МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
к.п.н., доцент кафедры ЭВМ ЮУрГУ

_____ Ю.Г. Плаксина

“___”_____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

“___”_____ 2020 г.

Разработка компонента «Редактор курсов» для программной системы ECoD

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2020.308-041.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ Н.С. Силкина

Автор работы,
студент группы КЭ-403
_____ М.Н. Глизница

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
“___”_____ 2020 г.

Челябинск-2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
03.02.2020

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Глизнице Максиму Николаевичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 24.04.2020 № 627)
Разработка компонента «Редактор курсов» для программной системы ECoD.
2. **Срок сдачи студентом законченной работы:** 01.06.2020.
3. **Исходные данные к работе**

3.1. Силкина Н.С., Соколинский Л.Б. Структурно-иерархическая дидактическая модель электронного обучения // Вестник Южно-Уральского государственного университета. Серия «Вычислительная математика и информатика». 2019. Том 8, № 4. С. 56-83. DOI:10.14529/cmse190405

3.2. Ivanova, O.N., Silkina N.S. Competence-Oriented Model of Representation of Educational Content // Proceedings of the 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO'2017, Opatija, Croatia, May 22 26, 2017. IEEE, 2017. P. 791–794.

3.3. Силкина Н.С., Соколинский Л.Б. Система UniCST – универсальная среда электронного обучения // Системы управления и информационные технологии. 2010. № 2. С. 81-86.

3.4. Жигальская Н.С. Моделирование дидактической структуры электронных учебных комплексов // Вестник Южно-Уральского государственного университета. Серия «Математическое моделирование и программирование». 2008. № 27(127). Вып. 2. С. 4-9.

4. Перечень подлежащих разработке вопросов

4.1. Изучить структурно-иерархическую дидактическую модель.
4.2. Провести обзор моделей представления деревьев в реляционных СУБД.
4.3. Провести обзор средств визуализации древовидных структур.
4.4. Спроектировать редактор.
4.5. Реализовать редактор.
4.6. Провести тестирование редактора.
4.7. Внедрить редактор в систему ECoD.

5. Дата выдачи задания: 03.02.2020.

Научный руководитель,
ст. преподаватель кафедры СП Н.С. Силкина

Задание принял к исполнению М.Н. Глизница

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В настоящее время очевидна актуальность развития форм образования с использованием интернета и информационных технологий. Получение образования по такой форме позволяет учащемуся заниматься в удобное для себя время, в удобном месте, а также позволяет существенно снизить затраты на подготовку специалистов. При этом учащемуся необходимо иметь лишь компьютер, подключенный к сети интернет [3].

Этот вариант образования появился в России в 1990-е годы, а в 2003 стал юридически признанным. Но наибольшей активности его реализация достигла в последнее десятилетие. Из статистических данных [9] следует, что знания стареют каждые 3-5 лет, а профессиональные знания – каждые 2-3 года, в связи с чем необходимо их постоянное совершенствование. Дистанционное образование дает возможность массового непрерывного обучения.

При формировании единой информационной среды образования возможность передачи электронных учебных курсов и их частей с сохранением дидактической структуры образовательного контента от одного участника к другому приобретает особую важность [13]. Такой обмен сделает возможным для образовательных учреждений хранение и многократное использование не только различных курсов, но и их элементов. В связи с этим различными организациями были разработаны стандарты на структуру и представление элементов содержания электронных учебных курсов. Одним из таких стандартов является SCORM [15–18]. Стандарт SCORM определяет структуру пакета учебных материалов и интерфейс среды выполнения [2].

Однако, стандарты, определяющие принципы формирования дидактической структуры электронных учебных курсов, на сегодняшний день отсутствуют [6]. По этой причине перенос дидактических блоков из

одного курса в другой затруднен. В связи с этим использование стандарта SCORM недостаточно для получения максимального эффекта при внедрении электронного образования.

По этой причине на кафедре системного программирования ЮУрГУ была разработана структурно-иерархическая дидактическая (СИД) модель электронного образования. Основой СИД модели является граф-план курса, представляющий собой ориентированное дерево, узлами которого являются образовательные модули. В настоящее время программную поддержку СИД модели обеспечивает система ECoD (Electronic Course Designer), прототипом ECoD является система UniCST, которая описана в работах [3, 4, 7]. Таким образом, реализация интерфейса для работы с иерархическими структурами тесно связана с предложенным подходом, а поиск и реализация эффективного алгоритма хранения иерархических структур в реляционных базах данных являются актуальными задачами.

