CASE-BASED SIMULATION

Design and Architecture Document

Team Members:

Jacob Schmitt, Timothy Mai, Austyn Mancini, Kuan Liu, Nathaniel Hector, Sean Jerzewski

Sponsor:

Neil Toporski

Prepared by Kuan Liu on 03/07/2024

Table of Contents

1. Introduction

1.1. Purpose

This document will go over in detail about the design and architecture that is currently present in the case base simulation application.

1.2. Application Summary

The Case-Based Simulation Builder is a web-based application designed to allow developers to create, manage, and deploy interactive, case-based simulations. This tool is primarily aimed at enhancing training and educational processes across various industries such as healthcare, finance, emergency services, and engineering, by facilitating immersive and interactive learning experiences.
The application is structured around a decision tree that includes four main types of nodes: Scenario, Information-Gathering, Decision-Making, and End nodes. Users begin their journey in the simulation with a scenario node where a problem set is presented. They then navigate through the simulation, making choices and gathering information which are represented by different interactive nodes within a visually organized tree structure.

A key feature of the application is its dynamic tree display, built using Flutter, which allows for up to 10 levels of depth and multiple branching paths. This display supports the addition and deletion of nodes, ensuring flexibility and adaptability in the simulation design. Nodes within the tree are visually distinctive and clickable, each represented by specific shapes and colors. Clicking on these nodes brings up modal pop-up windows that show relevant data or decisions points, with fields that auto-populate with saved data or remain empty if not previously filled.

The Case-Based Simulation Builder is a robust platform that not only supports the development of critical thinking and decision-making skills in a risk-free environment but also provides tools for consistent training and assessment, adapting to the specific needs of various user groups and learning scenarios.

1.3. Application Environment

To accommodate the diverse range of individuals utilizing this system, it is imperative for the system to exhibit a high degree of adaptability across various platforms. The web application should be universally accessible through any web browser, ensuring no user is restricted by their choice of technology. Furthermore, the application's design must be responsive, with interfaces that adjust or provide scrolling to ensure content is fully viewable on screens of all sizes and resolutions.

2. System Architecture

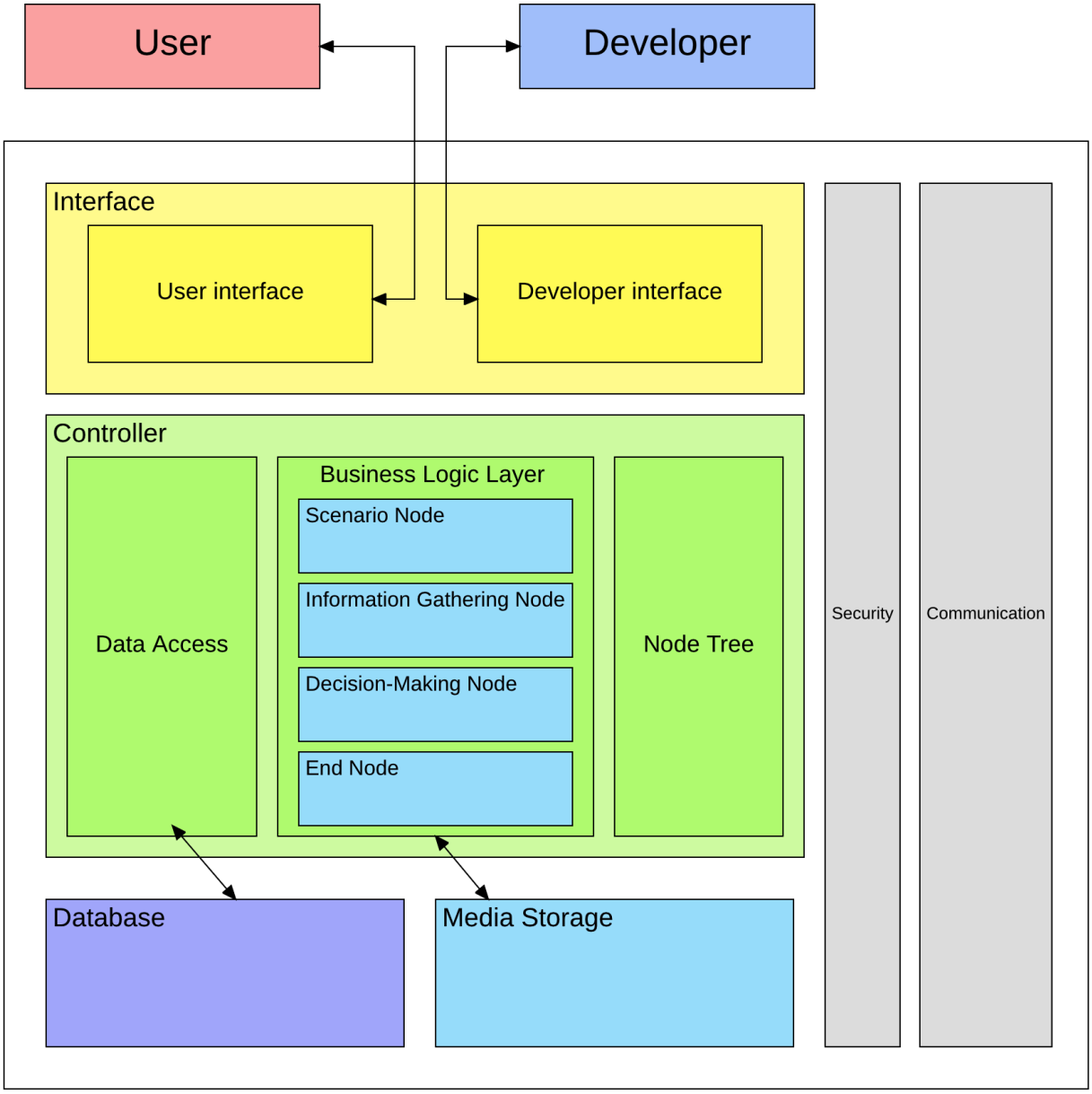
2.1. Architectural Strategies

The Case-Based Simulation is a web application developed using Flutter Web, which leverages the Flutter SDK to build web applications and employs Dart as its programming language. This application is integrated with Firebase, a real-time database that facilitates the storage and synchronization of data across user interfaces and the application itself. It also utilizes Firebase Storage to manage media files like images, videos, and audio, which are essential for enhancing the interactive elements of the simulations.

Additionally, the application incorporates Flutter packages such as the YouTube embedded player and audio player to enrich user engagement. Flutter Web provides comprehensive solutions for both the frontend and backend functionalities of the Case-Based Simulation. For testing purposes, the application is hosted on Firebase Hosting to ensure internal quality assurance before launch, while the production version will be deployed on an AWS server to cater to end-users.

Looking ahead, the flexibility of the Flutter SDK allows developers to extend this same codebase to support not only web applications but also mobile platforms like iOS and Android, significantly broadening the application's reach and usability.

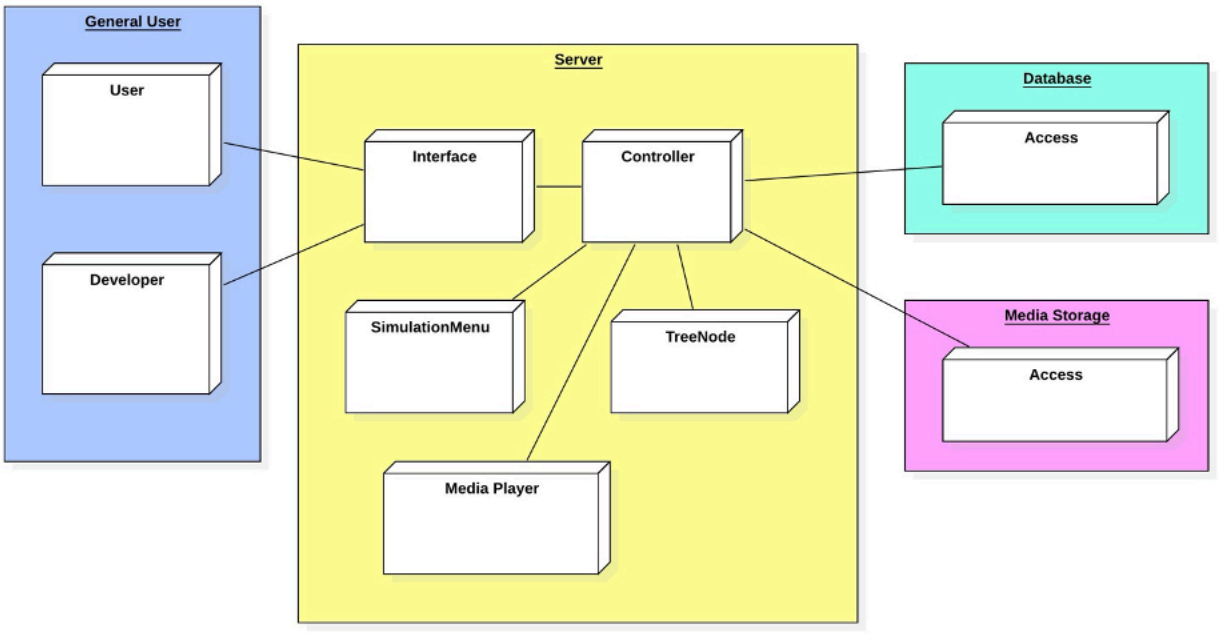
2.2. Logical View



The diagram presents a logical view of the Case-Based Simulation web application's architecture, highlighting the interaction between users and developers with the system. Users engage with simulations via a dedicated interface, while developers utilize a separate interface to create and manage these simulations. The application's core is a controller layer that houses the business logic, orchestrating the flow through various

nodes—Scenario, Information Gathering, Decision-Making, and End—which represent the steps of the simulation. These nodes are organized within a dynamic Node Tree, central to the simulation's interactive experience. Below this, the Data Access component mediates between the business logic and the Database, where simulation data is stored, and the Media Storage, where multimedia content is kept. Security and Communication are overarching layers that ensure data integrity and efficient communication throughout the application. This architecture supports a robust, interactive platform that separates concerns for an efficient and secure user experience.

2.3. Physical View

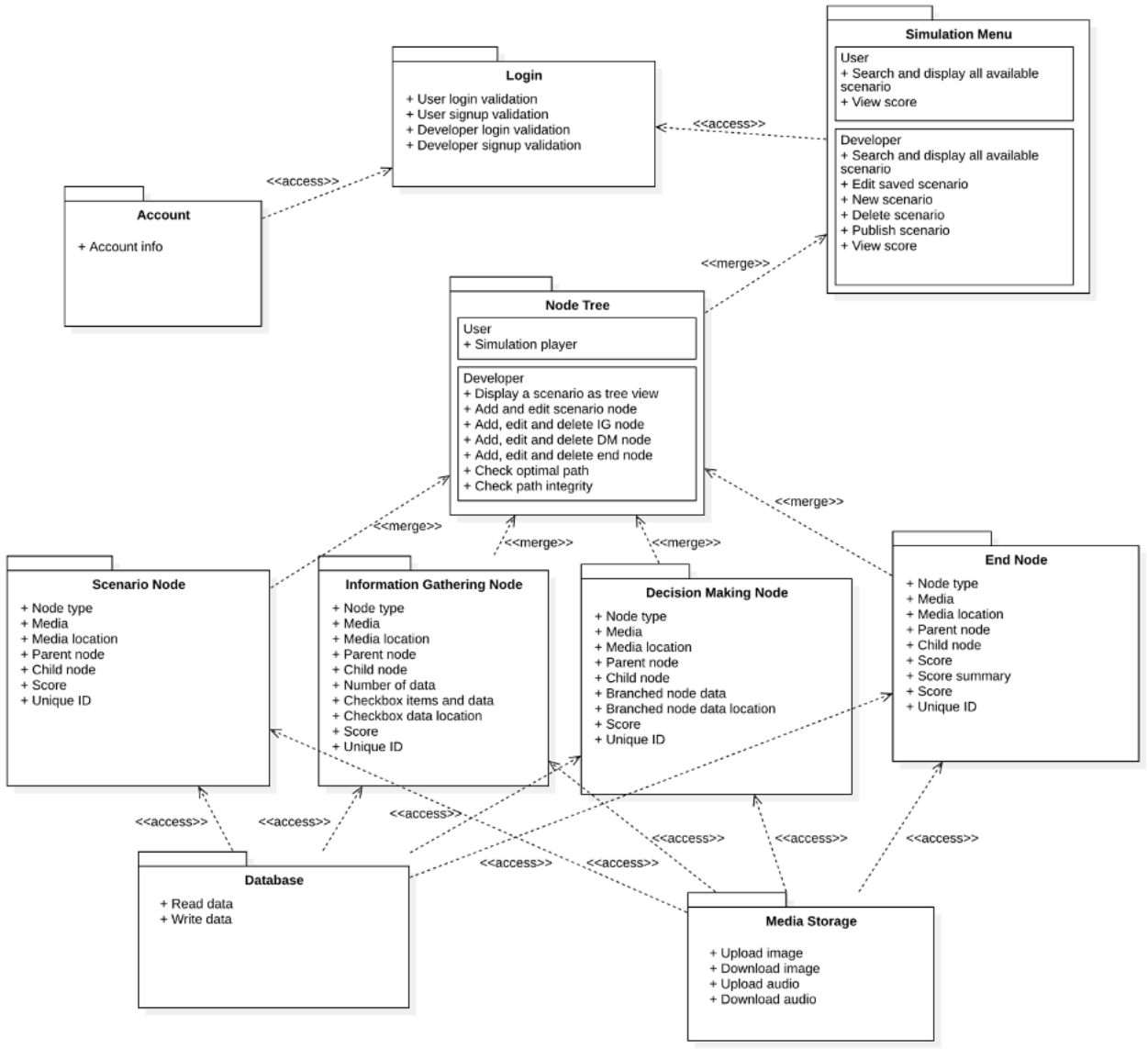


This is a physical view diagram of a software system, detailing the architecture of the Case-Based Simulation application from a deployment perspective. Here's a concise description of its components:

Two primary actors, the General User and Developer, interact with the system's Server, which is the heart of the application. The Server encompasses an Interface for user interactions, a Controller for managing the application logic, a SimulationMenu for navigation within the simulations, and TreeNode components that represent the structure of the simulation content. The Media Player within the server suggests that

multimedia content is a core part of the user experience. External to the Server but connected to it are the Database and Media Storage components, each with an Access module indicating how the system interacts with persistent storage. The Database would handle data operations such as storing user progress, while Media Storage would contain files such as images, videos, and audio clips used in the simulations. This diagram illustrates a system designed for scalability and efficient management of interactive simulation content, with a clear delineation between the different layers of interaction, data handling, and media management.

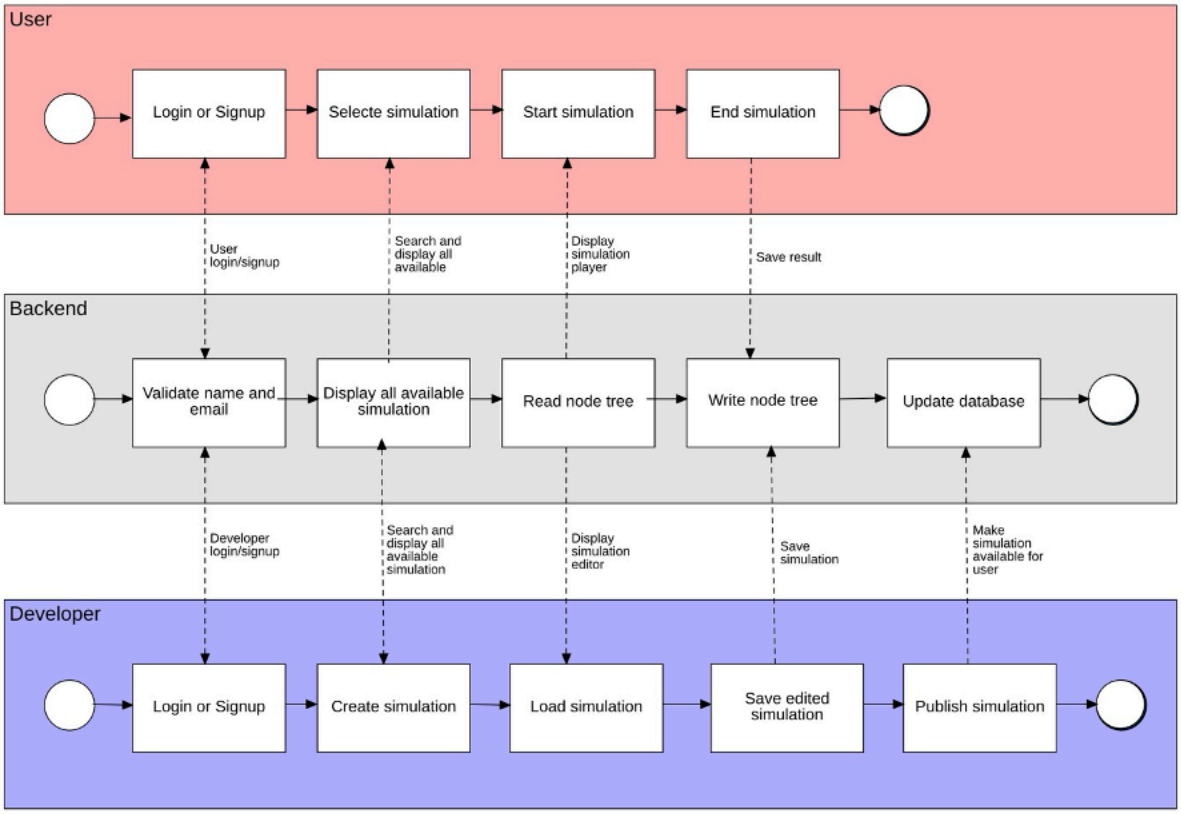
2.4. Development View



The development view of the Case-Based Simulation application presents a structured architecture tailored for developer interaction and user engagement. Central to the application is the Node Tree, a complex structure where developers can manipulate scenario, information gathering, decision making, and end nodes, each with unique identifiers and media associations. This tree is pivotal for constructing simulations, ensuring a logical flow and integrity of the decision-making paths. A dedicated Login module secures access, distinguishing between user and developer roles, while the Account component manages personal information. The Simulation Menu serves a dual

purpose, offering users the ability to interact with simulations and providing developers with tools to create, edit, and monitor their simulations. Interaction with persistent storage is facilitated through a Database for data transactions and a Media Storage component for handling multimedia content, crucial for the application's immersive experiences. This architecture supports a seamless workflow, accommodating the dynamic needs of developing and maintaining complex, interactive simulations.

