The Development Process of eLearning Application for Configurations of the Operating Room

Daria-Elena Melinte
Faculty of Computer Science "Alexandru Ioan Cuza" University Iasi, Romania
dariamelinte2003@gmail.com

Adrian Iftene
Faculty of Computer Science "Alexandru Ioan Cuza" University Iasi, Romania
adiftene@gmail.com

ABSTRACT
The application of 2D and 3D modeling has expanded across various domains, encompassing interior and exterior design, eLearning games and applications, animations, and futuristic advertisements. This work explores the development of an eLearning application designed to evaluate bioengineering students and assist them in preparing an operating room using 3D models. The findings of the exploration have led to the development of an application characterized by a multi-role framework. Within this framework, educators are afforded the capability to manage multiple groups of students, as well as to design and administer assessments that evaluate students' knowledge. Concurrently, students have the opportunity to enroll in multiple groups and complete the assessments provided by their instructors. Furthermore, both educators and students are granted access to a "playground" feature, which serves as a training or learning environment. This feature allows users to visualize the operational environment they are working within from various perspectives, thereby enhancing their understanding and interaction with the content.

Author Keywords
Medical application; 3D Models; eLearning

ACM Classification Keywords
H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces. H.3.2. Information Storage and Retrieval: Information Storage.

General Terms
Human Factors; Design; Measurement.
DOI: 10.37789/ icusi.2024.24

INTRODUCTION
The medical field, vital for saving countless lives daily, intersects with informatics to enhance the capabilities of healthcare professionals. Nurses, who play a crucial role in patient care and operating room preparation, face challenges in training due to limited access to these busy environments. To address this, computer science can offer a virtual system for both students and healthcare professionals to practice room setup for various procedures [1]. This work proposes the development of such a platform, enabling users to learn and be assessed through tests designed by instructors, ultimately improving their preparedness for real-world scenarios.

MEDICAL OR EDUCATIONAL APPLICATIONS THAT EMBED 3D SPACE

Incision
Incision[1] is a successful application in the medical field, designed to help hospital staff learn and refresh protocols. It offers features like human anatomy education and immersive 3D visualization of an operating room (see Figure 1). Widely used in healthcare centers, it is not available for personal use, as individual accounts must be linked to a hospital subscribed to the service.

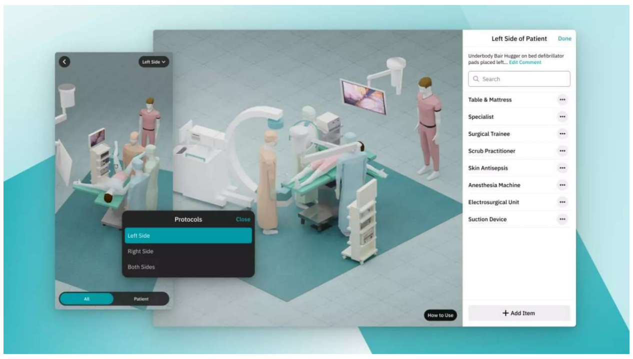


Figure 1. Incision[2].

CoSpaces Edu
CoSpaces Edu[3] is an e-learning platform that enables educators to create and share interactive 3D and VR environments for immersive learning. It includes tools for managing students, assigning online tasks, and monitoring their work (see Figure 2). However, creating assignments requires coding experience. The platform offers both free and paid options.

STATE OF THE ART
After analyzing the existing medical and educational applications, we can say that no application currently offers comprehensive training for operating room setup, underscoring the need for this development. While various tools support medical training and 3D space interaction, the proposed virtual system would fill a crucial gap by providing specialized training for nurses and healthcare professionals in this area. The proposed application would

[1] https://www.incision.care/
[2] https://www.incision.care/products/assist
[3] https://www.cospaces.io/

allow professors to manage multiple students, create assignments and tests, and enable students to practice operating room setups for various scenarios.

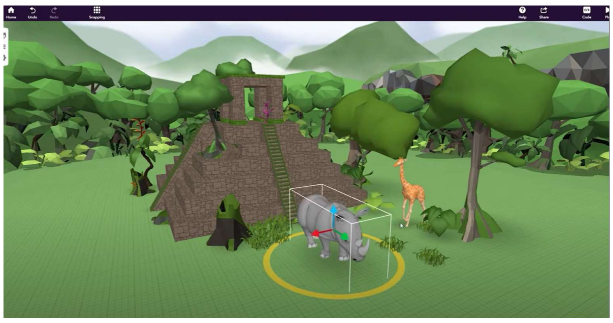


Figure 2. CoSpaces Edu[4].

A crucial feature would be a 3D space where users can interact with objects, utilizing Blender's visualization format for clarity and ease of understanding.