Постановка задачи

Целью выпускной квалификационной работы является разработка компонента «Редактор курсов» для программной системы ECoD. Для достижения поставленной цели необходимо решить следующие задачи:

1. изучить структурно-иерархическую дидактическую модель;
2. провести обзор моделей представления деревьев в реляционных СУБД;
3. провести обзор инструментов для визуализации деревьев;
4. спроектировать и реализовать систему;
5. провести тестирование системы и внедрить редактор в систему ECoD.

Структура и содержание работы

Работа состоит из введения, шести глав, заключения и списка литературы. Объем работы составляет 58 страниц, объем списка литературы – 21 источник.

В первой главе описывается структурно-иерархическая дидактическая модель, которая используется рассматриваемой системой ECoD для хранения образовательного контента.

Вторая глава посвящена обзору различных моделей хранения деревьев в базах данных, в ней рассматриваются примеры деревьев и их представлений и производится их сравнение по различным критериям с целью подбора наиболее подходящей модели для хранения деревьев в рамках выполняемой работы.

В третьей главе рассматриваются различные библиотеки JavaScript для отображения древовидных структур на веб-страницах и производится их сравнение по различным критериям с целью подбора наиболее подходящей библиотеки для отображения деревьев в рамках выполняемой работы.

В четвертой главе представлена архитектура системы ECoD и использующейся ей базы данных, рассмотрены функциональные и нефункциональные требования к системе, приведена диаграмма вариантов использования системы, а также описана архитектура приложения и приведен макет пользовательского интерфейса.

В пятой главе детально описывается реализация приложения, разработанного на основе составленного списка требований.

В шестой главе приведены результаты тестирования разработанного приложения, подтверждающие корректность работы его основных функций.

В приложении А содержится спецификация вариантов использования разработанного редактора курсов.

1. СТРУКТУРНО-ИЕРАРХИЧЕСКАЯ ДИДАКТИЧЕСКАЯ МОДЕЛЬ

Структурно-иерархическая дидактическая модель относится к моделям дидактической структуры контента. Модели дидактической структуры контента основаны на понятии дидактического типа образовательного контента. Образовательный элемент некоторого дидактического типа является логически обособленным элементом образовательного контента, описывающим некоторую тему, либо оценивающим степень освоения темы учеником. Подробнее модель представлена в работе [8].

Основными объектами СИД модели являются:

- энциклопедии, состоящие из образовательных модулей;
- образовательные программы, включающие в себя образовательный стандарт и наборы универсальных, общепрофессиональных и профессиональных компетенций;
- электронные учебные курсы.

Электронный учебный курс в соответствии с СИД моделью может быть структурирован по дидактическим компонентам (вертикальное слоение) и уровням детализации (горизонтальное слоение). Модель также отличает поддержка возможности деления образовательных объектов на дидактические компоненты. Это позволяет автоматически верифицировать дидактическую полноту электронного учебного курса, включать элементы одного курса в другой, не теряя дидактическую структуру образовательного контента, а также выделять из курсов отдельный дидактический слой, на основе которого может быть создан самостоятельный учебно-дидактический документ.

Образовательные материалы в СИД модели хранятся в электронных учебных энциклопедиях (ЭУЭ). ЭУЭ состоят из модулей, каждый из которых содержит образовательные материалы, связанные с определенным понятием.

Компоненты модуля включают в себя:

- concept: теоретическое описание понятия;
- open_question: вопрос с открытым ответом, предназначенный для самопроверки;
- example: пример решения задачи с применением изучаемого понятия;
- exercise: задача, проверяющая способность применять освоенное понятие;
- close_question: вопрос с закрытым ответом, предназначенный для проверки учащегося;
- problem: практическое задание, которое формирует навыки применения освоенного понятия при решении задач;
- bibitem: библиографическая ссылка.

Разделяемому объекту контента (SCO – Sharable Content Object) соответствует модуль в СИД модели, что обеспечивает совместимость со стандартом SCORM. Компоненты модуля являются коллекциями, представляющими собой список элементов, имеющих соответствующий дидактический тип. Коллекция может быть пустой.

В СИД модели предусмотрены операции над объектами модели, в том числе над электронными учебными курсами и граф-планами. Под *граф-планом* понимается иерархическая структура электронного курса, где вершинам сопоставляются учебные модули, а связи отображают отношение «необходимо изучить». В таблице 1 приведены некоторые команды для работы над электронными учебными курсами и граф-планами.