2.5. Process View



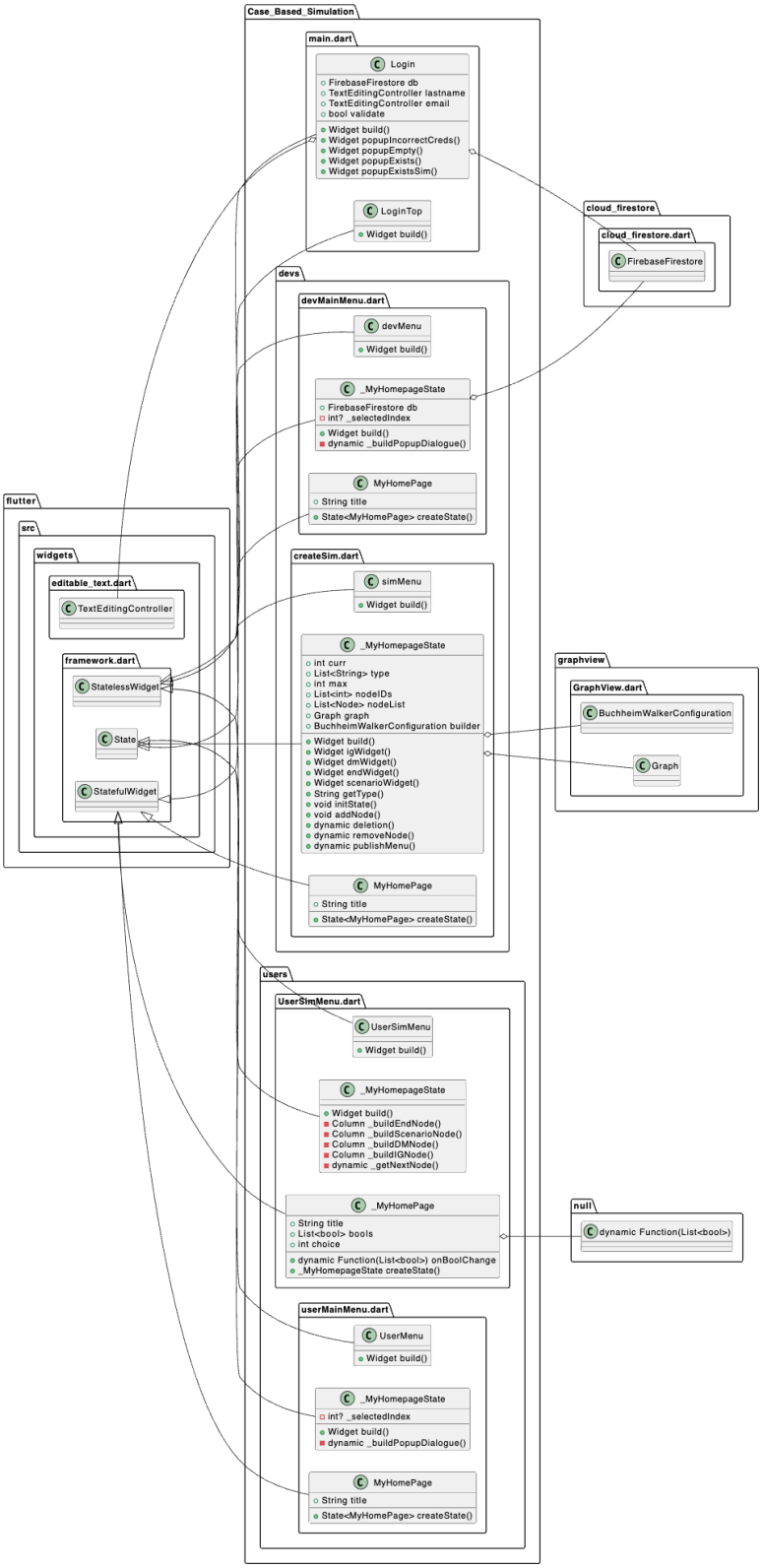
The diagram outlines the process view of the Case-Based Simulation application, detailing the user, developer, and backend interactions. Users start by logging in or signing up, which is validated by the backend. Once authenticated, they can select a simulation from a list provided by the backend, which retrieves available simulations. The simulation begins, is interactively managed by the user, and upon completion, the

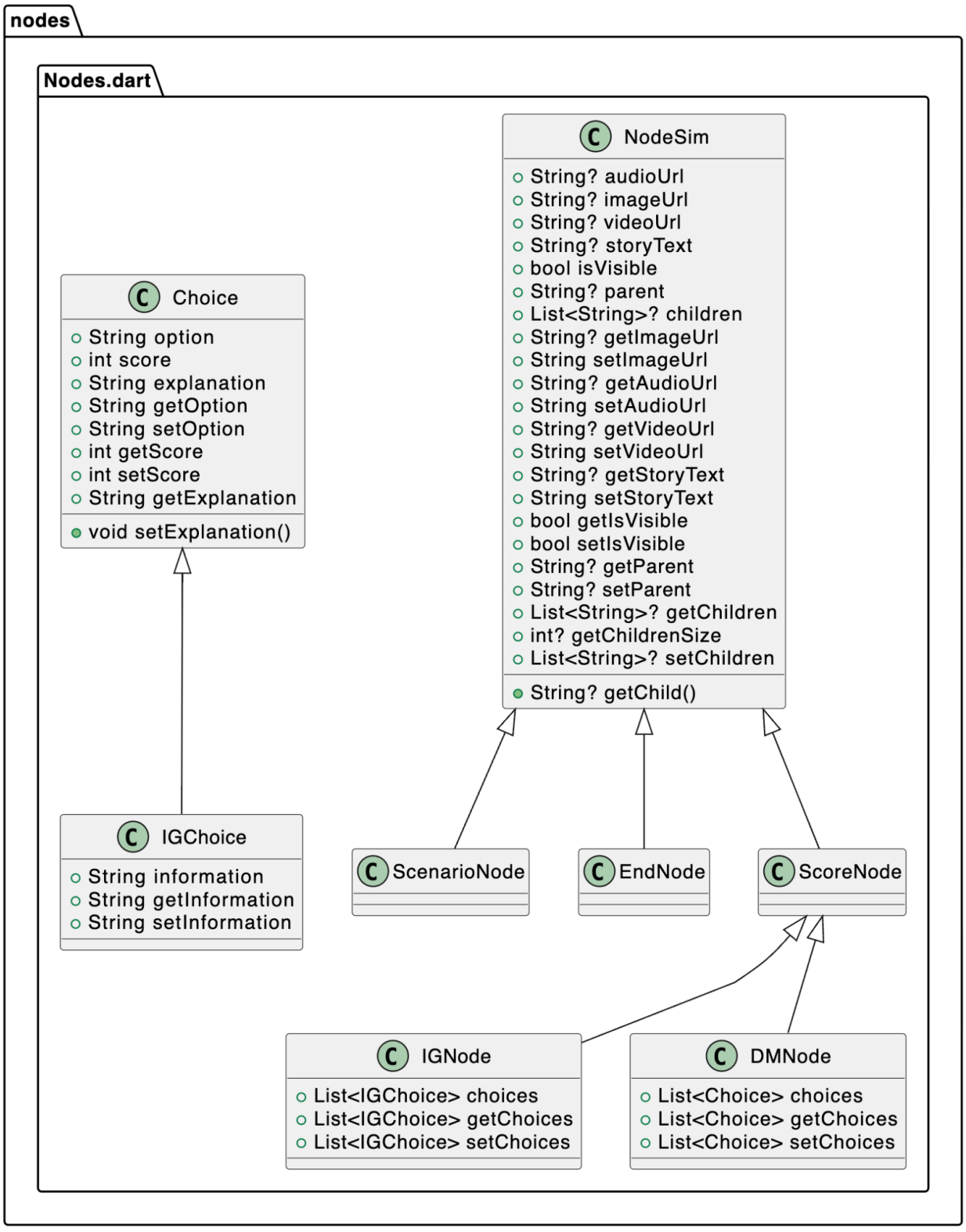
results are saved to the backend where the node tree is written and the database is updated.

In parallel, developers go through a similar login process. They have the ability to create new simulations, which involves loading and configuring the simulation content. After editing and finalizing the simulations, developers can save their work. The backend, again, plays a pivotal role in writing the node tree to reflect these changes and updating the database accordingly. Once finalized, developers can publish the simulations, making them available for user access.

This process view effectively captures the flow of activities from the perspectives of the user and developer, with the backend serving as the central hub that processes, stores, and retrieves data, ensuring a smooth operation of the Case-Based Simulation application.

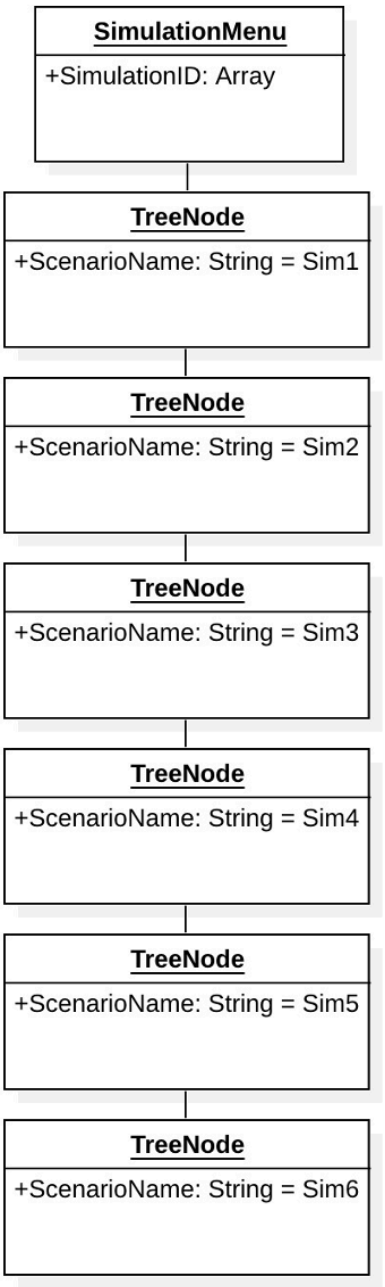
3. Class Diagram




nodes
Nodes.dart
NodeSim
String? audioUrl
String? imageUrl
String? videoUrl
String? storyText
bool isVisible
String? parent
List<String>? children
String? getImageUrl
String setImageUrl
String? getAudioUrl
String setAudioUrl
String? getVideoUrl
String setVideoUrl
String? getStoryText
String setStoryText
bool getIsVisible
bool setIsVisible
String? getParent
String? setParent
List<String>? getChildren
int? getChildrenSize
List<String>? setChildren
String? getChild()
Choice
String option
int score
String explanation
String getOption
String setOption
int getScore
int setScore
String getExplanation
void setExplanation()
IGChoice
String information
String getInformation
String setInformation
ScenarioNode
EndNode
ScoreNode
IGNode
List<IGChoice> choices
List<IGChoice> getChoices
List<IGChoice> setChoices
DMNode
List<Choice> choices
List<Choice> getChoices
List<Choice> setChoices

4. Object Diagram

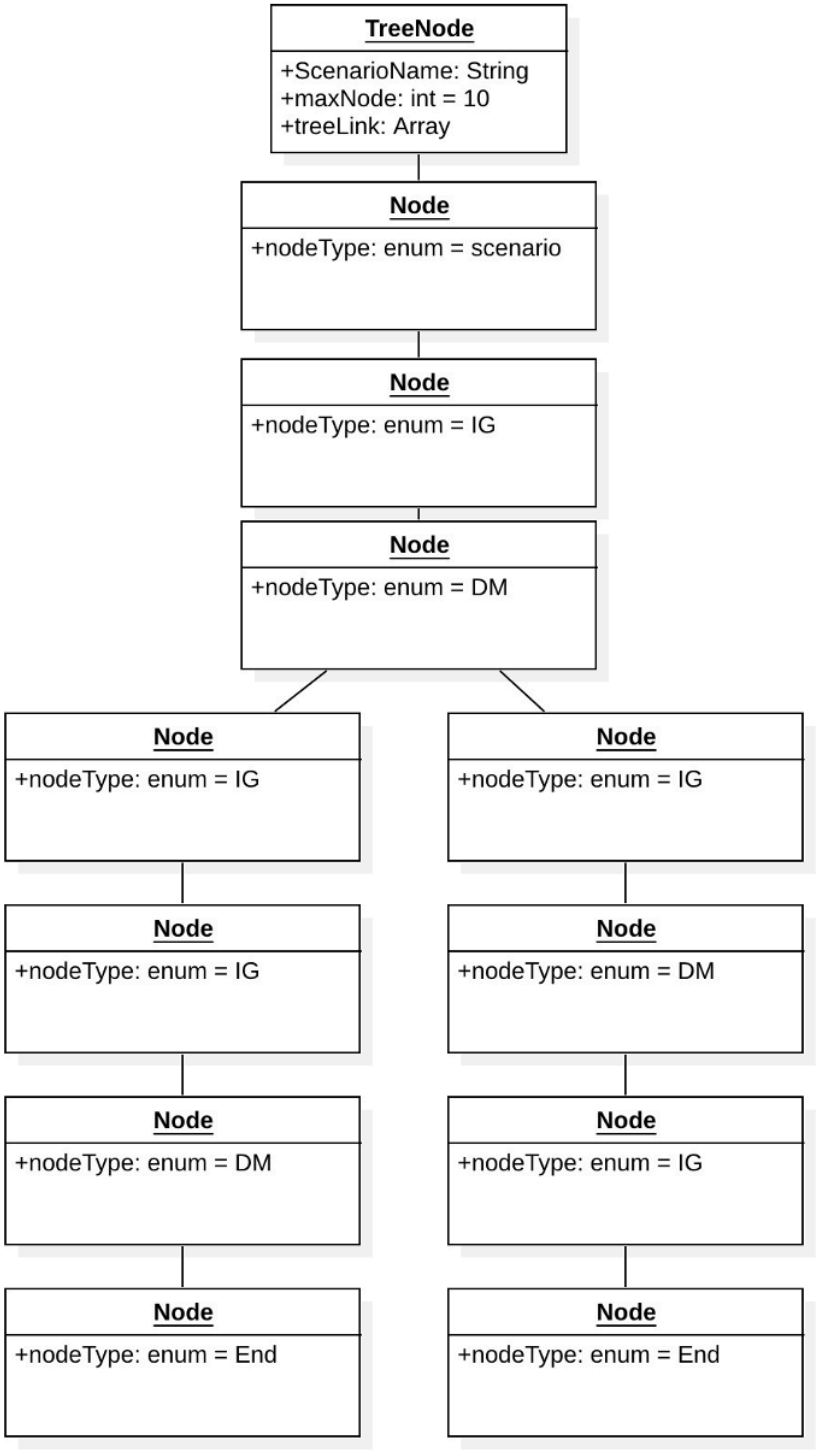
4.1. Simulation Menu Object View



The object diagram illustrates a menu system for a simulation application, composed of a SimulationMenu object that maintains an array of SimulationNames, serving as a directory for individual simulations. The menu connects to a series of TreeNode objects, each with a ScenarioName attribute. These TreeNodes represent different scenarios

within the simulation application, structured in a way that suggests a hierarchical or sequential access pattern. The SimulationMenu functions as the entry point for users or developers to select and interact with various simulation scenarios, providing an organized and accessible interface for navigating through the different simulations encapsulated by the TreeNode objects.