In developing a training application for the Bioengineering Faculty of Iași, key considerations include identifying the target users and ensuring intuitive engagement. Although AR/VR technology offers immersive learning, it's limited by the overcrowding of operating rooms and the lack of AR/VR equipment for all students. Instead, a web application was chosen over a native app because it can be accessed from any location, effectively addressing the overcrowding issue and providing a satisfactory user experience with proper UI/UX design.

APPLICATIONS ACTORS

To better understand the application that was created, it is necessary to understand who it interacts with and how. We will describe the entities that the application interacts with, along with the actions that can be taken in the following sections. The built application interacts both with human users and with different software systems. we can therefore define the following actors with which the application interacts:

- the **human user**, who can be an *admin*, *student* or *teacher*;
- the **software system that maintains the database** (MongoDB);
- **the software system in which the 3D models are stored** to be used later in the application (Digital Ocean).

Depending on the role of each actor, some actions will be defined that they can do within the application. We will describe below the actions that each actor can do.

The Admin

The administrator manages all aspects of the application, including user support. Admin functions include logging in/out, profile management, and full control over 3D model categories and models. Admins can also create, view, and delete groups and tests, manage test submissions, and configure 3D environments in the playground.

The Professor

Professors can create and manage their accounts and profiles. They view and organize 3D objects and categories, create and manage groups, and design tests. Professors can view their tests and solutions, but can only update test solutions before the submission deadline. They also interact with the playground for 3D model configurations.

The Student

Students can manage their accounts and profiles, view 3D objects and categories, and join or leave groups. They can view and interact with assigned tests, create and edit solutions within the allowed times, and access the playground for practice and exploration.

MongoDB Atlas Cluster

MongoDB [2] Atlas Cluster provides a NoSQL Database-as-a-Service, handling data storage and security. It connects to the application, validates requests, and protects database content from unauthorized access.

DigitalOcean Space

DigitalOcean Space offers scalable object storage for 3D models, with S3 compatibility and built-in Content Delivery Network (CDN) [3]. It securely stores and manages model files, ensuring that only the application can access and manipulate them.

In designing a complex application, it is crucial to understand the needs and roles of each user. The next section will detail the system architecture and implementation to meet these needs (see Figure 3).

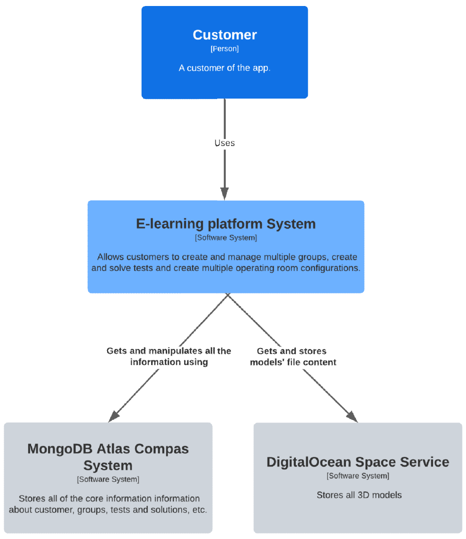

Customer
[Person]
A customer of the app.
Uses
E-learning platform System
[Software System]
Allows customers to create and manage multiple groups, create and solve tests and create multiple operating room configurations.
Gets and manipulates all the information using
Gets and stores models' file content
MongoDB Atlas Compas System
[Software System]
Stores all of the core information information about customer, groups, tests and solutions, etc.
DigitalOcean Space Service
[Software System]
Stores all 3D models

Figure 3. System Context Diagram.

[4] https://www.youtube.com/watch?v=lyWE9CAJwIg\&ab_channel=CoSpacesEdu

ARCHITECTURE AND IMPLEMENTATION

The following sections will cover the database, service, and application architectures, detailing their usage, architectural decisions, and key features.

Database Architecture

The database architecture is crucial for managing 3D operating room configurations. A NoSQL database with collections such as *Object Instance*, *Linkage*, *Object Model*, and *Category* supports storing and manipulating object coordinates. Collections like *User*, *Credential*, and *Group* manage user data and group memberships, while *Test* and *Result* collections handle test creation and student submissions (see Figure 4).

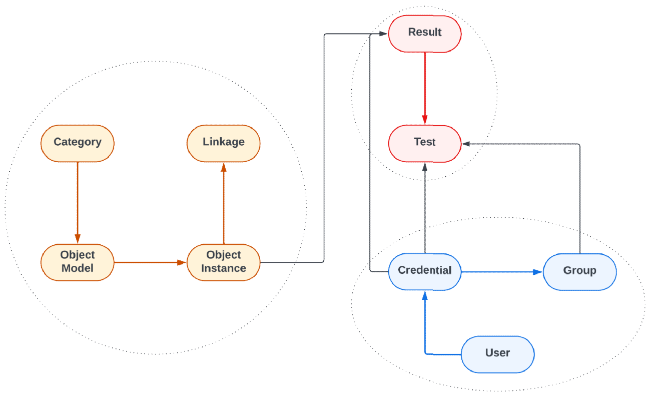


Figure 4. Database Models Diagram.