Табл. 1. Команды для работы над электронными учебными курсами и граф-планами

Команда	Действие
INSERT_COURSE(edu_prog)	Добавление в образовательную программу нового (пустого) курса
GET_ENCYCLOPEDIA(course)	Доступ к энциклопедии курса
GRAPH_PLAN(course)	Доступ к корневой вершине граф-плана курса
FETCH_CHILD(node)	Получение указателя на текущую дочернюю вершину узла node
RESET_CHILD(node)	Перемещение внутреннего курсора на первую дочернюю вершину узла node
NEXT_CHILD(node)	Перемещение внутреннего курсора на следующую дочернюю вершину узла node
PRIOR_CHILD(node)	Перемещение курсора на предыдущую дочернюю вершину узла node
INSERT_CHILD(node)	Для добавления новой дочерней вершины узла node
DELETE_CHILD(node)	Удаление дочерней вершины узла node, на которую указывает курсор
SET_LINK(node, module)	Связывание узла граф-плана node с модулем module
module = GET_LINK(node)	Получение доступа к модулю энциклопедии курса, который ассоциирован с узлом граф-плана node
INSERT_UC_INTO_GP(node, number)	Добавление номера number в коллекцию номеров универсальных компетенций узла граф-плана node
DELETE_UC_FROM_GP(node, number)	Удаление номера number из коллекции номеров универсальных компетенций узла граф-плана node
UC_ISEXIST(node, number)	Определение наличия универсальной компетенции с номером number в узле node

Таким образом, структурно-иерархическая дидактическая модель дает возможность представить электронный учебный курс в виде иерархической структуры, называемой граф-планом, состоящей из учебных модулей и связей, задающих отношение «необходимо изучить».

2. АНАЛИЗ МОДЕЛЕЙ ПРЕДСТАВЛЕНИЯ ДЕРЕВЬЕВ

В рамках выпускной квалификационной работы необходимо найти наиболее подходящую модель хранения древовидных структур, которыми будут представлены учебные курсы, в реляционных базах данных. К наиболее популярным моделям относятся:

- модель «родитель-потомок» (Adjacency List) [5];
- модель материализованного пути (Materialized Path) [1];
- модель «вложенные множества» (Nested Sets) [14].

Для сравнения указанных моделей можно использовать такие критерии, как сложность изменения структуры дерева (добавление, перемещение и удаление узлов) и сложность осуществления выборки из дерева.

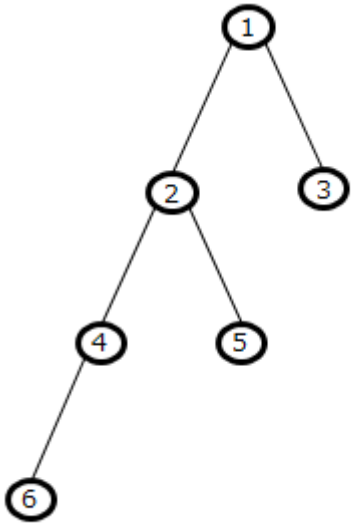
2.1. Модель «родитель-потомок»

Модель, известная под названием «родитель-потомок», является наиболее простой в реализации. В соответствии с этой моделью, каждая запись таблицы будет соответствовать узлу дерева и хранить его уникальный идентификатор и ссылку на родительский узел. Если узел является корнем, то ссылка на родителя в его записи будет пустой. Пример дерева и его представление согласно данной модели изображены на рисунке 1.

Низкая трудоемкость многих операций, изменяющих структуру дерева, относится к преимуществам данной модели. К примеру, для добавления нового узла нужно лишь вставить в таблицу его запись со ссылкой на родителя, а перемещение узла и всего его поддерева требует только изменения ссылки в этом узле.

Однако в данной модели существуют сложности при обращении к потомкам второго и дальнейших поколений. Эти поколения не имеют прямой связи с выбранным предком, поэтому для обращения к ним

необходимо использовать рекурсивные операции. Такие операции, помимо существенного увеличения нагрузки на сервер, не являются надежными, так как во многих серверах баз данных существует ограничение на максимальное количество вызовов рекурсивной процедуры, что означает, что такая процедура не сможет гарантированно извлечь все поддерево вне зависимости от его высоты.



ID узла	ID родителя
1	NULL
2	1
3	1
4	2
5	2
6	4

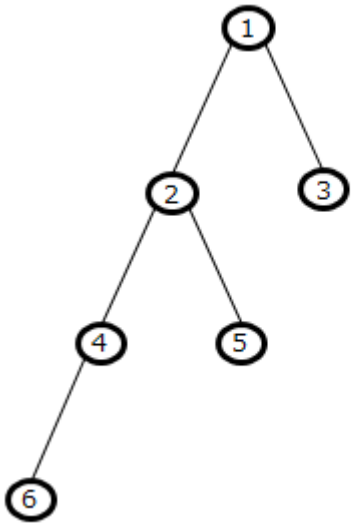
Рис. 1. Пример дерева и его представление
согласно модели «родитель-потомок»

Таким образом, модель «родитель-потомок» хорошо подходит для задач, в которых требуется активное изменение структуры дерева и не требуется выборка узлов с их потомками, но неэффективна для задач, в которых требуется выборка поддеревьев.