4.2. Tree Node Object View

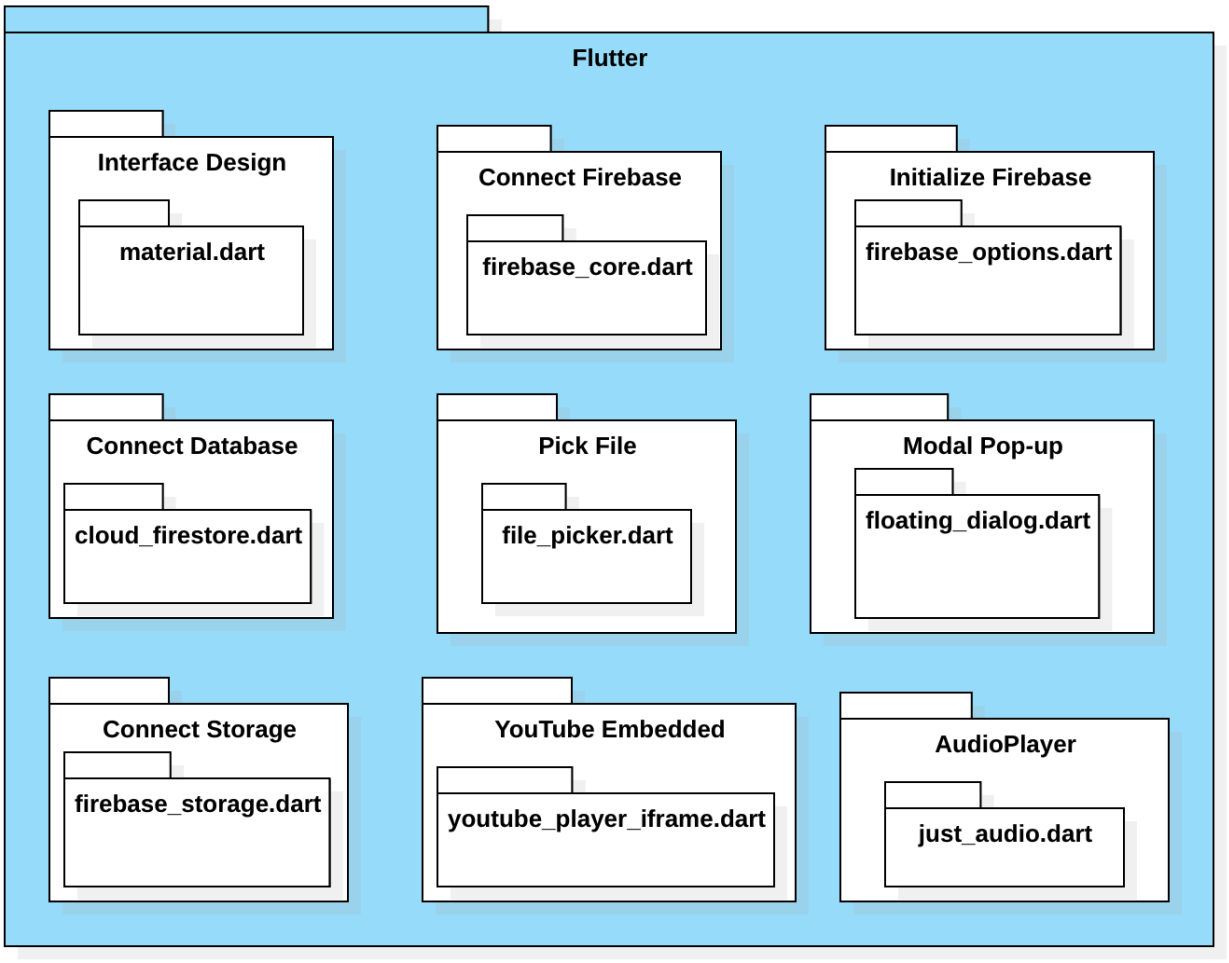


The object diagram provided showcases a TreeNode structure which serves as a framework for organizing node elements within a simulation. The top-level TreeNode is

defined by attributes including a ScenarioName and a treeLink, an array likely meant to hold connections to other nodes, with a maximum node link of 10. Below this, a hierarchy of Nodes is established, each differentiated by a nodeType enumerated value. These Node objects represent different stages or types of interactions within the simulation, indicated by their enumerated types: 'scenario', 'IG' (Information Gathering), 'DM' (Decision Making), and 'End'. The tree structure depicted by the diagram illustrates a branching path, where a scenario node leads to various IG and DM nodes, further expanding into additional IG nodes and eventually converging at End nodes. This arrangement suggests multiple possible pathways through the simulation, culminating in different endpoints depending on the decisions made during the simulation. The design encapsulated in the diagram highlights the complexity and interactivity of the simulation, which is structured to dynamically respond to user choices, creating a versatile tool for scenario-based learning or decision-making processes.

5. Package Diagram

5.1. General View



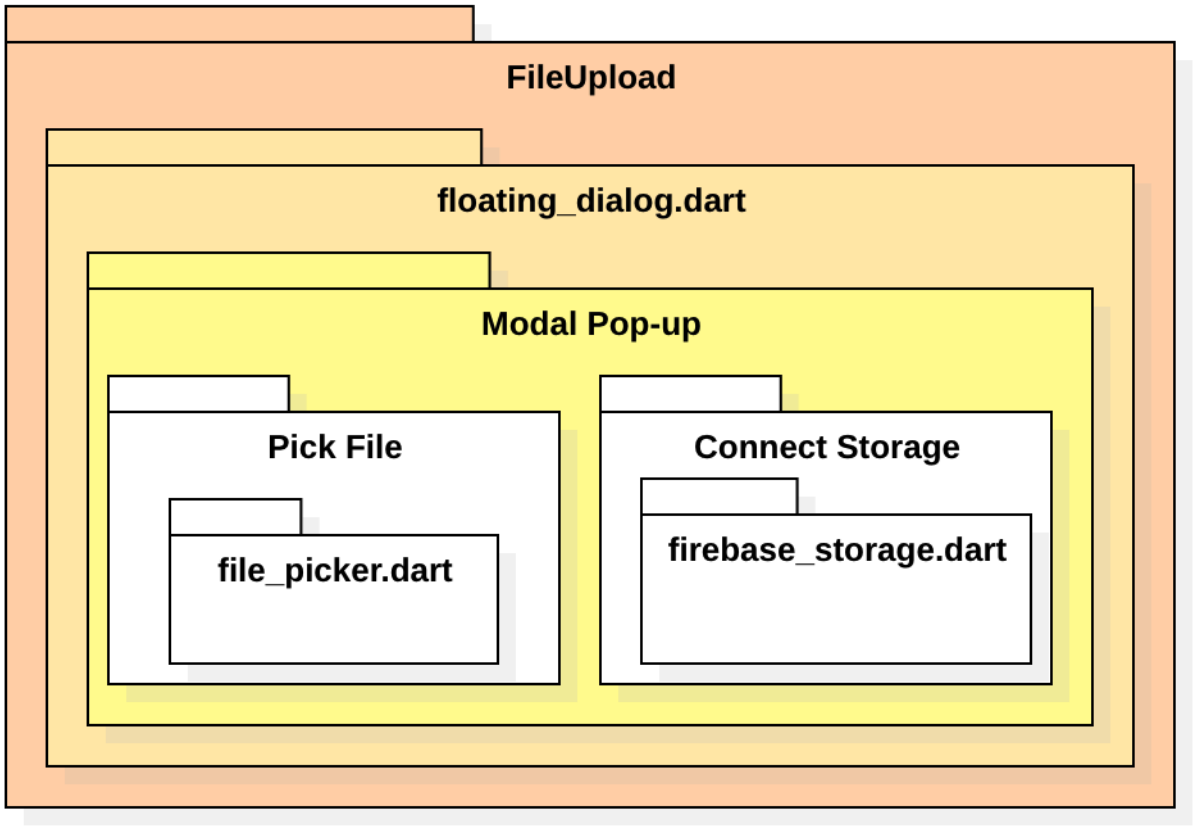
The diagram represents a package diagram for the Flutter-based Case Based Simulation project, detailing the modular organization and the dependencies between various software components. The Interface Design is handled by the material.dart package, ensuring that the application adheres to Material Design guidelines. Connection to Firebase is managed through firebase_core.dart, which is essential for initializing and configuring Firebase with firebase_options.dart. The database interactions are facilitated by cloud_firestore.dart, allowing for real-time data exchange and storage with Firebase's Cloud Firestore service.

For file management, file_picker.dart is used, enabling the application to access and upload files from the device. The user interface for interactions such as file selection or displaying additional information is managed by floating_dialog.dart, which likely

provides modal pop-up dialog functionality. Multimedia content is handled through separate packages: youtube_player_iframe.dart for embedding YouTube videos and just_audio.dart for audio playback capabilities.

This package diagram effectively outlines the structure and interconnectivity of the application's components, emphasizing a clear separation of concerns where each package serves a distinct and essential purpose in the overall functionality of the application, from user interface design to database interaction and multimedia handling.

5.2. File Upload View

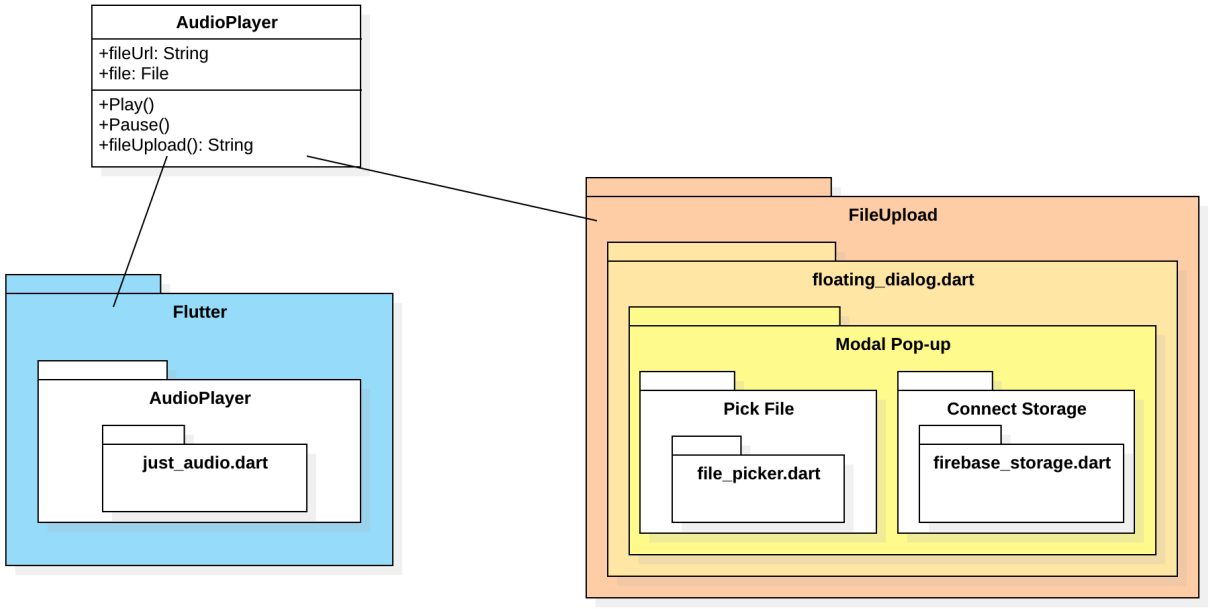


The package diagram details the structure of the file upload functionality within a Flutter application. The overarching module, FileUpload, contains the floating_dialog.dart file, which likely manages the user interface for file uploading interactions. Within this FileUpload package, there's a Modal Pop-up section, emphasizing the user experience and interface during the file selection process.

This Modal Pop-up segment includes two core components: file_picker.dart for the Pick File feature, allowing users to navigate their device's file system to select a file, and firebase_storage.dart for the Connect Storage feature, which establishes a connection to Firebase Storage for the actual file upload.

Together, these packages form a cohesive file upload system, presenting the user with a modal dialog to select files and then handling the upload process to store the files in Firebase Storage, ensuring a seamless integration of front-end file selection with back-end storage services.

5.3. Audio Player View



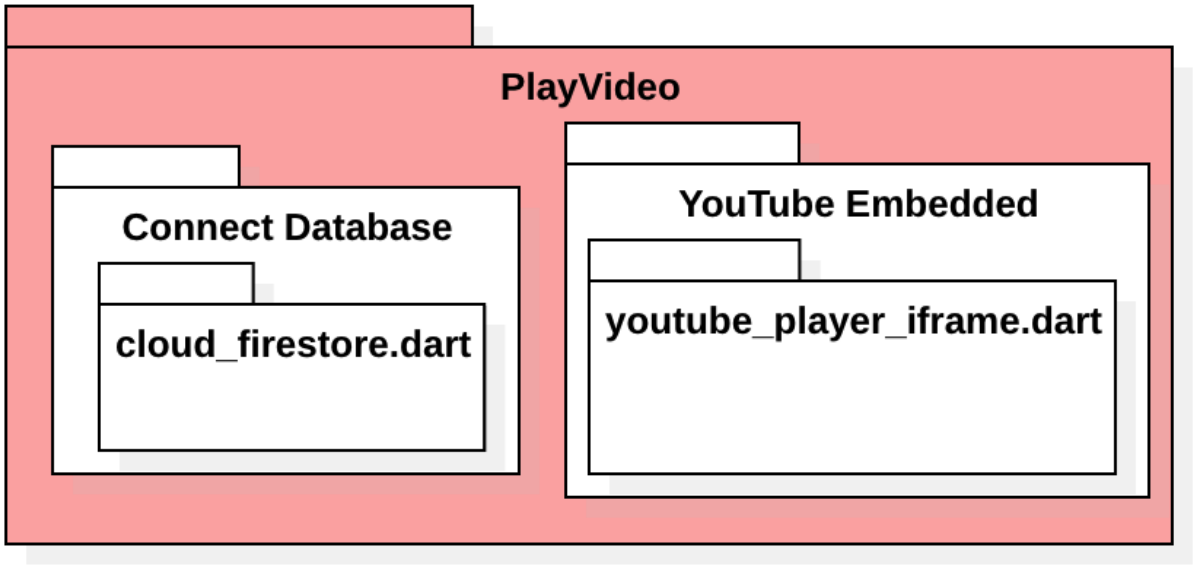
The package diagram portrays the components involved in the audio upload functionality of a Flutter application. The Flutter package contains an AudioPlayer module, managed by the just_audio.dart library, which facilitates audio playback with functions to play and pause. The AudioPlayer class also has a method for file upload, which likely returns a String path or URL of the uploaded file.

In addition, there’s a FileUpload package that works in conjunction with the AudioPlayer. It includes a floating_dialog.dart component that might handle the user interface for

audio file selection and uploading. Within the FileUpload package, there's a Modal Pop-up section comprising two sub-components: file_picker.dart for selecting audio files and firebase_storage.dart for connecting to and storing files in Firebase Storage.

This setup indicates a streamlined process where users can choose audio files to upload and play them within the application, with backend support for file storage through Firebase. The diagram suggests a cohesive flow from file selection to playback and storage, encapsulating the user interaction and backend processes within a modular framework.

5.4. YouTube Player View



The package diagram for the YouTube player function outlines the PlayVideo module which is split into two primary components for a Flutter application. The 'Connect Database' component utilizes the cloud_firestore.dart package, indicating the app's capability to interface with Cloud Firestore for database operations, such as retrieving video URL or storing user preferences. The 'YouTube Embedded' component uses youtube_player_iframe.dart, which suggests the application can embed YouTube videos directly into its interface, likely providing users with seamless video streaming

capabilities. Together, these components allow the PlayVideo module to support database-connected video content management and display within the application.

6. Robustness Diagram

The robustness diagram represents the interaction flow between users, developers, and a web application, outlining the system's response to different roles during the login and account validation process. Users and developers begin by logging in or signing up, and their requests are directed to the web application. Once a login or signup request is made, the application distinguishes between users and developers, leading to different validation processes and menu retrievals.

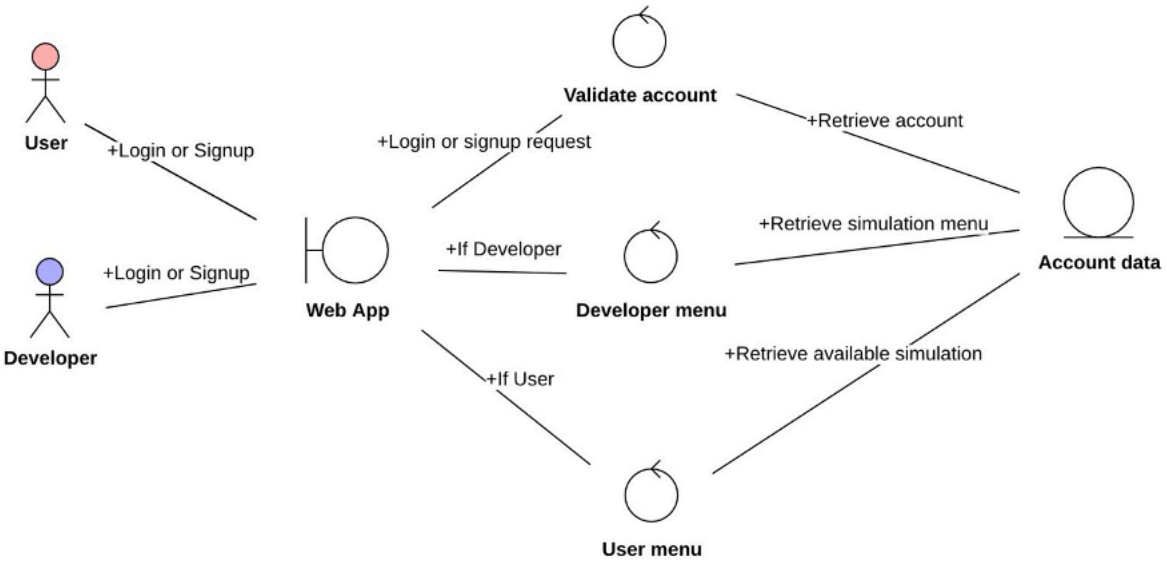
For developers, the application processes the login request, validates the account, and then grants access to the developer menu. This menu allows them to retrieve simulation menus and interact with account data, possibly for the purpose of creating or managing simulations.

For general users, the application validates the account similarly and directs them to a user menu. From this menu, users can retrieve available simulations to participate in or run simulations for which they have access.