The *Credential* collection stores user authentication details, including a unique *ID* or *email*, *password* hash, and *authentication token*. It ensures secure access by validating credentials. The *User* collection contains personal information about users, identified by *ID* or *credential ID*. Fields include *role* (admin, student, professor), *name*, *phone number*, and *institution*.

The *Group* collection stores information about groups created by professors, identified by *ID* and *token*. It includes the *professor's credential ID* and a *list of student credential IDs*, eliminating the need for additional queries. Each test that a professor creates will be stored as a document in the *Test* collection, which holds documents for tests created by professors, including *test ID*, *name*, *description*, and reference documents for *credentials* and *groups*. It includes fields for *minimum and maximum scores*, *start and due dates*, and *test status*. The collection *Result* stores documents representing test solutions, including correct solutions and student submissions. It features *test ID*, *user credential*, *instance list, submission time, scale, and score*. The instances referenced in the *Result* collection are stored in the *Object Instance* collection. It manages 3D object instances with unique *IDs* and *UUIDs*, including *model* references and *positions* stored in JSON format for precise spatial operations.

A *model* is a 3D object. It can be created in any tool that has this purpose and is going to be used in the application in an exported version of type *.fbx*. This is why each document from the *Object Model* collection is going to contain information about 3D models used in the application, including *category*, *name*, *description*, and *model size*. Models are stored separately in DigitalOcean Space. The *Category* collection defines object model categories with unique *IDs* and category *names*.

The last collection, *Linkage*, stores links between *test instances*, including connection details such as *instance references, box points,* and *coordinates*, necessary for understanding surface connections between objects.

Back-End Architecture

Model View Controller (MVC) Design Pattern in a REST API We implemented a REST API using the MVC design pattern [4] to ensure a robust and scalable platform. Leveraging Flask, we created an API that is flexible, easy to maintain, and capable of growth. The API's structure is organized into key components:

- The **app**, which is the core initializes the routes and establishes the database connection.
- The **routes**, which are defined by their names, accepted parameters, and responses, routes handle specific requests and invoke corresponding services.
- The **services**, that manage the operations and methods that interact with the models, perform the core functionality of the API.
- Each **model** defines the schema for documents stored in their respective collections, ensuring data is structured and accessible.

API routes

The API supports various routes for creating, reading, updating, and deleting documents. To manage user-specific content access, we use authentication middleware with encrypted access tokens (HS256). Each token contains user information and an expiration date. Requests to protected routes are validated via token decoding; invalid tokens halt the request process. The API is organized into nine categories: *categories*, *object models*, *object instances*, *links*, *results*, *tests*, *groups*, *users*, and *authentication*. Each category includes routes for document management and additional functions like uploading models or joining groups. The authentication category handles registration, login, logout, and profile management.

API Services

The API services handle interactions with models and return results processed by routes. For example, the group service includes functions for *adding a group* (receives input data, creates a new group instance, and saves it to the database, managing exceptions as needed), *receiving groups* (retrieves group data based on filters and credentials, dynamically constructing queries and returning the data or error messages), *updating a group* (updates group information based on an identifier and new data, fetching and returning the updated group while handling exceptions), *deleting a group* (finds and deletes a group by its identifier, verifying the existence and managing exceptions), or *joining a group* (adds a student to a group using a group code, checks for group existence and student enrollment, updates the group, and handles exceptions). Each service manages requests, processes data through model interactions, and returns appropriate responses based on the operation's success or failure.

API Models

The API uses nine collections, each defined by a class that inherits from MongoEngine's Document class: *Category*, *ObjectModel*, *ObjectInstance*, *Linkage*, *User*, *Credential*, *Group*, *Test*, and *Result*. For instance, the *ObjectModel* class represents the document structure for the Object Model collection, managing data validation, integrity, and business logic. This aligns with the MVC architecture, where models handle data and related operations.

Tests' Solution Score

A critical aspect of the application is the automatic evaluation of student-submitted solutions. Formally, a solution is represented as a graph, where nodes are object instances and arcs are constraints defining object positioning in 3D space [5]. The evaluation process involves comparing two graphs: the student's solution and the correct model. This comparison yields a similarity score between 0 and 1. The algorithm evaluates the graphs by comparing nodes and arcs and applying penalties. We will detail the methods used for this comparison and penalty application (see Figure 5).

Each pair of instances from the two graphs is compared to obtain a similarity index. This index is based on the alignment of positions on each axis, with a threshold allowing for approximate matches. A penalty is applied if the instances do not match. The resulting score is normalized between -0.25 and 1 by dividing by the maximum possible score. Finally, the individual similarity indices are aggregated to produce an overall similarity score for the entire set of instances.