2.2. Модель материализованного пути

Модель материализованного пути похожа на модель «родитель-потомок», так как каждая запись об узле по-прежнему хранит уникальный идентификатор и ссылку на родителя. Но помимо этих данных, в каждой записи содержится список узлов на пути от корня дерева к данному узлу, упорядоченный от корня и объединенный в строку с использованием

некоторого символа в качестве разделителя. Пример дерева и его представление согласно данной модели приведены на рисунке 2.



ID узла	ID родителя	Путь к корню
1	NULL	/
2	1	/1/
3	1	/1/
4	2	/1/2/
5	2	/1/2/
6	4	/1/2/4/

Рис. 2. Пример дерева и его представление согласно модели материализованного пути

Простота изменения структуры, главное преимущество модели «родитель-потомок», в некоторой степени сохранено в модели материализованного пути. Операции добавления нового узла и перемещения поддерева теперь требуют дополнительных операций с полем пути к корню, что несколько усложняет их. Однако удаление поддерева больше не требует рекурсии, так как часть записи о пути к корню однозначно определяет все узлы, являющиеся потомками некоторого узла. По той же причине намного проще становится операция извлечения поддерева.

Однако поле пути к корню в данной модели является строковым, с длиной, зависящей от высоты дерева. Особенно высокие деревья, таким образом, будут требовать длинной строки, хранящей путь к корню, что может привести к серьезному увеличению размеров базы данных в памяти, а также к замедлению исполнения запросов.

По этим причинам, данная модель хорошо подходит для большинства задач, так как приспособлена и к изменению структуры дерева, и к выборке поддеревьев. Однако размер текстового поля будет ограничивать высоту дерева, а также приводить к увеличению общего размера базы данных.

2.3. Модель «вложенные множества»

Модель «вложенные множества» основана на работе с древовидными структурами не как с графами, а как со множествами, таким образом используя более совместимый с реляционной алгеброй подход. Древовидная структура представлена в данной модели как набор вложенных множеств. Для этого каждый узел содержит два целочисленных параметра: левый и правый ключ. Эти ключи удовлетворяют следующим условиям:

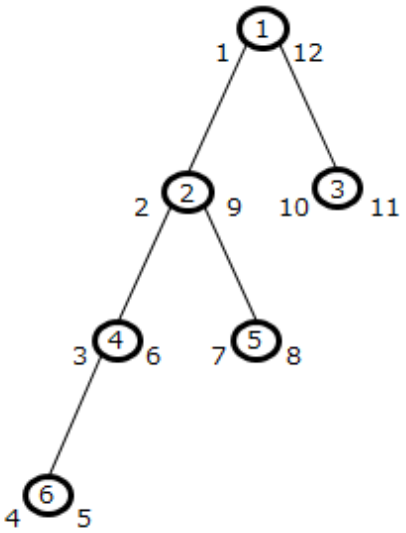
- правый ключ узла больше левого;
- левый ключ потомка больше левого ключа родительского узла;
- правый ключ потомка меньше правого ключа родительского узла;
- интервалы, определенные левыми и правыми ключами узлов, имеющих одного родителя, не должны пересекаться.

Пример дерева и его представление согласно данной модели приведены на рисунке 3.

Скорость и простота выборки является главным преимуществом этой модели. Границы, определенные ключами, позволяют выбрать всех потомков некоторого узла одним запросом, который не осуществляет сравнения строк и поэтому выполняется быстрее.

Операции, изменяющие структуру дерева, однако, при использовании данной модели имеют высокую сложность и низкую скорость. Добавление или удаление даже одного узла может потребовать изменения многих записей вплоть до всей таблицы. Перемещение узлов и

поддеревьев тоже усложнено, так как требует изменений как в перемещаемых узлах, так и в узлах, не затронутых перемещением.



ID узла	Левый ключ	Правый ключ
1	1	12
2	2	9
3	10	11
4	3	6
5	7	8
6	4	5

Рис. 3. Пример дерева и его представление согласно модели «вложенные множества»

Преимущества и недостатки этой модели приводят к выводу, что данная модель подходит для случаев, когда при выборке узлов требуется максимальная производительность, а изменение структуры дерева требуется редко. К таким случаям относится поставленная задача – хранение структуры электронных учебных курсов в системе ECoD.

Некоторые преимущества модели «вложенные множества» для рассмотренной задачи хранения структуры электронного учебного курса в базе данных приведены в статье [6].

В результате обзора литературы была выбрана наиболее подходящая для выбранной цели модель хранения деревьев в реляционной базе данных, а именно «вложенные множества». Редактор курсов должен быть способен к модификации структуры, оформленной в соответствии с этой моделью.