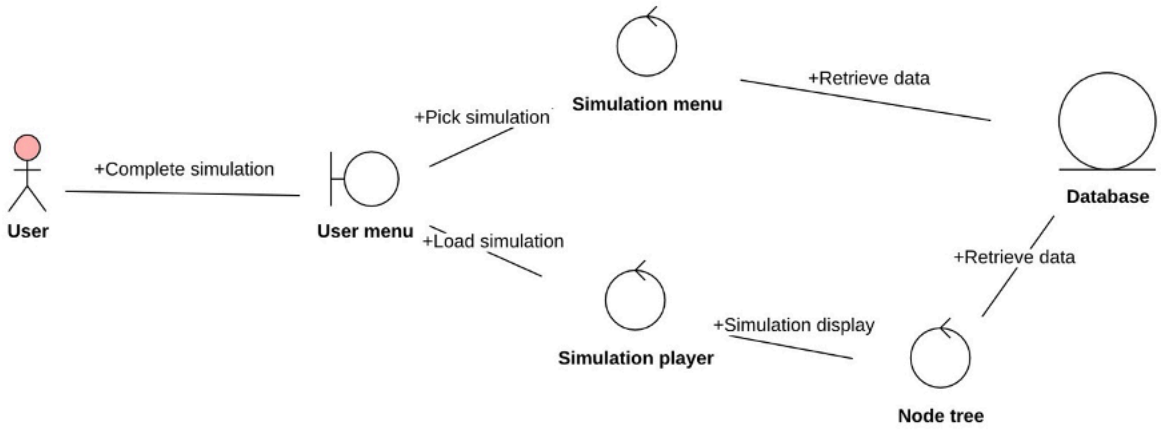
This diagram reflects a system designed with clear role-based access, ensuring appropriate functionality and data is provided depending on whether the individual is a developer or a user, with robust account validation and data retrieval processes integral to the user experience.

Actors: User, Developer
The following robustness diagram shows all the actions that the Actors can perform while using the Case-Base Simulation.

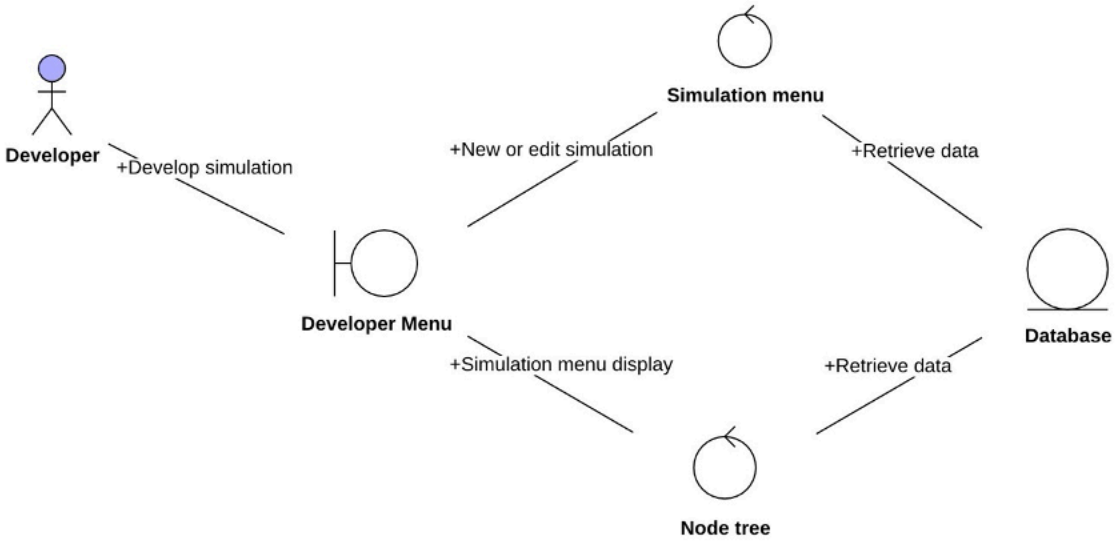
6.1. General View



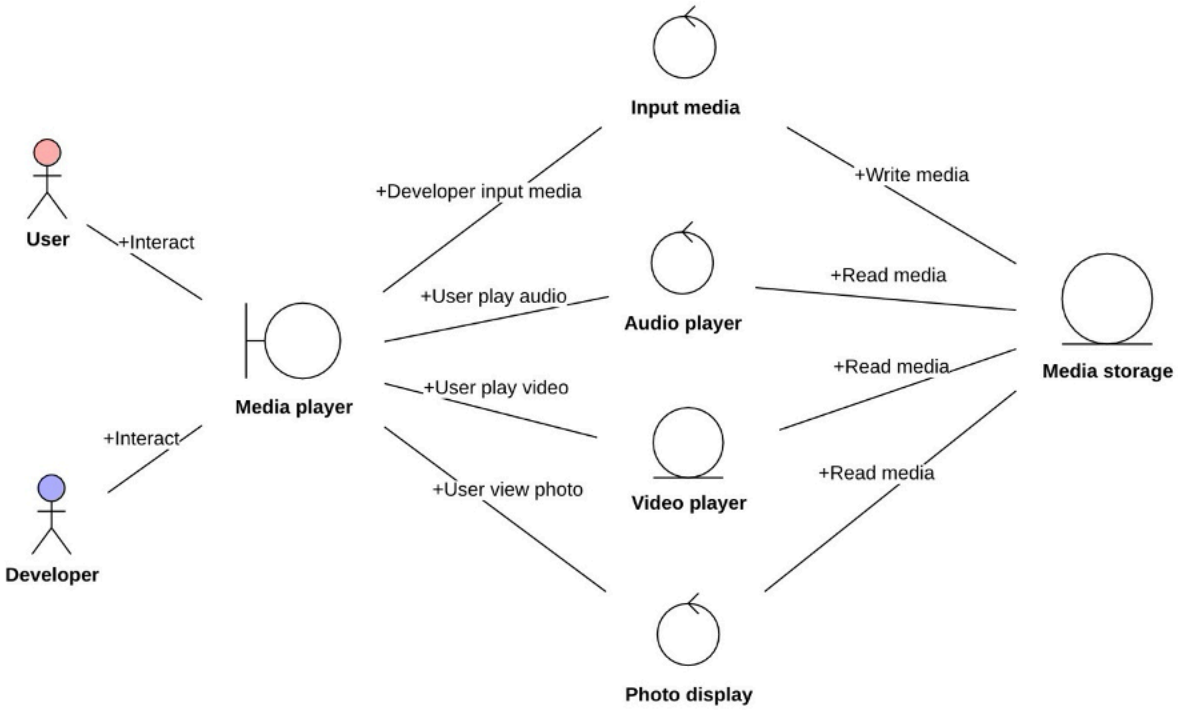
6.2. User View



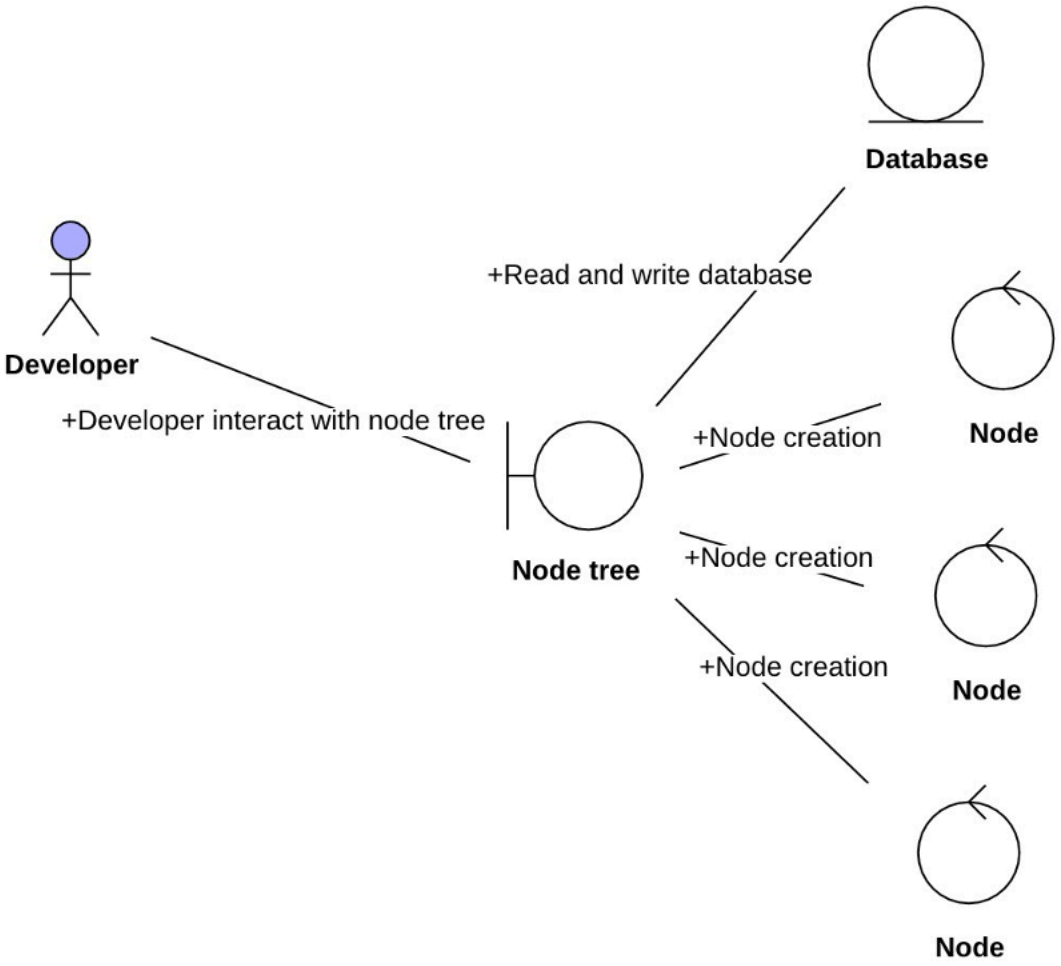
6.3. Developer View



6.4. Media PLayer View



6.5. Node TreeView

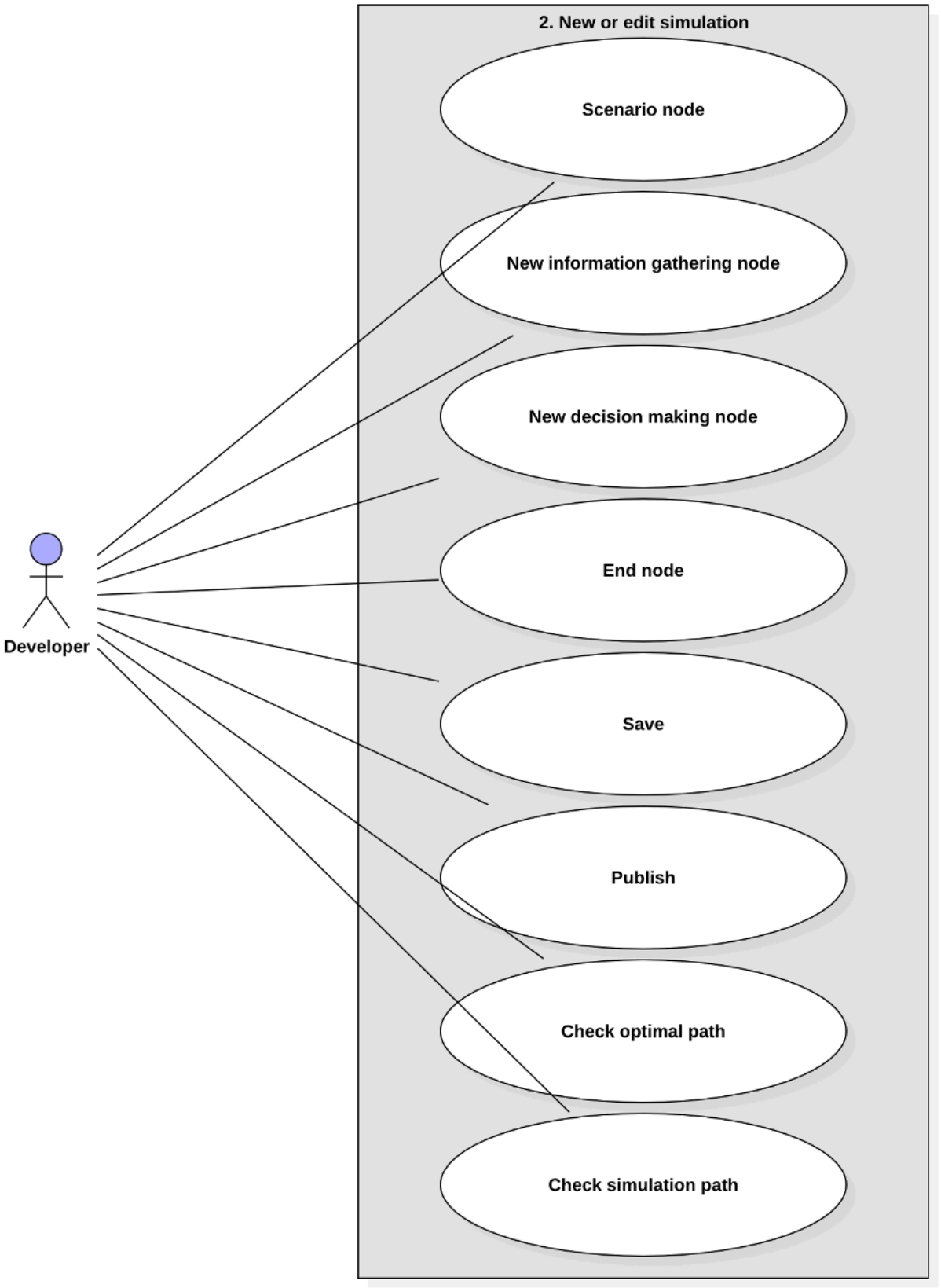


7. Use Case Diagram

Actors: User, Developer
The following use case diagram shows all the actions that the Actors can perform while using the Case-Base Simulation.

The use case diagram shows various interactions between users and developers within the Case-Based Simulation system. Both users and developers can log in or sign up. Once authenticated, a user can view and complete simulations, and then view their scores. Meanwhile, developers have more extensive capabilities: they can create or edit

simulations, delete simulations they have created, publish simulations to make them available to users, and view all users' scores, presumably to analyze the effectiveness or popularity of different simulations. This diagram outlines a clear separation of roles, with developers having the ability to manage the lifecycle of simulations, from creation to publishing and subsequent performance monitoring.

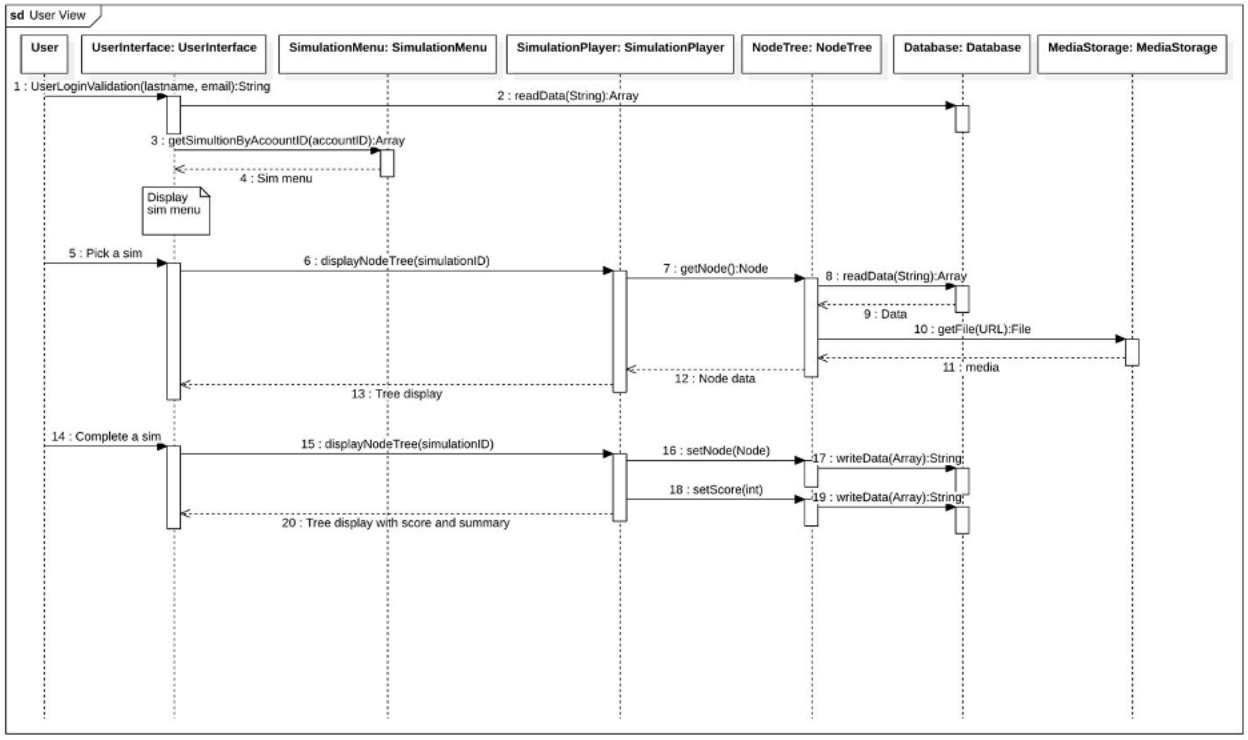

2. New or edit simulation
Scenario node
New information gathering node
New decision making node
End node
Save
Publish
Check optimal path
Check simulation path
Developer

The use case diagram illustrates the various actions available to a developer within a simulation creation system. The developer can start by creating or editing a simulation. During this process, they have the ability to add or modify different types of nodes within the simulation, including scenario nodes, information gathering nodes, and decision-making nodes, as well as bringing the simulation to a conclusion with an end node.

Once the simulation nodes are in place, the developer can save their progress. They also have the option to publish the simulation, making it accessible to users. Furthermore, the developer can check the optimal path of the simulation to ensure logical progression and desired outcomes. Additionally, they can verify the simulation path for consistency and correctness. This use case diagram captures the comprehensive set of tools and functionalities a developer has to create, test, and deploy interactive simulations within the system.