The evaluation of linkages follows a similar approach. Linkages are assessed based on the coordinates of connection points in 3D space and relative to each instance. The comparison involves checking these coordinates across all three axes and evaluating the angle differences between the slopes formed by the connection points. A maximum score of 2 points is given if the angle difference is less than 10 degrees, with decreasing scores and penalties for larger differences. The similarity score for linkages ranges from -0.17 to 1 and is aggregated into a single value. Normalization is applied to account for different scales in coordinates. Additionally, the number of nodes and arcs is compared, with penalties for discrepancies. The final similarity score, between -1 and 1, is calculated as a weighted average of instance and linkage scores, with penalties applied as needed. The algorithm returns a final score between 0 and 1.

The similarity score is computed for each submitted result, including the original solutions. The score for a test is calculated using the following formula:

$$score = similarityScore/originalSimilarityScore \times 100$$

In conclusion, the algorithm effectively compares results by focusing on geometric and spatial relationships using several design techniques:

- *Greedy decisions*: it uses Euclidean distance for positional similarity, making local comparisons based on proximity and normalizing positions by scale.
- *Divide and Conquer*: the algorithm breaks down the problem into smaller tasks—comparing instances and linkages separately—before aggregating the results for a final similarity score.
- *Weighted scoring*: different components (model matching, positional accuracy, angle differences) are weighted to reflect their importance in the overall score.
- *Penalty-based adjustments*: discrepancies, such as mismatches in the number of instances or linkages, incur penalties, adding a logical layer to the assessment.

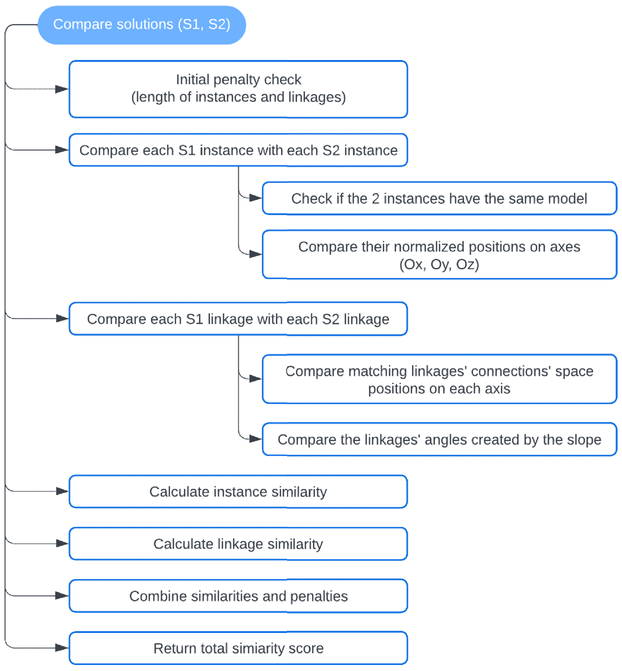

Compare solutions (S1, S2)
Initial penalty check (length of instances and linkages)
Compare each S1 instance with each S2 instance
Check if the 2 instances have the same model
Compare their normalized positions on axes (Ox, Oy, Oz)
Compare each S1 linkage with each S2 linkage
Compare matching linkages' connections' space positions on each axis
Compare the linkages' angles created by the slope
Calculate instance similarity
Calculate linkage similarity
Combine similarities and penalties
Return total simiarity score

Figure 5. Comparison algorithm Diagram.

Front-End Architecture

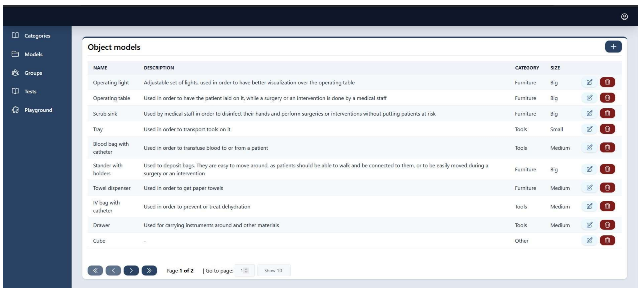
This section describes the key modules and components of the front end, which encapsulate all the functionalities detailed in this paper.

Authentication and User's Profile Modules

Upon first accessing the application, users are greeted with a landing screen that offers authentication options. They can either log in or register. Initially, users cannot access most features without creating a profile. The profile page allows users to create or update their profile information, enabling full interaction with the application's features.

Categories and Object Models Modules

The core concept of the application involves arranging and evaluating an operating room using 3D objects. Object models and their categories are managed by an admin role, which handles their creation, visualization, and modification. The admin can view, update, and delete both categories and object models on dedicated pages, where they are listed in tables (see Figure 6).