3. ВИЗУАЛИЗАЦИЯ

Визуализация древовидных структур в рамках веб-приложения для наибольшей совместимости с современными браузерами и широких возможностей интерактивности будет осуществляться с помощью сторонних библиотек JavaScript. В этом разделе рассмотрены некоторые доступные библиотеки. Критериями для сравнения являются качество визуализации и интерактивность.

Treant.js

Treant.js [20] – библиотека JavaScript для визуализации древовидных структур с помощью HTML, CSS и SVG (Scalable Vector Graphics).

Как видно из примеров применения библиотеки, доступных на официальном сайте, визуализация осуществляется с применением элементов HTML, таких как и <img>, внутри полотна. К плюсам этого подхода можно отнести тот факт, что текст внутри полотна обычно является выделяемым, а изображения доступны по прямым ссылкам. Это позволяет пользователям сохранять текст и изображения напрямую из визуализации дерева.

Однако эта библиотека не имеет встроенной реализации некоторых возможностей интерактивности, таких как выделение узлов различными цветами при кликах мышью, так как HTML не предназначен для создания динамического контента.

Большинство приведенных на официальном сайте примеров применения библиотеки являются статичными. В случае, когда полная структура дерева не умещается в полотно, предназначенное для отображения визуализации, используется прокрутка с помощью полос прокрутки.

Один из примеров применения библиотеки Treant.js приведен на рисунке 4.

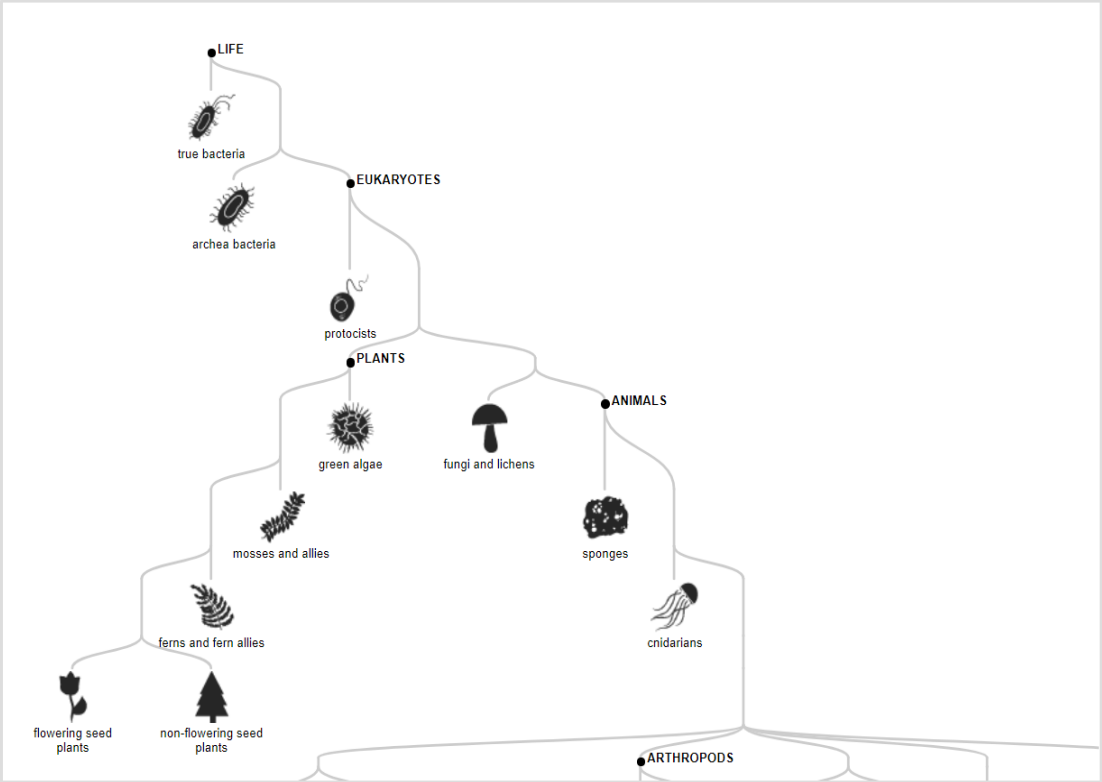


Рис. 4. Пример использования библиотеки Treant.js для визуализации дерева эволюции живых организмов

Sigma

Sigma [19] – библиотека JavaScript, предназначенная для отрисовки графов и позволяющая более наглядно представлять деревья на веб-страницах.

Примеры, приведенные на официальном сайте библиотеки, демонстрируют, что создаваемая библиотекой визуализация состоит из полотна с изображениями. Полотно поддерживает масштабирование и перемещение изображения, а также некоторую интерактивность, к примеру, выделение всех связанных с выбранным узлов.