8. Sequence Diagram

8.1. User Action



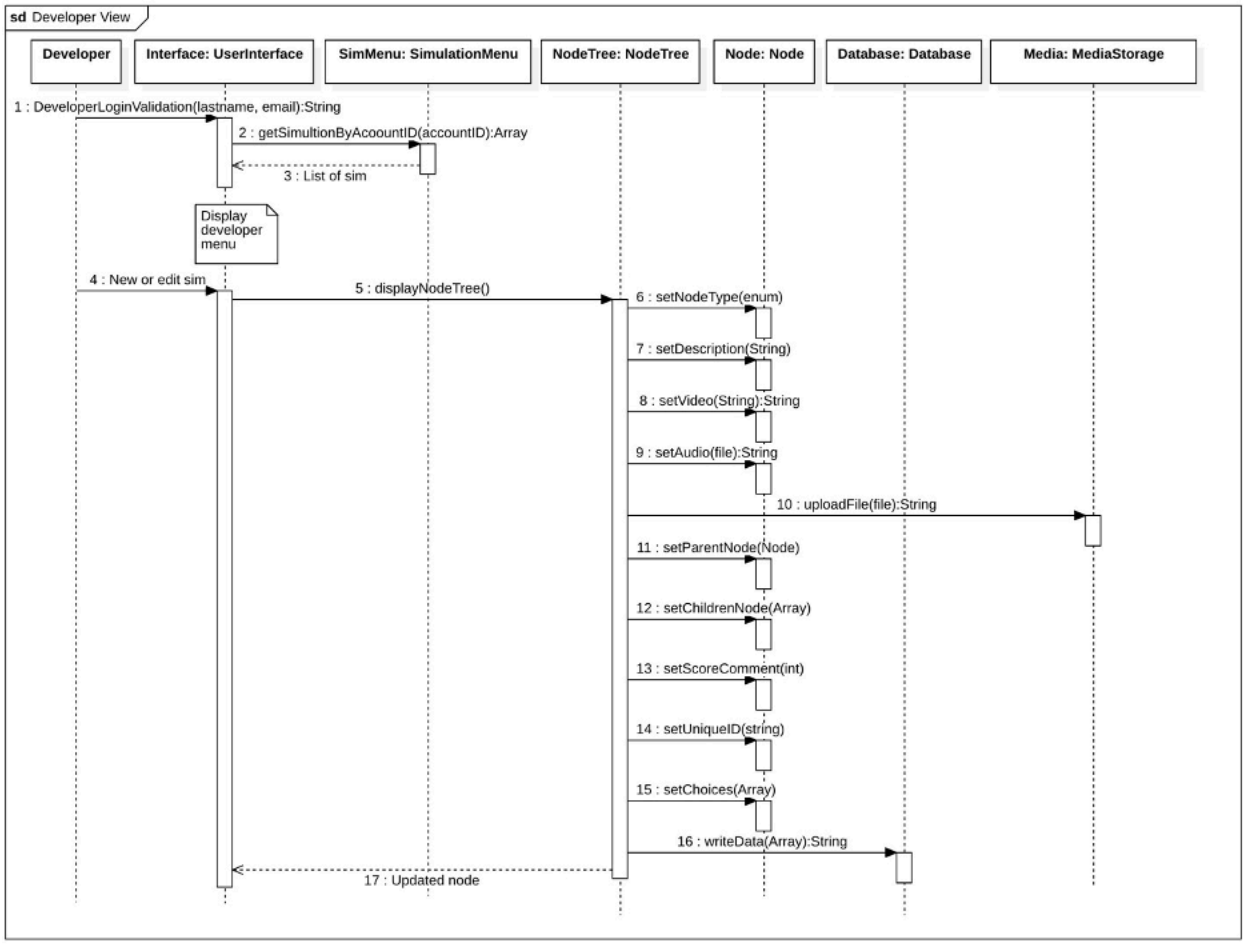
The sequence diagram depicts the flow of interactions for a user within a simulation system. The process begins with the user performing a login validation, which, upon success, triggers the retrieval of simulations associated with their account. Once the simulation menu is displayed, the user selects a simulation, prompting the system to display the corresponding node tree of the chosen simulation.

As the user interacts with the simulation player, node data is read and possibly manipulated within the node tree. Simultaneously, the system interacts with the database to retrieve or update data and with media storage to fetch any associated media files, such as images or videos, by their URL.

Upon completing the simulation, the node tree is displayed again, this time with the user's score and a summary of the simulation's outcome. The score is then set for the user, and the resulting data is written back to the database.

Throughout this sequence, there are several back-and-forth interactions between the components of the system, indicating a robust and responsive design that can handle dynamic content and user inputs effectively.

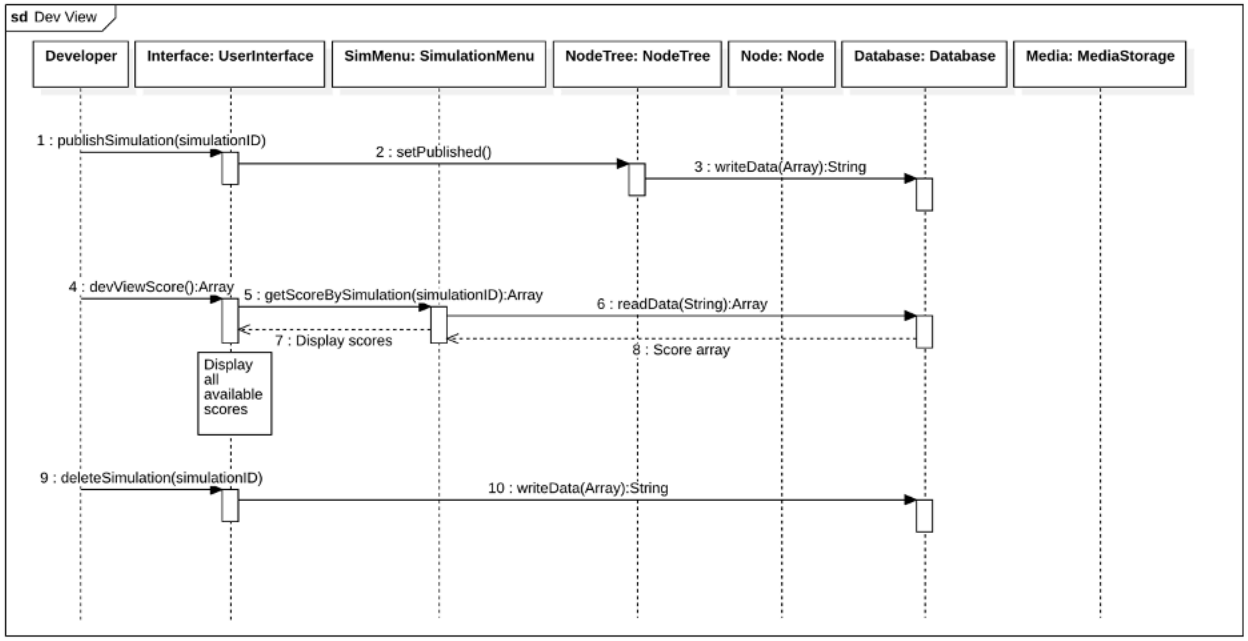
8.2. Developer Action



In the sequence diagram, the interaction between a developer and the system begins with the developer going through a login validation process. Once authenticated, the developer retrieves a list of simulations associated with their account and proceeds to either create a new simulation or edit an existing one.

The simulation editing process involves several steps: the developer displays the node tree, sets node types and descriptions, and assigns video and audio content to nodes using their respective URLs or files. The media is uploaded to the system, after the developer sets parent-child relationships within the node tree, adds comments on scores, and sets various choices and information available to the user within the simulation.

Once the nodes are fully configured, the developer finalizes the setup by saving the updated node data to the database. This sequence outlines a comprehensive, multi-step process that allows the developer to shape the simulation's narrative and decision paths, ensuring an interactive and engaging user experience.



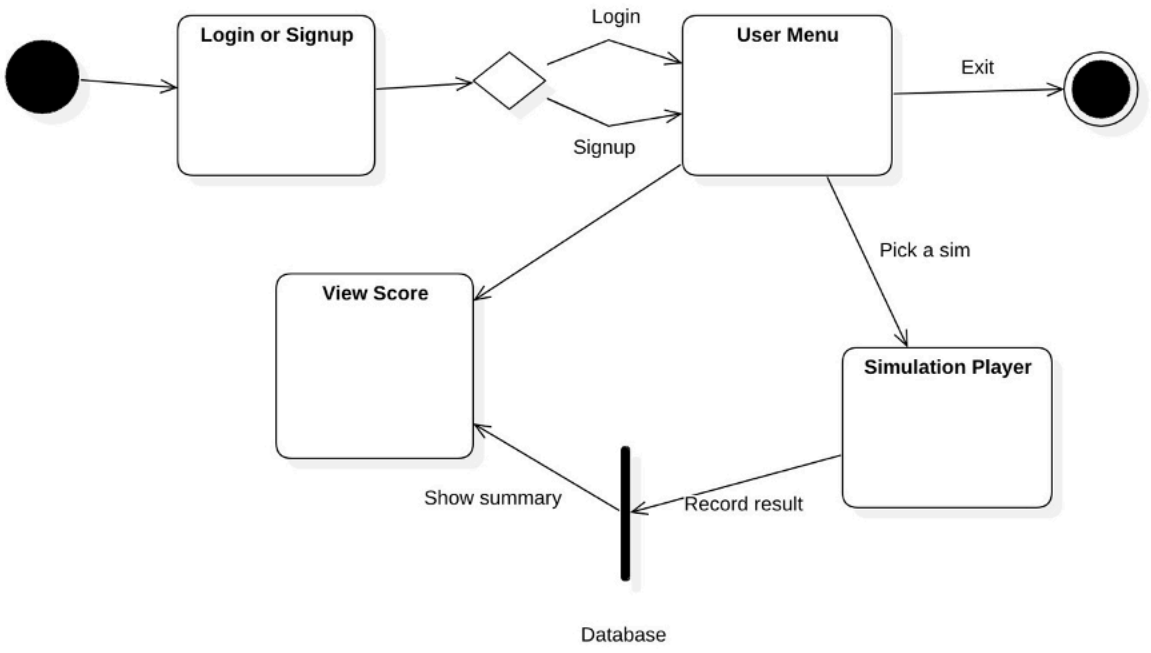
In the provided sequence diagram for the developer view, the interaction begins with the developer initiating the publication of a simulation by sending the simulation name to the system. The system then sets the simulation to a published state and updates the database accordingly.

Next, the developer requests to view scores related to a specific simulation. The system retrieves these scores from the database, which are then displayed to the developer, providing insight into user performance or simulation effectiveness.

Lastly, if needed, the developer has the ability to delete a simulation. This action is also facilitated by the system through an interaction with the database, where the simulation data is removed based on the provided simulation name, effectively completing the lifecycle of a simulation within the system. This sequence diagram outlines the critical functions related to the maintenance and evaluation of simulations that a developer can perform.

9. Activity Diagram

9.1. User Action

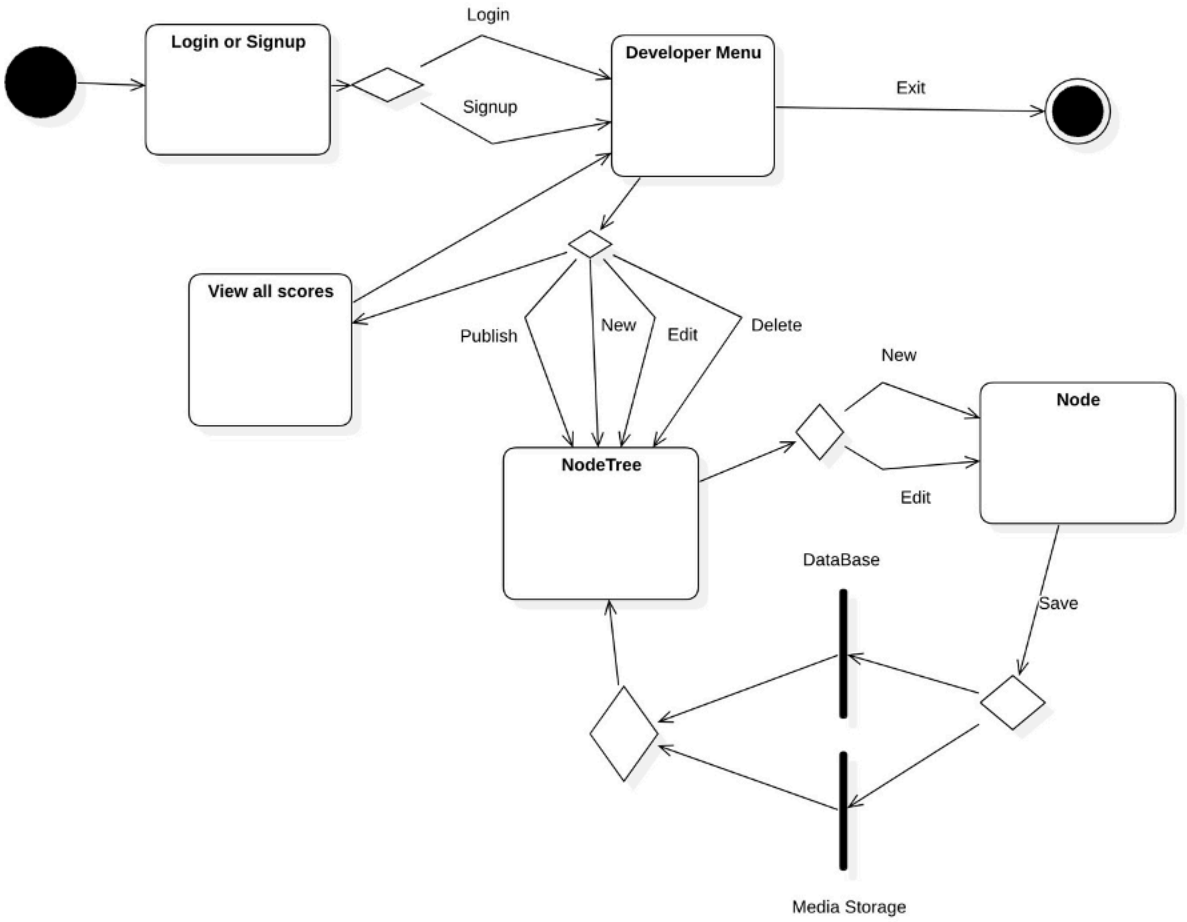


The activity diagram provides a visual flow of user actions within a simulation system. The process begins with a user choosing to either log in or sign up. Based on the choice, the flow diverges: a successful login takes the user to the User Menu, or signup before reaching the User Menu.

Once at the User Menu, the user can choose to exit the system or proceed to 'Pick a sim' to engage with a simulation. Selecting a simulation takes the user to the Simulation Player, where they can interact with the simulation. Upon completion, the results are recorded in the database, and the user is then able to 'View Score'. Viewing the score presents a summary of the simulation results, completing the user's journey through the simulation process.

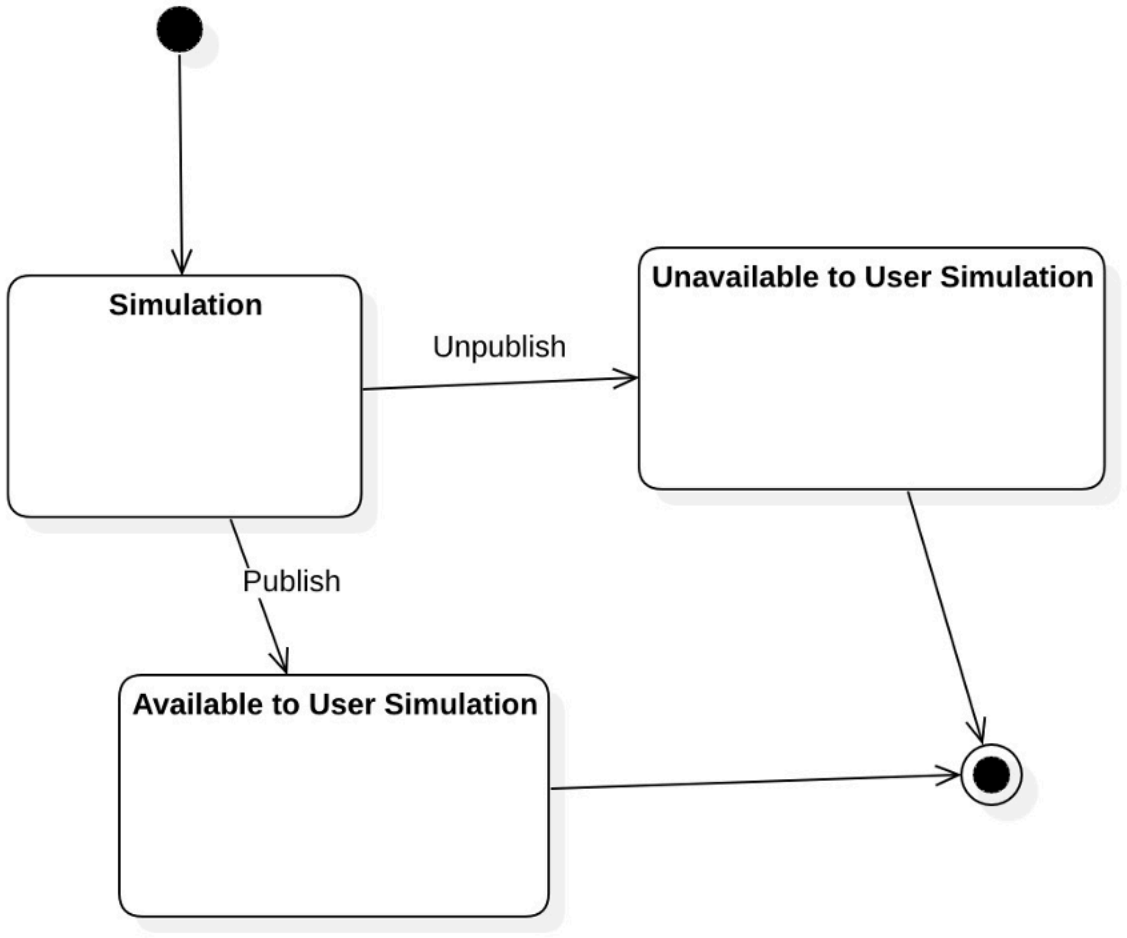
This activity diagram outlines the core functionalities available to the user, providing a clear pathway from authentication to simulation engagement and results review, with the database serving as the backend support throughout the process.

