which is in the root of the Github project. This folder contains five other folders inside it. The *Access Control* folder contains the code for this microservice. The *Activities* folder is meant to include all microservices related to questions and other activities to test the learner's knowledge on the content presented in the lesson. For now, it contains only one folder, *Multiple Choice Question*, that contains the code for this service. The *Content Pages* folder is meant to include everything related to lesson content. For now, three folders are included on it. The *Pages* folder contains the services related to the content page itself. The *Phrases* and *Text* folders include the services for these two components that can be included in content pages. The *Lessons* folder contains the code for this microservice. Finally, the *Helper* folder does not contain any service, but has auxiliary functions used by the microservices.

An overview of each microservice can be seen in the next sections from this chapter. Sample calls to these endpoints can be seen in the Github project. There is a file in the root of the project named `Nisotak.postman_collection.json` that has API calls examples. Besides that, we used most of the endpoints that will be presented here in the author tool application. If you want to see them being applied in an actual application context, refer to the folder `author-tool-app` in the root of the project.

4.1 Access control

The access control service includes everything related to users' access. The user model can be seen in the file `model.js`, stored in the Access Control folder. For each user we store the username, email, password and role in the database. The email is the user's unique identifier. The password is encrypted before it is stored in the database. The roles available were explained in Chapter 2. The model, however, shows one extra possible role: "inactive". Any user that has requested to access the system will have the role set to inactive until it is approved or denied by a system admin and will not be able to interact with the system until the approval.

The Access Control service has six endpoints:

- `/register (POST)`: Used to submit access requests for new users. Receives username, email and password and saves the new user. Please note that only the email is verified for uniqueness. This means that there can be repeated usernames.
- `/login (POST)`: Returns the JWT that will be used to access the remaining services.

- `/users/inactive (GET)`: Used to get all inactive users (i.e. users that the admin has not approved yet). Only admins have access to this service.
- `/users/active (GET)`: Used to get all active users (i.e. users that the admin has not approved yet). Only admins have access to this service.
- `/users/updateRole (POST)`: Used to update the role of a user.
- `/users (DELETE)`: Used to delete a user. Please note that deleting a user does not delete the content created by that user.

4.2 Lesson

The Lesson service includes everything related to a lesson - but not the other components contained by the lesson. The lesson model can be seen in the file `model.js`, stored in the Lesson folder. For each lesson we store information such as the author - which is the user who created the lesson using the Author Tool -, title, and the lesson level and difficulty. The authors can opt to include an audio file to accompany the lesson. The path to this file is stored in the database and the file itself is stored in a folder in the server. Although the model includes an image object, it is not being used at this time, as the functionality to include images in a lesson was not implemented. Finally, It is important to highlight that the model includes a list called *pages*, that stores references to content pages and multiple choice questions which are part of that lesson. For each item in this list, the order of presentation chosen by the author, *pg_display_order*, is also stored. To see all the data stored for each lesson, please refer to the `model.js` file.

The following endpoints are available in this service:

- `/ (GET)`: Returns all the lessons the user who made the request is allowed to see.
- `/search (GET)`: Returns lessons filtered by a specific dialect or by a certain term. In case the search is being made by term, the fields that will be searched are title, topic and introduction, in both the Indigenous Language and English.
- `/:id (GET)`: Returns a lesson by id.
- `/add (POST)`: Creates a new lesson and returns its id.
- `/update/:id (POST)`: Updates a lesson by id.
- `/:id (DELETE)`: Deletes a lesson by id. Note that a call to this endpoint will not only delete the lesson itself, but all the content contained by the lesson (content pages and its components and multiple choice questions).
- `/update/:lessonid/pageRef (POST)`: Creates a reference to a page contained by the lesson.
- `/update/:lessonid/pageRef (DELETE)`: Deletes a reference to a page contained by the lesson.
- `/media/audio/:lessonId (GET)`: Downloads the audio file from a lesson.
- `/media/audio/:lessonId (DELETE)`: Deletes the audio file from a lesson.

4.3 Content Page

The Content Page service includes everything related to a content page - but not its components, there are separate services for text and phrase contents. The content page

model can be seen in the file `model.js`, stored in the `Content Page/Pages` folder. For each content page we store a name `pg_table_name` - which is a name for that content page created by the author. We also store a list called `content`, that contains references to phrases and texts owned by that content page. For each item in this list, the order of presentation chosen by the author, `display_order`, is also stored. Finally, we store a reference to the lesson that owns that content page.

The endpoints available in this service are:

- `/` (GET): Returns all the content pages the user is allowed to see.
- `/:id` (GET): Returns a content page by id.
- `/add` (POST): Creates a new content page and returns its id. A reference to this content page is added in the lesson it belongs to.
- `/update/:id` (POST): Updates a content page by id.
- `/update/:pageid/contentRef` (POST): Includes a reference to a specific content (phrase or text, for now) in the content page.
- `/update/:pageid/contentRef` (DELETE): Deletes a reference to a specific content (phrase or text, for now) in the content page.
- `/:id` (DELETE): Deletes a content page by id. Note that a call to this endpoint will not only delete the content page itself, but all the content contained by the page (texts and phrases). Moreover, it also makes a call to the lesson service that contains the page being deleted, to delete the reference to this page that exists in the lesson.