Однако, библиотека не предназначена для работы с иерархическими структурами, такими как структура курса ECoD, из чего следует, что создание с помощью этой библиотеки визуализации курса, наглядно демонстрирующей его иерархическую структуру, может потребовать

слишком больших временных затрат по сравнению с другими рассмотренными библиотеками.

Один из примеров использования этой библиотеки приведен на рисунке 5.

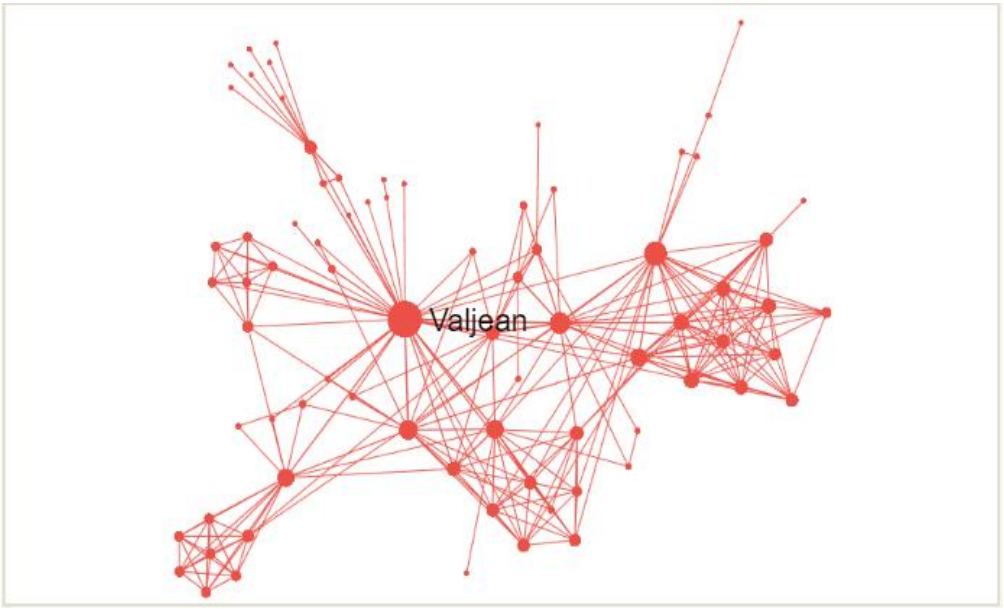


Рис. 5. Пример использования библиотеки Sigma для визуализации структуры данных

Dracula.js

Dracula.js [11] – библиотека, предоставляющая набор инструментов для отображения интерактивных графов.

К плюсам этой библиотеки относится простота, позволяющая быстро написать код для построения нужного графа. Библиотека также поддерживает некоторую визуальную интерактивность, такую как перетаскивание узлов.

Однако простота библиотеки означает, что реализация таких функций, как выбор узла с помощью мыши, не поддерживается библиотекой. Библиотека не предназначена для работы с иерархическими структурами, а также не имеет возможности настроить визуальное отображение узла. Dracula.js также не имеет практически никакой документации, так как ее официальный сайт предлагает только несколько

примеров. Кроме того, предлагаемая библиотекой интерактивность не является полезной для разработки редактора курсов, а требуемая интерактивность, такая как выделение узлов различными цветами, не поддерживается.

Один из примеров использования этой библиотеки приведен на рисунке 6.

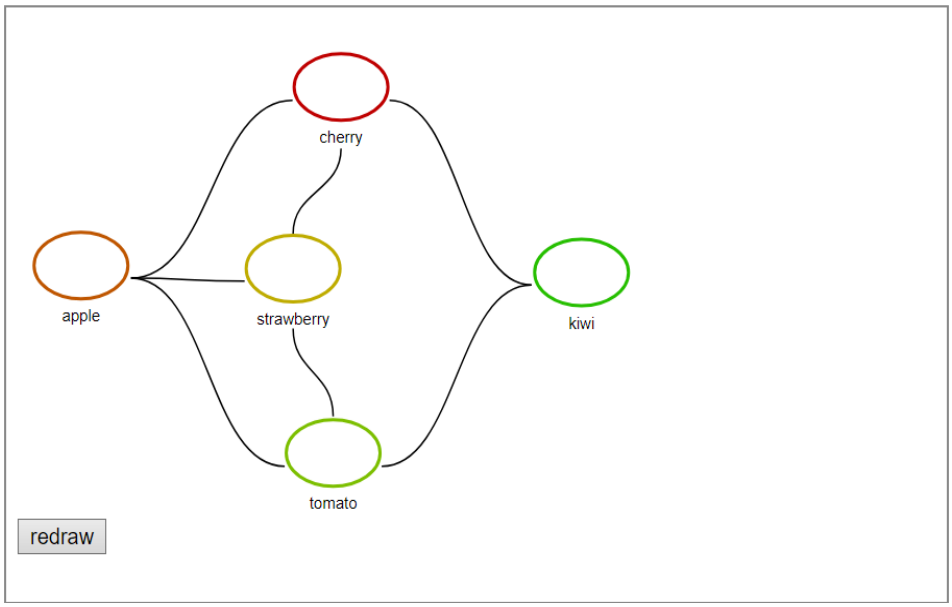


Рис. 6. Пример использования библиотеки Dracula.js для визуализации структуры данных