9.2. Developer Action



The activity diagram maps out the functionalities accessible to a developer within a simulation system. After logging in or signing up, the developer is directed to the Developer Menu. From this central hub, the developer has multiple options: view all user scores, create a new simulation, edit an existing simulation, or delete a simulation. When creating or editing a simulation, the developer works with a NodeTree, where they can add new nodes or edit existing ones. Each node's data can be saved to the database. Additionally, the developer has the option to publish a simulation, making it available for users. For new nodes, there is an interaction with Media Storage, implying that nodes can include media elements which need to be saved and managed. Upon completing their tasks, the developer can exit the system. This diagram effectively outlines the comprehensive set of actions that a developer can perform, from managing simulations to reviewing performance metrics.

9.3. Simulation State



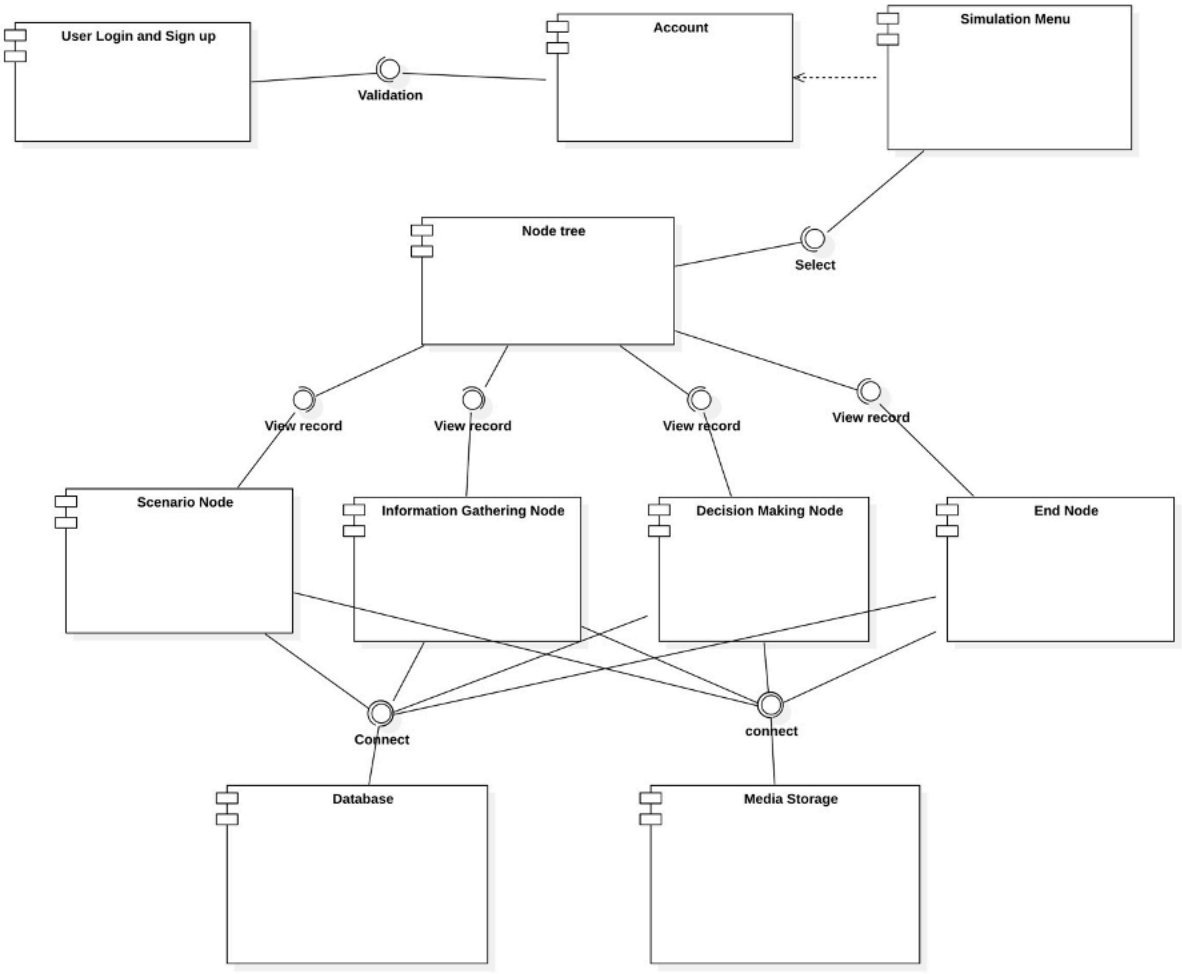
The activity diagram represents the publishing process of a simulation. It begins with a simulation that is in a neutral state. From this point, the developer has two options: to publish the simulation, making it available to users, or to leave it unpublished, keeping it unavailable to users. If the simulation is published, it transitions to a state where it is accessible for users to interact with. Conversely, the developer can also take an already available simulation and unpublish it, reverting it to a state where users cannot access it. The process concludes once the simulation is either made available to users or remains/made unavailable, demonstrating a simple, binary workflow that is central to content management within the system.

10. Component Diagram

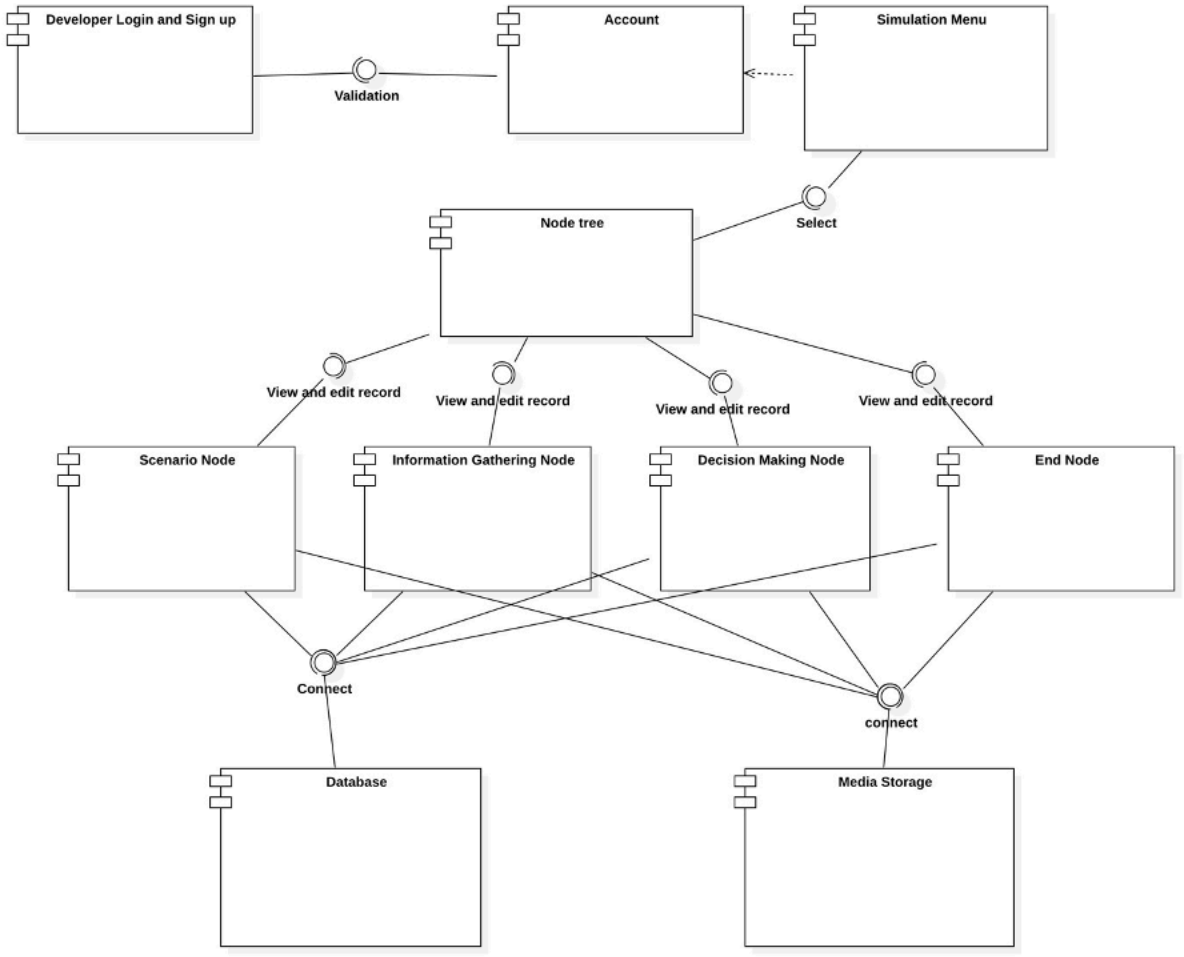
The component diagram outlines the structure of a system for creating and managing simulations. The process starts with "Login and Sign up," leading to "Validation" and then branching out to "Account" management and "Simulation Menu" selection. From the Simulation Menu, a "Node Tree" is accessible, where various types of nodes - "Scenario Node," "Information Gathering Node," "Decision Making Node," and "End Node" - can be selected and linked to form the framework of the simulation.

Each node type allows for viewing and editing records, with all interactions being managed and connected to a central "Database" that stores the simulation data. Additionally, "Media Storage" is another component linked to the nodes, indicating that multimedia elements are integrated into the simulations. The connections between the nodes and these two storage components suggest a dynamic system that supports complex data and media management, allowing for detailed customization and record-keeping of the simulations.

10.1. User Component Diagram



10.2. Developer Component View

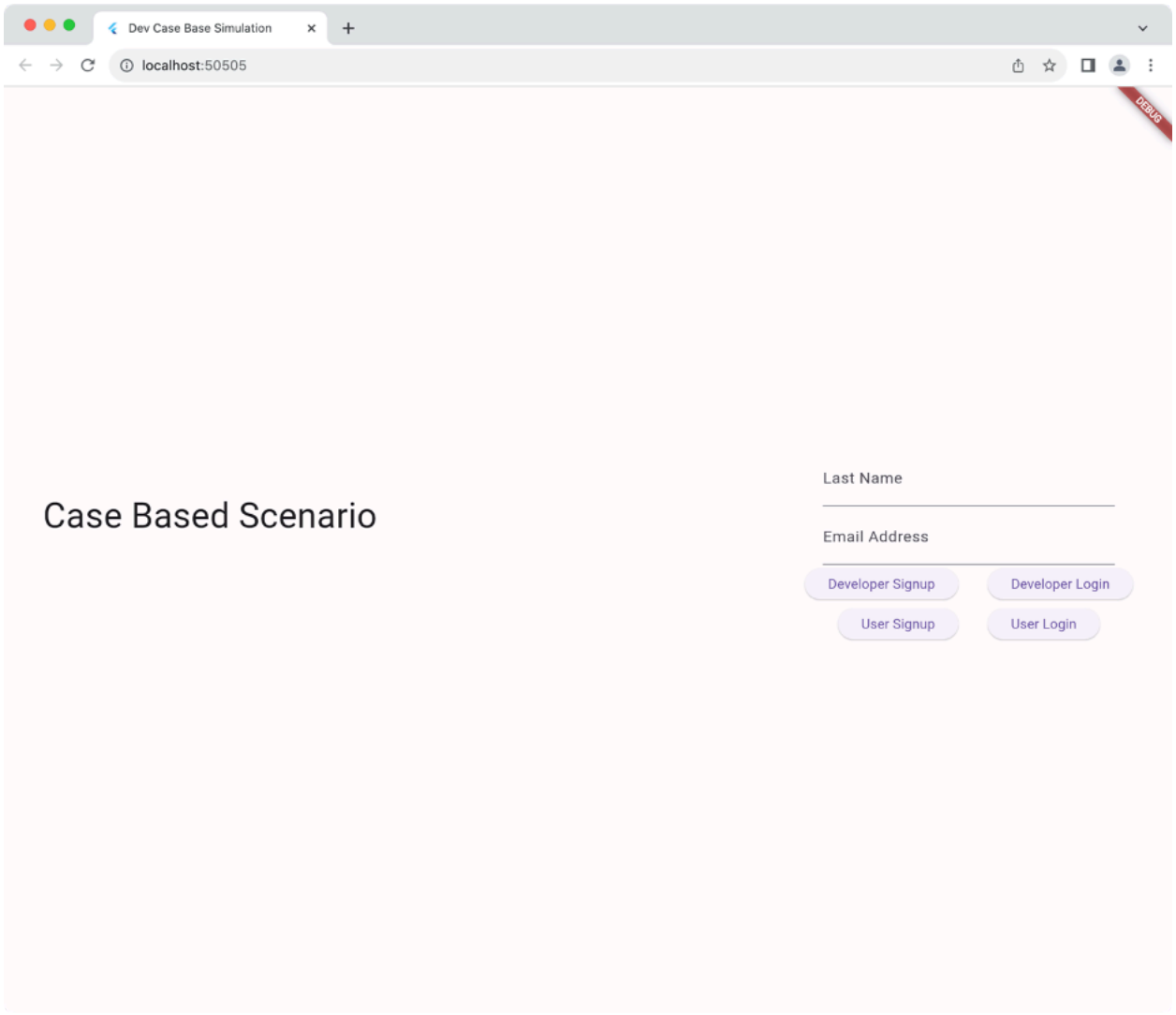


11. UI Design

This section will provide snapshots of key interfaces that the user will interact with within the case base simulation application.

11.1. General View

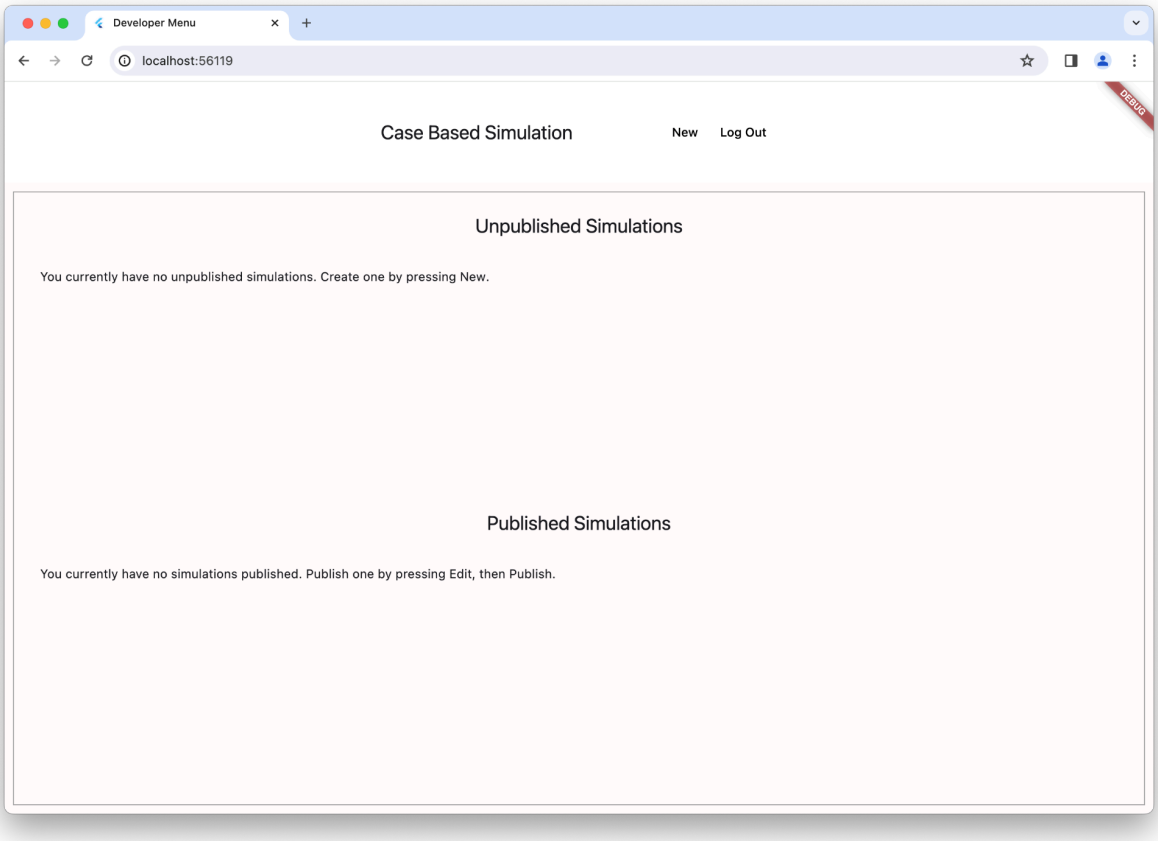
Login



This screen allows users to sign up or log in, offering the option to log in as either a developer or a general user.