4.4 Text Content

The Text Content service was created to include everything related to this type of content, that is meant to allow authors to include stories in the content pages. The text content model can be seen in the file `model.js`, stored in the `Content Page/Text` folder. For each text content we store a name `cnt_table_name` - which is a name for that story created by the author. The authors can opt to include an audio and an image file to accompany the text content. The path to these files is stored in the database and the files themselves are stored in a folder in the server. Besides that, the author can write the story in an indigenous language and in English - `text_cree` and `text_english`. The `cnt_page_id` is a reference to the content page that contains the phrase.

The endpoints available at the Text Content service are:

- `/` (GET): Returns all the text content the user is allowed to see.
- `/:id` (GET): Returns a text content by id.
- `/add` (POST): Creates a new text content and returns its id. A reference to this text is added in the content page the text belongs to.
- `/update/:id` (POST): Updates a text content by id.
- `/:id` (DELETE): Deletes a text content by id. A call to this endpoint will make a call to the content page service that contains the text being deleted, to delete the reference to this text that exists in the content page.
- `/media/audio/:textid` (GET): Downloads an audio file from a text content.
- `/media/image/:textid` (GET): Downloads an image file from a text content.

- `/media/audio/:textid (DELETE)`: Deletes an audio file from a text content.
- `/media/image/:textid (DELETE)`: Deletes an image file from a text content.

4.5 Phrase Content

The Phrase Content service was created to include everything related to this type of content, that is meant to allow authors to include words or phrases broken into morphemes to the content page. The phrase content model can be seen in the file `model.js`, stored in the Content Page/Phrases folder. For each phrase content we store a name `cnt_table_name` - which is a name for that phrase created by the author. The authors can opt to include an audio and an image file to accompany the phrase content. The path to these files is stored in the database and the files themselves are stored in a folder in the server. Besides that, the phrases model include a list of morphemes. For each morpheme, the author can include its English translation (`morph_english`) and define its position in the phrase. The option to include a position was added to allow the authors to easily change the order of the morphemes from the phase, they make a mistake while inputting the data. The `cnt_page_id` is a reference to the content page that contains the phrase.

The endpoints available at the Phrase Content service are:

- `/ (GET)`: Returns all the phrase content the user is allowed to see.
- `/:id (GET)`: Returns a phrase content by id.
- `/add (POST)`: Creates a new phrase content and returns its id. A reference to this phrase is added in the content page the phrase belongs to.
- `/update/:id (POST)`: Updates a phrase content by id.
- `/:id (DELETE)`: Deletes a phrase content by id. A call to this endpoint will make a call to the content page service that contains the phrase being deleted, to delete the reference to this phrase that exists in the content page.
- `/media/audio/:phraseid (GET)`: Downloads an audio file from a phrase content.
- `/media/image/:phraseid (GET)`: Downloads an image file from a phrase content.
- `/media/audio/:phraseid (DELETE)`: Deletes an audio file from a phrase content.
- `/media/image/:phraseid (DELETE)`: Deletes an image file from a phrase content.

4.6 Multiple Choice Question

The Multiple Choice Question model can be seen in the file `model.js`, stored in the Activities/Multiple Choice Question folder. For each question, we store a name `pg_table_name` - which is a name for that question created by the author - and the question instructions both in the indigenous language and in English. An audio file can be added by the author to accompany the instructions. The path to this file is stored in the database. Besides that, the model contains a list `options`, which represents the possible answers to the question created by the author. For each answer option, the database stores its text in an indigenous language and in English and `is_correct` stores if that answer is the correct option or not. Besides that, for each answer the author can also opt to include an audio file and the path to this file is also stored in the database, inside the option object. All the audio files are

stored in the server. Finally, we store a reference to the lesson that owns that multiple choice question.

The endpoints available at the Multiple Choice Question service are:

- / (GET): Returns all the multiple choice questions the user is allowed to see.
- /:id (GET): Returns a multiple choice question by id.
- /add (POST): Creates a new multiple choice question and returns its id. A reference to this multiple choice question is added in the lesson the question belongs to.
- /update/:id (POST): Updates a multiple choice question text content by id. It is also possible to send an audio file to be attached to the question. Obs: Although it's possible to update text for answer options through this call, it's not possible to update answer option's audio files - see /update/:questionid/answerOption.
- /:id (DELETE): Deletes a multiple choice question by id. A call to this endpoint will make a call to the lesson service that contains the question being deleted, to delete the reference to this multiple choice question that exists in the lesson.
- /media/audio/:questionId (GET): Downloads an instruction audio file from a multiple choice question.
- /media/audio/:questionId (DELETE): Deletes an instruction audio file from a multiple choice question.
- /update/:questionid/answerOption/ (POST): Updates an answer option for a multiple choice question. It also receives an optional audio file to be attached to the answer option.
- /media/audio/:questionId/:optionId (GET): Downloads an audio file from an answer option in a specific question.
- /media/audio/:questionId/:optionId (DELETE): Deletes an audio file from an answer option in a specific question