Vis.js

Vis.js [21] – динамическая библиотека для визуализации различных наборов данных, рассчитанная на работу с динамическими данными, а также взаимодействие с пользователем. Библиотека поддерживает работу с двухмерными и трехмерными графиками, временными линиями, а также сетями. Дерево при использовании этой библиотеки реализуется как частный случай сети.

Внутри полотна, создаваемого при использовании данной библиотеки, не используются элементы HTML. Это ограничивает возможности по получению текстовых и других данных из элементов полотна, но позволяет отображенной структуре претерпевать самые

различные изменения, включая перемещение, масштабирование, изменение цвета и даже смещение отдельных узлов, которое может производиться с эмуляцией физики взаимодействия узлов друг с другом.

Библиотека имеет встроенные функции, позволяющие обрабатывать различные действия пользователя, в том числе клики мышью. Это позволяет реализовать интерактивное взаимодействие пользователя с деревом, в том числе выбор цели для действия прямо на изображении дерева с помощью мыши.

На рисунке 7 приведен пример использования библиотеки vis.js для визуализации древовидной структуры.

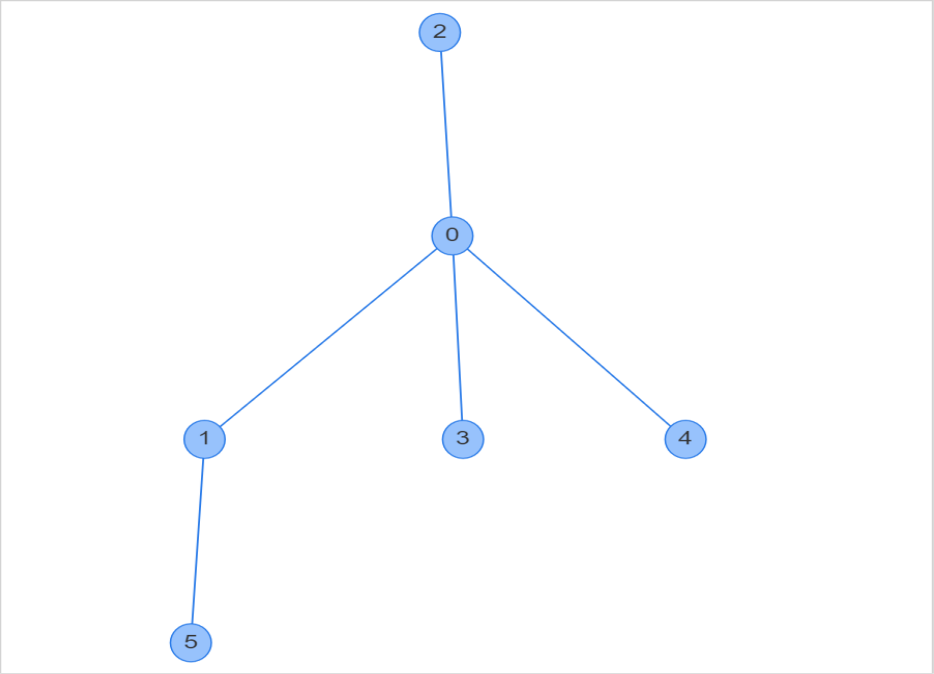


Рис. 7. Пример использования библиотеки vis.js для визуализации дерева

Поскольку библиотека vis.js предоставляет подробно настраиваемую визуализацию для дерева курсов, может адаптироваться к иерархическим структурам, а также предлагает наиболее широкие возможности по интерактивности, включая выбор узлов с помощью мыши и выделение выбранных узлов другим цветом, для реализации редактора курсов в системе ECoD была выбрана именно она.

4. ПРОЕКТИРОВАНИЕ

4.1. Архитектура системы ECoD

Архитектура системы ECoD, для которой производится разработка редактора курсов, представлена на рисунке 8.