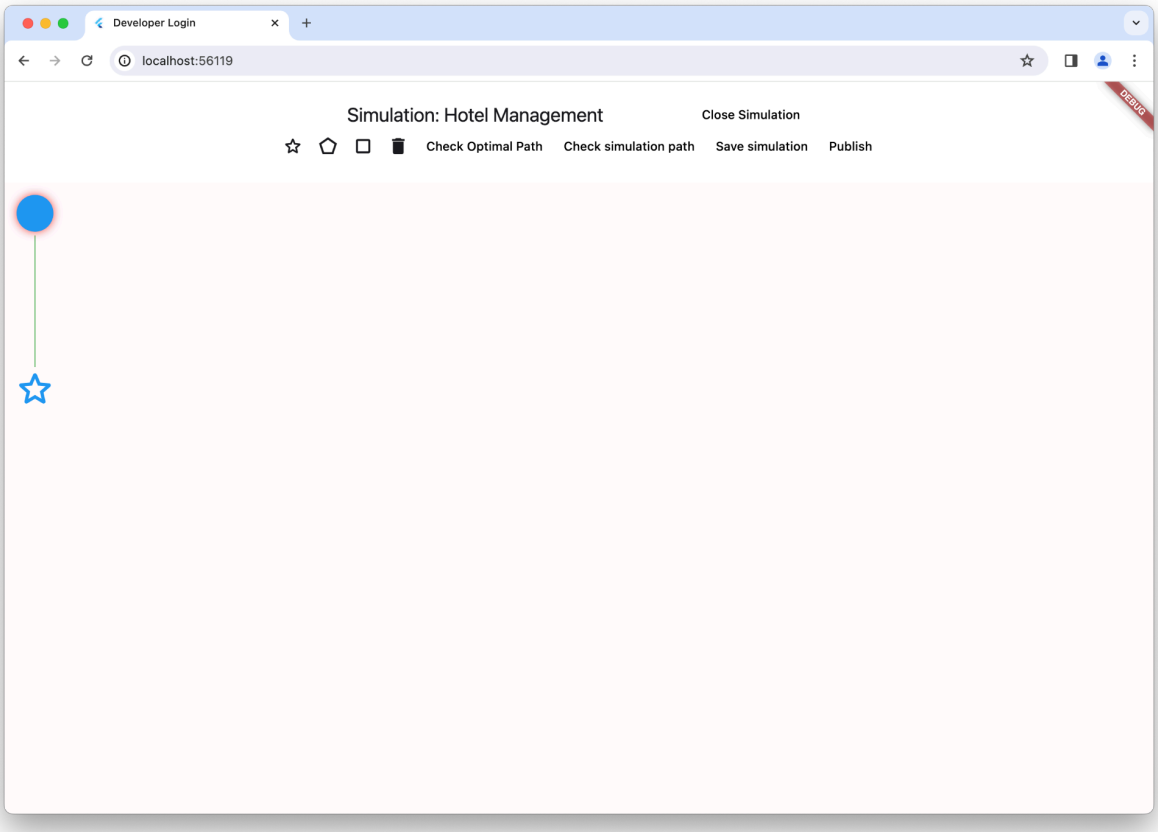
11.2. Developer View

Developer Menu



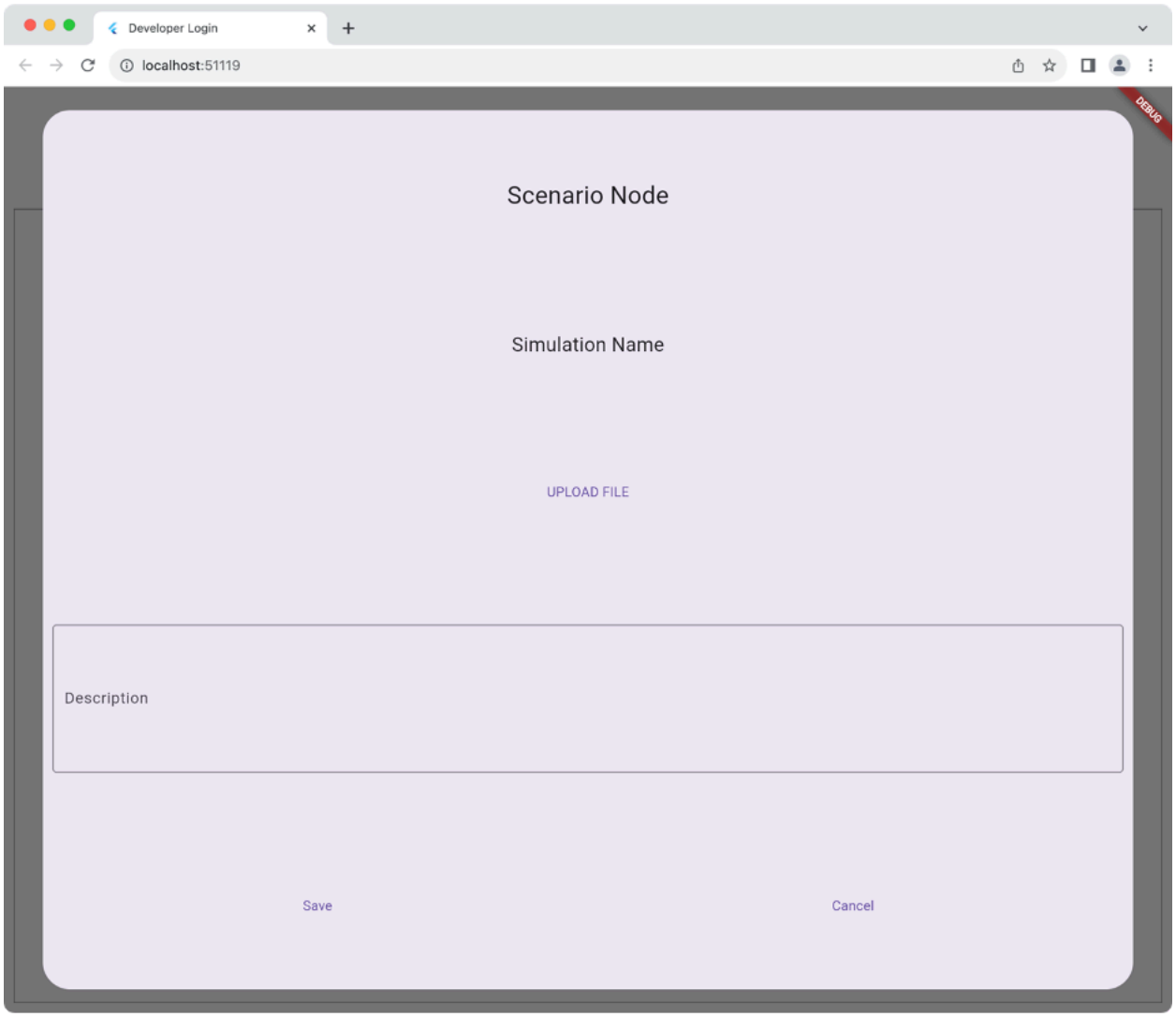
This menu allows developers to start a new simulation or edit and publish an existing one.

Developer Simulation Menu



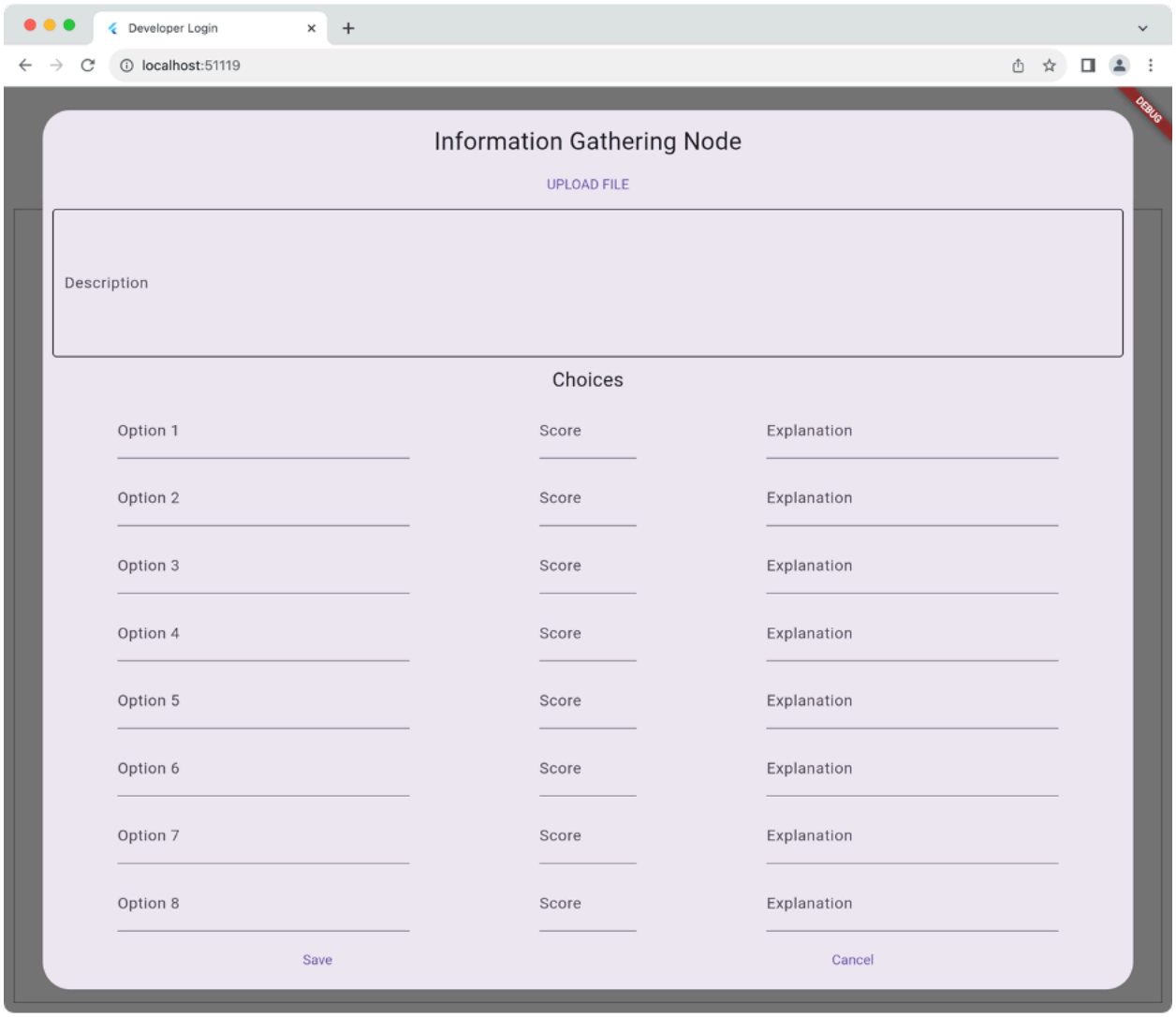
This simulation menu enables developers to edit simulations using an interactive node tree, providing a clear and navigable pathway.

Scenario Node



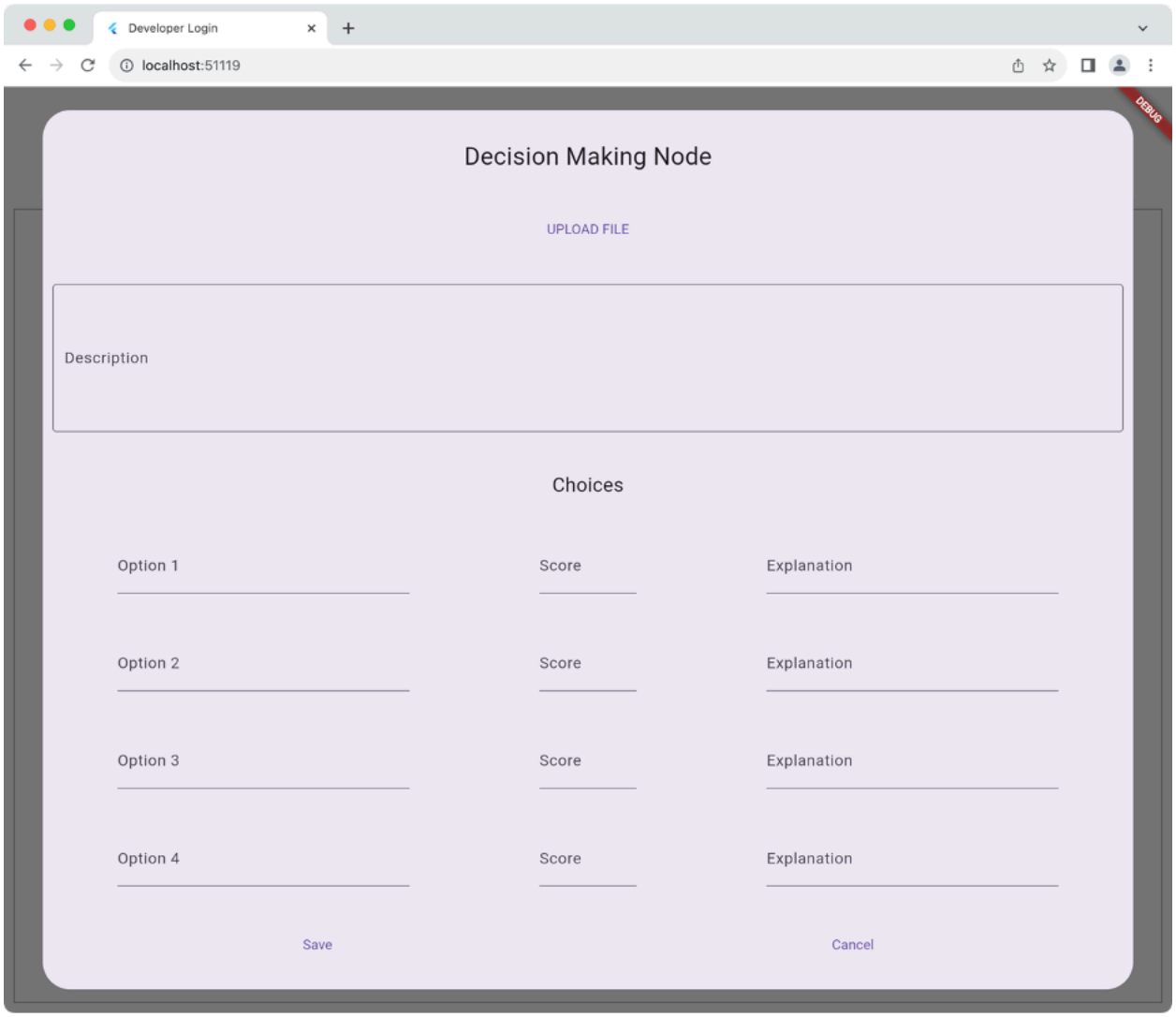
This node provides users with a scenario description, offering an introduction to the simulation.

Information Gathering Node



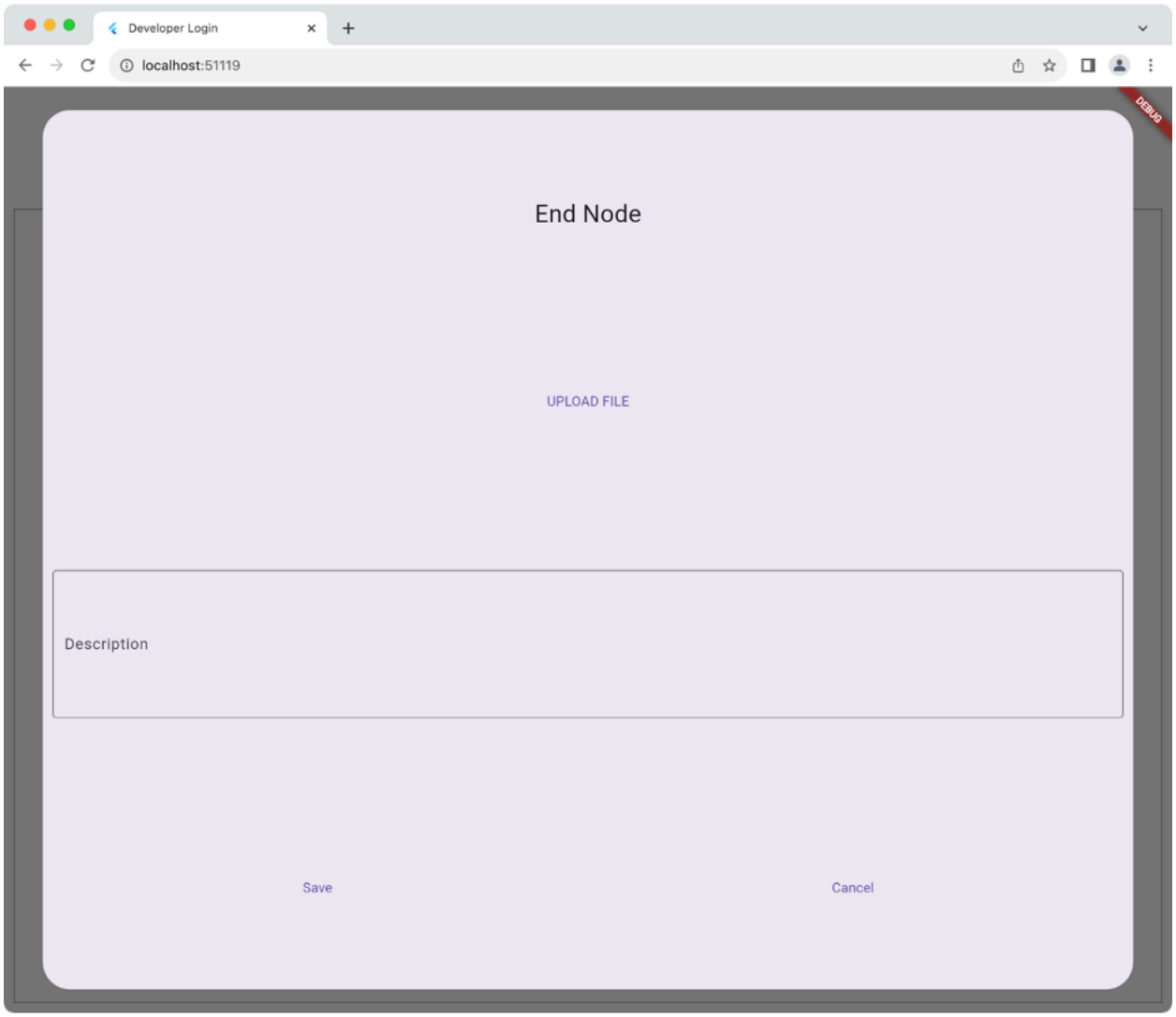
Each information gathering node offers up to eight options, providing specific details to the user upon selection, followed by an explanation at the end of the simulation.