Object models

NAME	DESCRIPTION	CATEGORY	SIZE
Operating light	Adjustable set of lights, used in order to have better visualization over the operating table	Furniture	Big
Operating table	Used in order to have the patient laid on it, while a surgery or an intervention is done by a medical staff	Furniture	Big
Scrub sink	Used by medical staff in order to disinfect their hands and perform surgeries or interventions without putting patients at risk	Furniture	Big
Tray	Used in order to transport tools on it	Tools	Small
Blood bag with catheter	Used in order to transfuse blood to or from a patient	Tools	Medium
Stander with holders	Used to deposit bags. They are easy to move around, as patients should be able to walk and be connected to them, or to be easily moved during a surgery or an intervention	Furniture	Big
Towel dispenser	Used in order to get paper towels	Furniture	Medium
IV bag with catheter	Used in order to prevent or treat dehydration	Tools	Medium
Drawer	Used for carrying instruments around and other materials	Tools	Medium
Cube	-	Other	

Page **1 of 2** | Go to page: 1 Show 10

Figure 6. Object Models' Page.

It is also worth mentioning that all the models presented and used in the application were personally made in Blender, and are further used in a *.fbx* format. While designing and creating them, we did some research to understand what is the most common form of these objects in the hospital (see Figure 7 for examples).

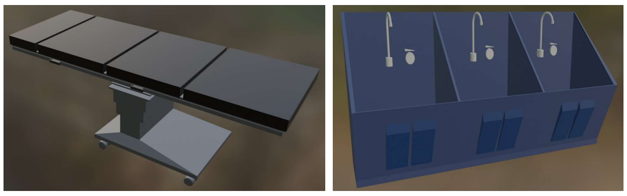


Figure 7. Operating Table (on the left), Scrub Sink (on the right)

Groups Module

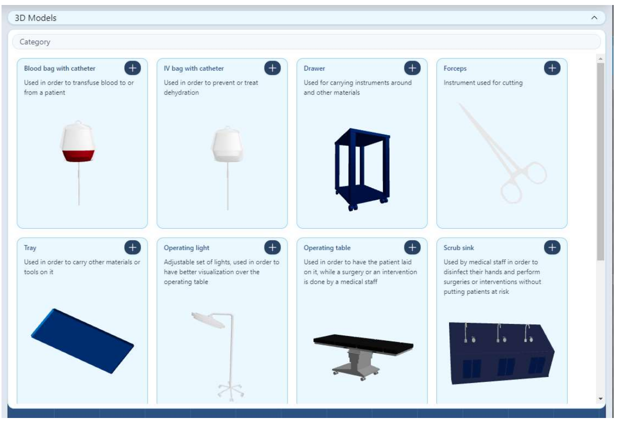
The *groups' module* is displayed almost the same for students and professors, each of them having some extra features. Both students and professors can visualize a table. The difference is that students only view the groups they are in, and can join a group, whereas professors can create, update, and delete a group, but they cannot join one. *A group's page* consists of two main components: (1) viewing information about the group, the access code to the group, and (2) the coordinating teacher, and viewing the students in a group. The student can also *leave the group* from that page, and the professor can *remove a student* from the group.

Playground Module

The playground module consists of a place where the user can play with a board, and interact with it, to see how object instances would move in 3D space, and what would the links between the instances look like. The playground can also be a place where students can practice on different configurations of an operating room. To create the playground, we created three main components:

- a menu for visualizing and selecting the models,
- a board for visualizing and manipulating the instances and linkages,
- and a footer, for scaling the board and changing the visualization perspective.

Models' menu is expandable and displays the available object models (see Figure 8). Each card from the menu displays the *3D model* and lets you manipulate the model as you want. Because of this, you can rotate, scale, and move the object, to visualize it better. The card also offers information about the object, such as its *name*, and a small *description*. The card also has a *button* that lets you *add the model to the board*. Besides better visualization, the menu allows you to filter the objects, based on their category.



Blood bag with catheter
Used in order to transfuse blood to or from a patient

IV bag with catheter
Used in order to prevent or treat dehydration

Drawer
Used for carrying instruments around and other materials

Forceps
Instrument used for cutting

Tray
Used in order to carry other materials or tools on it

Operating light
Adjustable set of lights, used in order to have better visualization over the operating table

Operating table
Used in order to have the patient laid on it, while a surgery or an intervention is done by a medical staff

Scrub sink
Used by medical staff in order to disinfect their hands and perform surgeries or interventions without putting patients at risk

Figure 8. Models' Menu Component.

The **Board** component displays instances added from the models' menu and their connections (see Figure 9). It features a grid system with a central red point for spatial orientation. A button allows users to *delete all instances and links*, simplifying the process of starting anew. Instances can be repositioned on the board, with their top-left coordinates stored to preserve their state. Each instance has a point layer for creating links between objects. A button on each instance enables the deletion of both the instance and its links. Links are created by selecting *connection points* on different objects; links between the same objects are not allowed. Each link records the coordinates of its connection points for accurate evaluation and display. Links can also be removed by clicking on them.