Decision Making Node



This decision-making node features up to five options, each accompanied by explanations that are displayed at the end of the simulation.

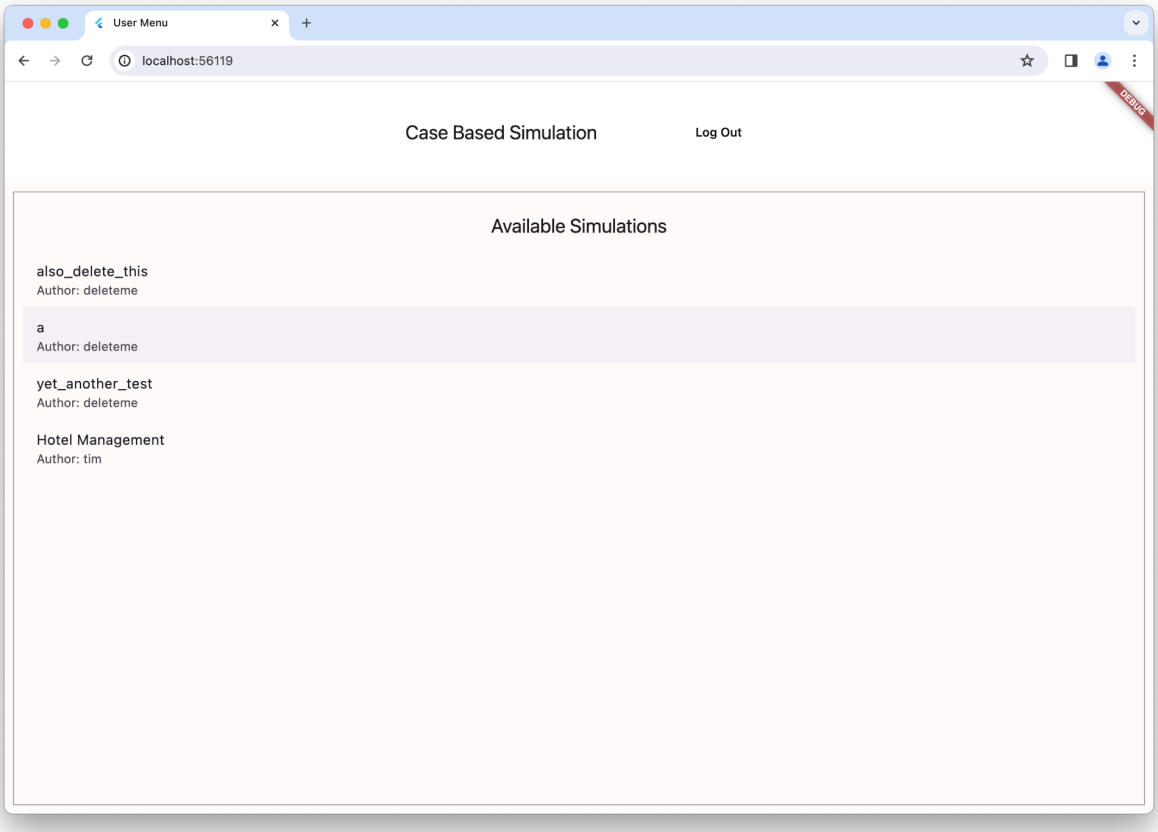
End Node



This end node includes a description that will be displayed at the conclusion of the node.

11.3. User View

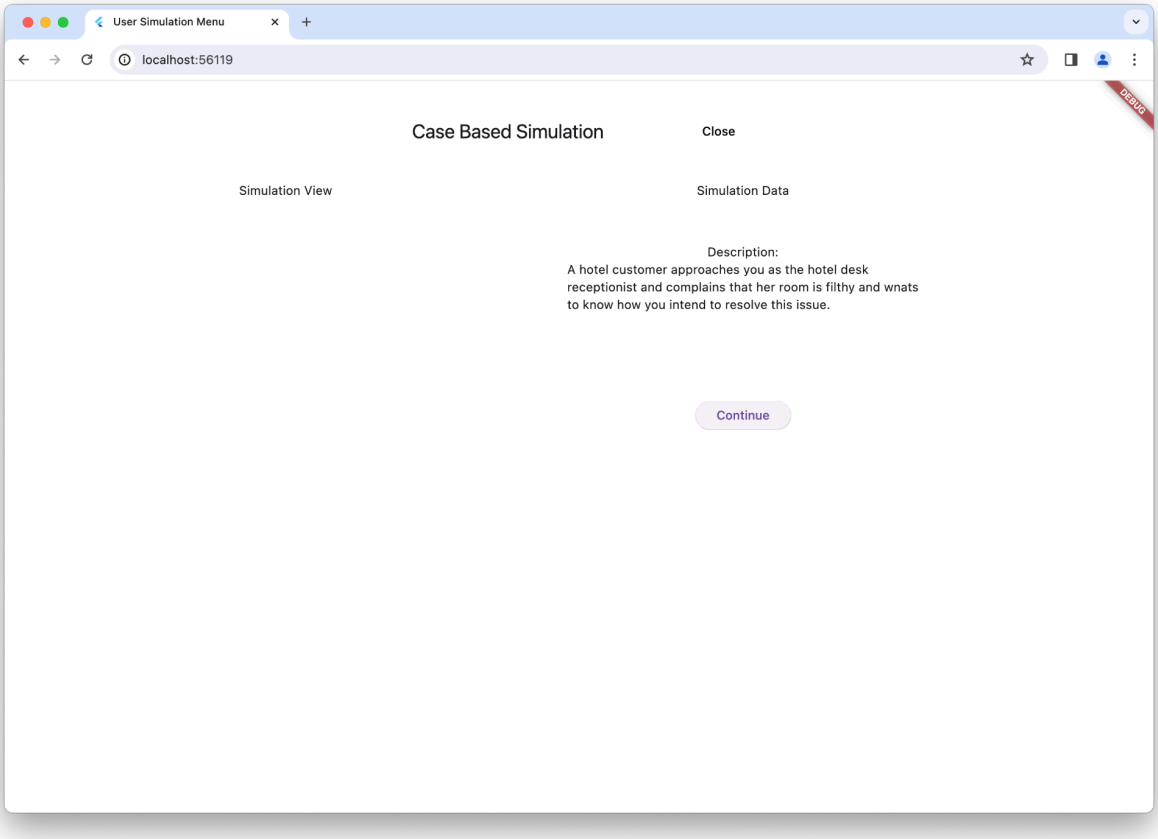
User Main Menu



This user menu displays a list of available simulations for users to select from or view their scores.

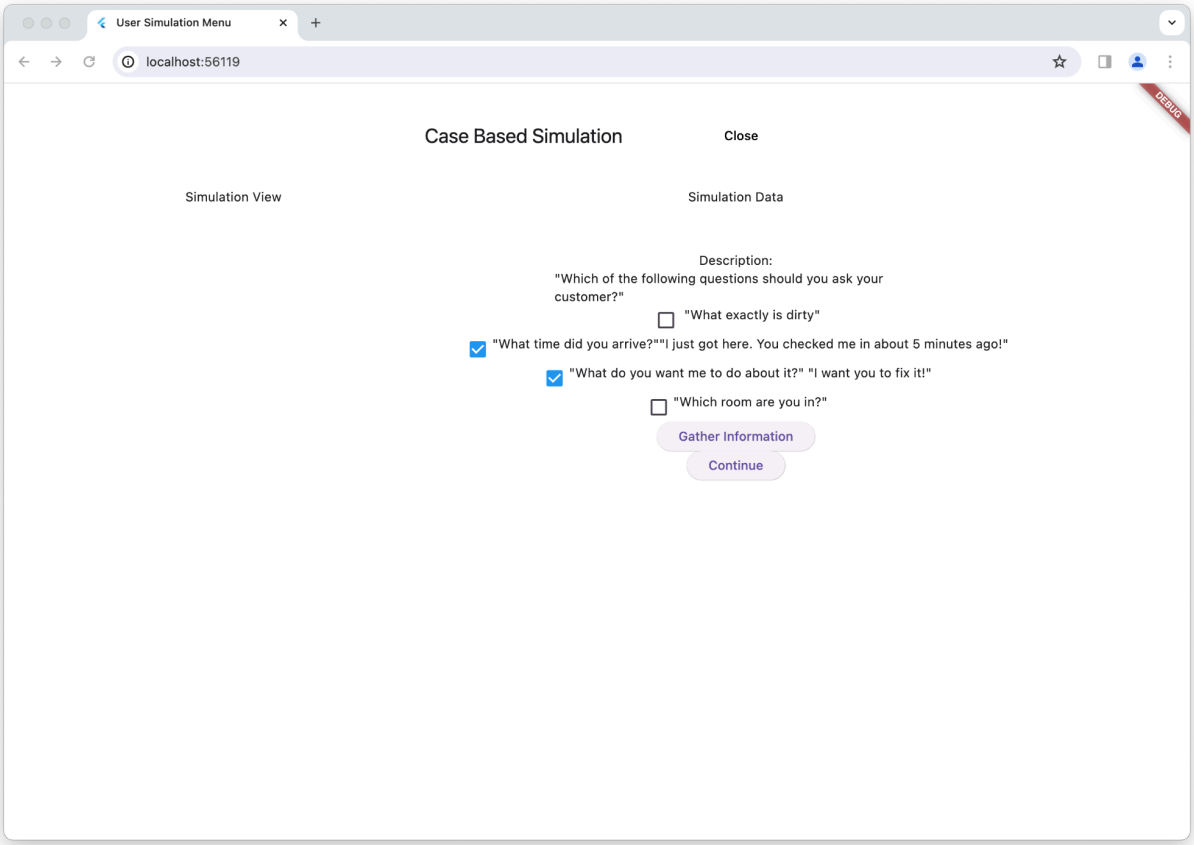
User Simulation Player

Scenario Node View



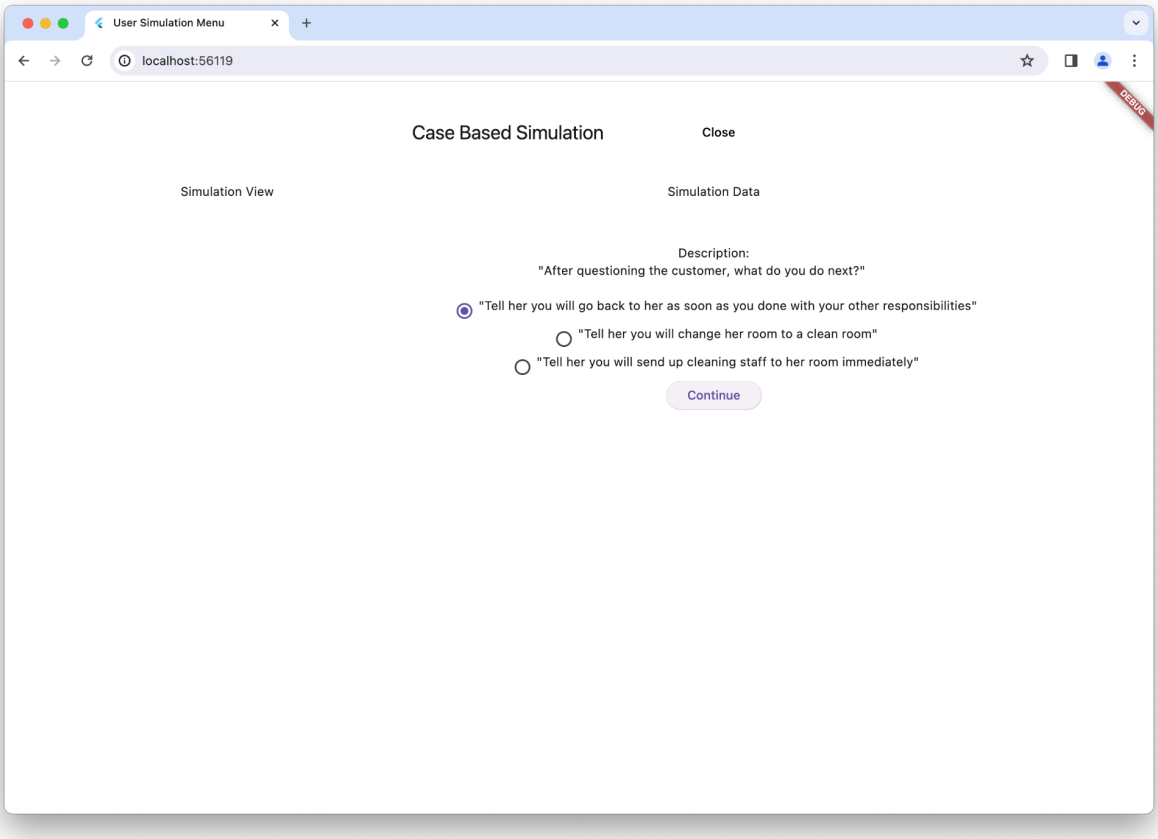
This scenario node view from the user side displays a description input by the developer.

Information Node View



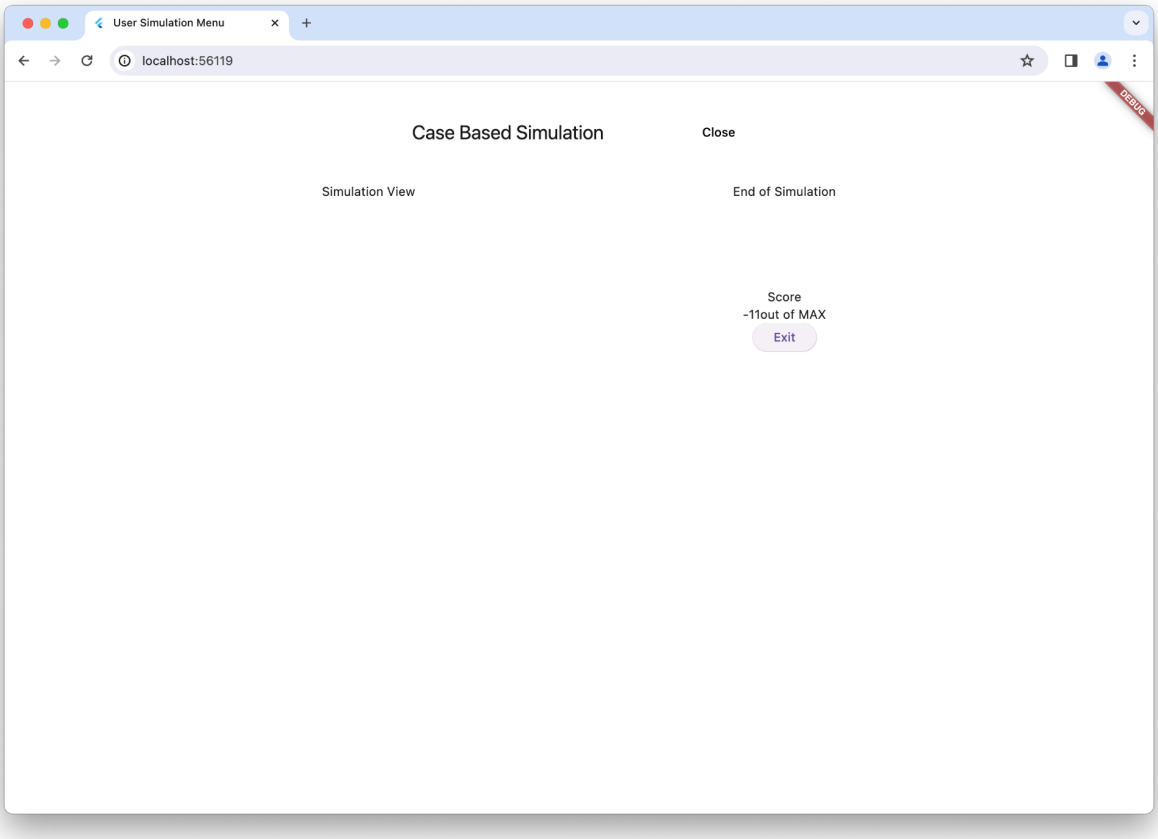
The information node delivers details to the user after they have selected options.

Decision Making Node View



In the decision-making node, the user reaches a point where they must select a choice.

End Node View



End node featuring a description and the final score.

Appendix A: Glossary of terms

#	Term	Description
1	Developer	A person who creates/builds a simulation using the product.
2	User	A person (a student/trainee) that uses the simulation to be evaluated on their performance of responses and actions within the simulation environment.
3	Node	Each step in the simulation path. There are 4 types of nodes: 1. Scenario (the start of the simulation) 2. End (the end of the simulation) 3. Information Gathering (IG) 4. Decision-Making (DM)
4	Simulation	The entire set of nodes that the user can traverse in their decision making and information gathering process. These are made by the developer and completed by the user.
5	Scenario Node	The problem set. What introduces the user to the environmental situation/case/problem they need to solve. The Scenario node is the first node. It introduces the problem set to the user. There is only one Scenario child node.
6	Information Gathering Node	The type of node that allows the user to check (appropriate or inappropriate) data to better understand the current conditions of the simulation. There is only one IG child node.
7	Decision-Making Node	The type of node that allows the user to make choices or decisions There are between 1 to 5 child nodes.
8	End node	The type of node that would be displayed at the end of a simulation with a final score and display the optimal path with explanations.

Appendix B: Revision History

Version	Date	Updated By	Comments
1.0	04/13/2024	Kuan Liu	Initial document creation