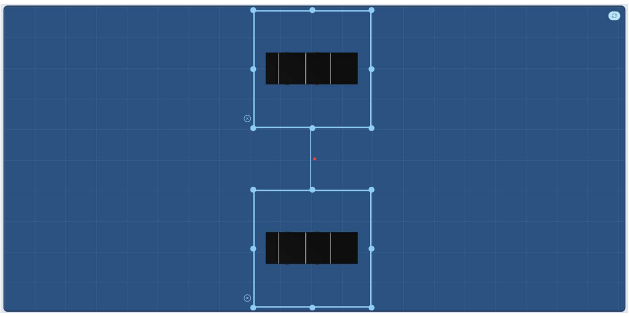
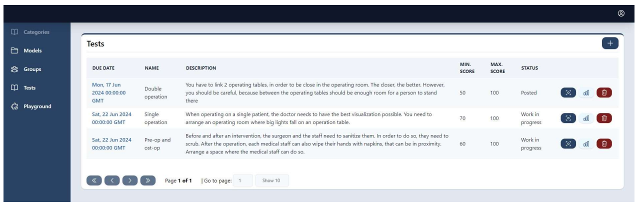


Figure 9. Board Component.

The **Footer** has two key components: the *scale* setting and the *perspective* setting. By default, the board's scale is set to 1, but it can be adjusted for better visibility and maneuverability as more components are added. Changing the scale recalculates the positions of all components and links to fit the new scale. The perspective component includes three buttons for viewing the board from different angles: *Top* (Oz axis), *Side* (Ox axis), and *Front* (Oy axis).

Test Module

We established before that one of this application's scopes is to enable professors to create tests and let students solve them. The test module was created around these key functionalities. When entering the *tests* page (see Figure 10) both students and professors can visualize the tests that are assigned to their group, respectively the tests that were created by them. A core difference is, though, the fact that students can only *solve those tests*, whereas the professors can *create, view, update, and delete tests*, as well as *see the solutions submitted by students*.



Tests

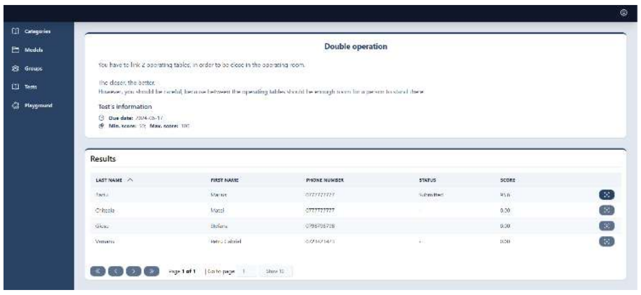
DUE DATE	NAME	DESCRIPTION	MIN. SCORE	MAX. SCORE	STATUS
Mon, 17 Jun 2024 00:00:00 GMT	Double operation	You have to link 2 operating tables, in order to be close in the operating room. The closer, the better. However, you should be careful, because between the operating tables should be enough room for a person to stand there	50	100	Posted
Sat, 22 Jun 2024 00:00:00 GMT	Single operation	When operating on a single patient, the doctor needs to have the best visualization possible. You need to arrange an operating room where big lights fall on an operation table.	70	100	Work in progress
Sat, 22 Jun 2024 00:00:00 GMT	Pre-op and ost-op	Before and after an intervention, the surgeon and the staff need to sanitize them. In order to do so, they need to scrub. After the operation, each medical staff can also wipe their hands with napkins, that can be in proximity. Arrange a space where the medical staff can do so.	60	100	Work in progress

Page **1 of 1** | Go to page: 1 Show 10

Figure 10. A Professor's Table of Tests.

If a professor wants to create a test, they are redirected to a separate page, where they complete the test's information, as well as an *official solution*. The test information and solution are separate activities, the first being a form displayed through a modal. Creating a test can be very complex, so the professors can save the progress, and submit the test to the students later. This is possible by setting the status of a test in *Work in Progress* or *Posted*.

It is worth mentioning, though, that a test cannot be created only by submitting the information and having an empty board. The professor can also visualize a test, and which students submitted their solution. This is done on a separate page, that presents the test's information, and a ranking table, that displays the students enrolled in the group for which the test was created, and their scores (see Figure 11).



Double operation

You have to link 2 operating tables, in order to be close in the operating room.

The closer, the better.
However, you should be careful, because between the operating tables should be enough room for a person to stand there.

Test's information

Due date: 2024-06-17
Min. score: 50 Max. score: 100

Results

LAST NAME	FIRST NAME	PHONE NUMBER	STATUS	SCORE
[illegible]	Marius	0777777777	Submitted	95.6
[illegible]	Matei	0777777777	-	0.00
[illegible]	Stefania	0798798798	-	0.00
[illegible]	Petru Gabriel	0721421421	-	0.00

Page **1 of 1** | Go to page: 1 Show 10

Figure 11. A test's ranking page.

A professor can also access each solution, to see how the student solved the test. The board that shows the solution is, however, disabled, to not let anyone except the actual student work on their test.

Store Architecture

To maintain and make the application function correctly, we need to keep track of the state of multiple variables and display the correct information accordingly. This is possible by creating a store that takes care of each of the modules presented in this section. The store was created using Zustand [6]. Because the size and complexity of the store are pretty big, we have fragmented it into multiple store slices: *dialog store*, *category store*, *object model store*, *authentication store*, *group store*, *test store*, *result store*, and a *playground store*.

Movement and Perspective over the Board Component

The browser consists of a 2D space, in which the (0, 0) coordinate point is the leftmost up point. The Ox axis is the width of the browser window, and the Oy axis is the length of the window. It does not, by itself, support 3D representations, but multiple libraries have been built to access this part, including the Three.js library [7], which we used in this application. With the library, 3D objects can be moved, rotated, scaled, etc. The problem with using it, however, is that when a 3D space is created, no higher-level layouts, consisting of elements such as divs, etc., can be inserted into it. This was problematic, as one of the needs of the application was to draw links between objects. For this reason, we tried to reproduce the most important functionality of the library, in the browser space, leaving the library only to handle the correct display of the models. We were thus able to create a space in which objects can be moved, scaled, and viewed from multiple perspectives, with the bonus of being able to draw connections between the objects in question. This was difficult, however, as we had to save the coordinates of an object in the 2D space of the browser, but also correlate with the view of the same object in the general 3D space. For this reason, both the instances and the links between them have saved positions in 3D space, but relative to the 2D space we are in, the one defined by the browser. In this way, when the perspective is changed, the object is repositioned to the point stored for the corresponding axis. We have thus managed to transform a 3D coordinate, made up of (*x*, *y*, *z*) points, into an independent structure. The structure defined as *PointType* stores the coordinates of the space defined by the browser. However, geometric dependencies must be maintained for the location of an object to be accurate. For this reason, we have analyzed how the coordinates of an object behave depending on the perspective from which it is viewed.

BOARD DEVELOPMENT AND DISCOVERIES

Three.js is widely regarded as a premier JavaScript library for rendering 3D content on the web due to its seamless integration with WebGL, a powerful API built on the OpenGL standard. WebGL facilitates the rendering of intricate 3D graphics directly within web browsers, leveraging the GPU for demanding tasks without requiring external plugins. Three.js abstracts the complexities of WebGL, making the development of interactive 3D applications more accessible to developers, even those lacking extensive knowledge of WebGL or OpenGL. The robust features of OpenGL, such as efficient rendering pipelines and broad support for graphical operations, underlie WebGL and, by extension, Three.js. However, while Three.js offers significant advantages in terms of performance and ease of use, it also presents certain limitations, particularly when integrating HTML elements within the 3D rendering environment.

During the development of an application aimed at enabling users to establish correlations between 3D objects on a board, a critical limitation of Three.js emerged: the inability to embed HTML elements, such as *<div>* tags, within a *<canvas>* layer. This restriction necessitated a workaround to maintain the required interactivity and correlation between objects. Consequently, the development team had to "rebuild" the Three.js movement system to achieve functionalities such as perspective viewing, scaling, and dragging of 3D elements. The dragging functionality was implemented using the React library `react-draggable`, which facilitated the creation of a board where elements could be moved interactively. For scaling and displaying the objects from various perspectives, the team applied geometric principles to accurately calculate the positions and sizes of all components, resulting in a rudimentary yet functional version of Three.js that primarily utilized the library for rendering the 3D models from the correct perspective as indicated on the board. To address the challenges of integrating 2D and 3D spaces, the development team devised a method for storing and manipulating object coordinates in a hybrid structure that accounts for both the 2D browser space and the 3D environment. A custom coordinate system was defined, where each position in 3D space is represented by a *PositionType* structure, comprising three *PointType* objects (*ox*, *oy*, *oz*) that store the x and y coordinates relative to each axis *(X, Y, Z)* in the browser's 2D space. This design allowed for accurate repositioning of objects when the perspective changes, by referencing the corresponding 2D coordinates for each axis.

The relationship between 3D and 2D coordinates was further refined by analyzing how object positions behave when viewed from different perspectives. As illustrated in Figure 12, each axis (X, Y, Z) has its mapping to the 2D space, where the browser's x and y coordinates correspond to different dimensions in the 3D space.

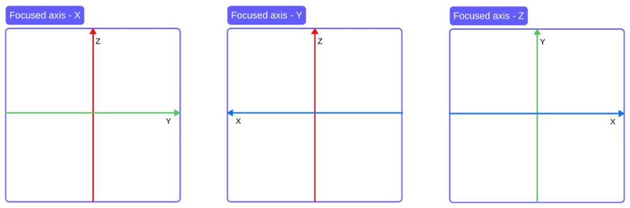


Figure 12. Perspectives of the board relative to the three axes.

For example, when the X axis is the focus, the coordinates from *ox* are used, with the 3D space's z-coordinate corresponding to the browser's y-coordinate, and the 3D y-coordinate corresponding to the browser's x-coordinate. Similarly, specific transformations were defined for when the Y or Z axis is the focus, as detailed in Figure 13.

Focused axis	3D equivalent of X	3D equivalent of Y
Ox	Y	Z
Oy	X	Z
Oz	X	Y

Figure 13. Equivalence of 3D coordinates in browser's 2D space.

We need to further define

$\Delta x = newX - oldX;\ \Delta y = newY - oldY$

When X is the focused axis and the browser's 2D coordinates x and y are updated, then we also need to update the following 2D coordinates:

- $Oy\colon oy.y = oy.y - \Delta y$
- $Oz\colon oz.y = oz.y - \Delta x$ Idem, when Y is the focused axis:
- $Ox\colon ox.y = ox.y - \Delta y$
- $Oz\colon ox.x = ox.x + \Delta x$

Likewise, when Z is the focused axis:

- $Ox\colon ox.x = ox.x - \Delta y$
- $Oy\colon oy.x = ox.x + \Delta x$

Defining and implementing these relationships, we were able to create an accurate movement on the board, that updates the positions of all perspectives when an item is moved. The custom implementation developed to overcome the limitations of Three.js in integrating 3D object manipulation within a 2D browser environment represents a significant advancement in Human-Computer Interaction. By creating a hybrid coordinate system that accurately correlates 2D and 3D spaces, and by implementing intuitive interaction techniques such as drag-and-drop using *react-draggable*, the approach enhances the usability and accessibility of 3D content manipulation for a broader range of users. This system not only ensures consistent object positioning across different perspectives but also reduces the cognitive load associated with 3D interactions, contributing to a more seamless and intuitive user experience. The improvements brought by this approach underscore the importance of flexible, user-centered design in HCI, particularly when standard tools fall short, and pave the way for future developments that could further refine and extend these interaction capabilities.

CONCLUSION

The final application created is a comprehensive e-learning platform with multiple modules and functionalities, including the management of student groups by teachers, the creation and evaluation of tests, and the development of a 3D space where both students and teachers can practice configuring operating rooms. The application's greatest complexity lies in managing the elements within the 3D space and evaluating test solutions, but all modules work together to create a cohesive educational tool. This platform allows students to learn how to arrange an operating room without physically being in one, while also helping teachers manage and evaluate their student groups more efficiently.

In the current version, the application board can display an operating room from only three perspectives. Future improvements include expanding the view to allow users to explore the board from all possible perspectives and enabling users to draw links between any two points on an instance rather than being limited to specific connection points. Additional functionality, such as rotating instances, incorporating specific operating rooms, and developing correlation rules between objects to facilitate linking within objects, would further enhance the application's educational capabilities. This would allow for more complex placement rules, not only between objects within the operating rooms but also about the operating rooms themselves.

In conclusion, the custom implementation of the 3D manipulation system represents a significant advancement in HCI within the e-learning context. By addressing the limitations of existing tools and developing a user-centered design, the application enhances the usability and accessibility of 3D content manipulation, ensuring consistent object positioning across different perspectives and reducing the cognitive load for users. This approach underscores the importance of flexibility and user-focused design in HCI, setting the foundation for future enhancements that could further refine these interaction capabilities and contribute to more effective educational experiences.

REFERENCES

1. E.E. Opait, D. Silion, A. Iftene, A., C. Luca, C. Corciova. (2024). "Mixed Realities Tools Used in Biomedical Education and Training", In Proceedings of the 18th International Conference on INnovations in Intelligent SysTems and Applications (INISTA 2024), 4-6 September 2024, Craiova, Romania
2. MongoDB, Inc. (2024). "What Is MongoDB?", https://www.mongodb.com/company/what-is-mongodb
3. DigitalOcean, LLC. (2024). "DigitalOcean. Spaces" https://www.digitalocean.com/products/spaces
4. GeeksForGeeks (2024). "MVC Design Pattern", https://www.geeksforgeeks.org/mvc-design-pattern/
5. SplashLearn, (2024). "Three Dimensional Shapes (3D Shapes)- Definition, Examples", https://www.splashlearn.com/math-vocabulary/geometry/3-dimensional
6. Zustand, (2024). "Introduction. How to use Zustand", https://docs.pmnd.rs/zustand/getting-started/introduction
7. B. Simon. (2024). "Become a Three.js developer", *Three.js Journey* (2024) https://threejs-journey.com/