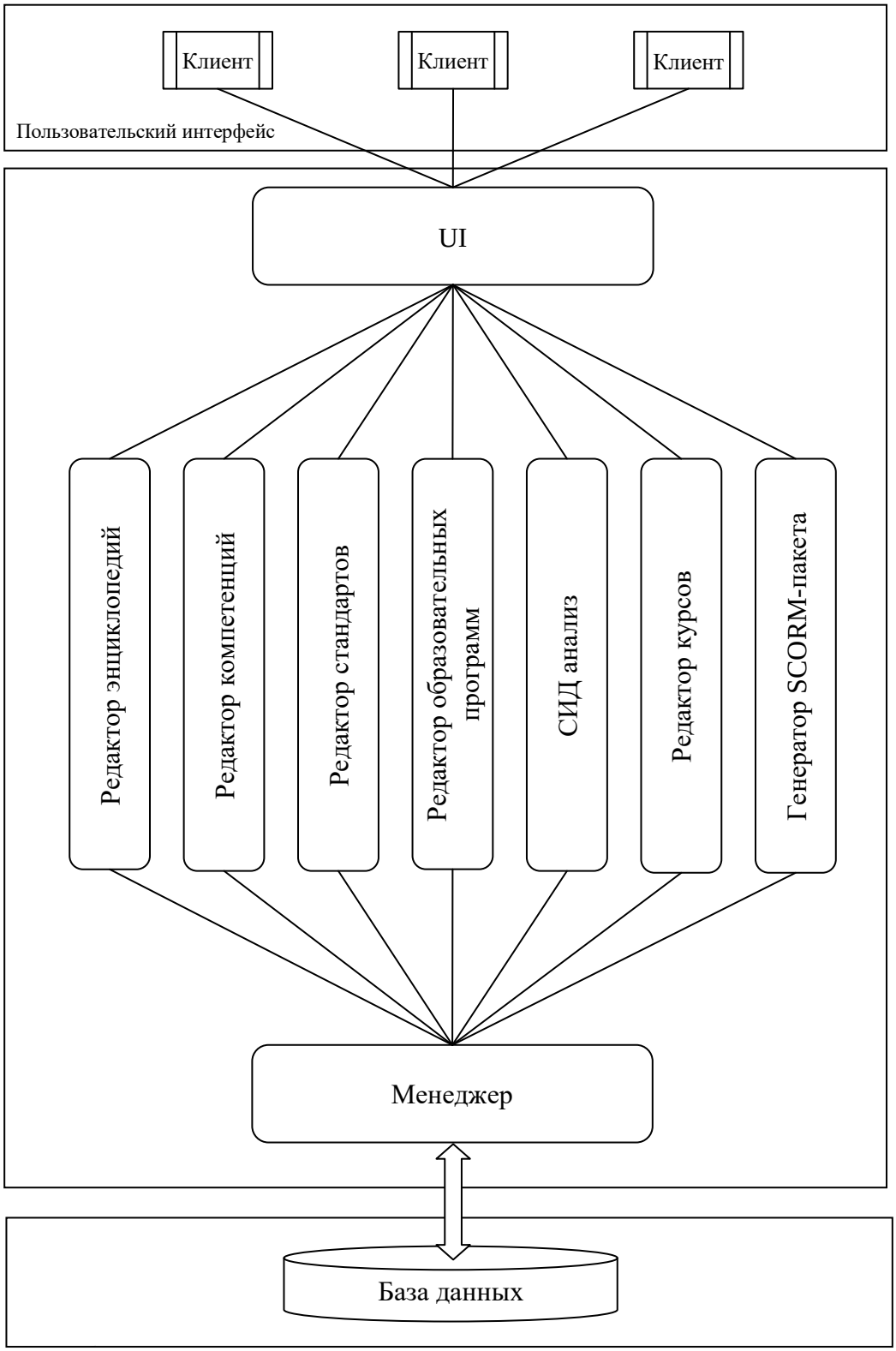


Рис. 8. Архитектура системы ECoD

Система состоит из следующих компонентов:

- компонент «UI» предоставляет пользователю доступ в соответствии с его правами в системе;
- компонент «Редактор энциклопедий» предоставляет пользователю интерфейс для выполнения операций создания, модификации и удаления энциклопедий и модулей;
- компонент «Редактор компетенций» предоставляет пользователю интерфейс для выполнения операций создания, модификации и удаления компетенций, создания источника компетенций;
- компонент «Редактор стандартов» предоставляет пользователю интерфейс для выполнения операций создания, модификации и удаления стандартов, редактирования списка универсальных и общепрофессиональных компетенций стандарта;
- компонент «Редактор образовательных программ» предоставляет пользователю интерфейс для выполнения операций создания, модификации и удаления образовательных программ, выбор стандарта, на основе которого разрабатывается образовательная программа, редактирования списка профессиональных компетенций программы, редактирования списка электронных учебных курсов программы;
- компонент «СИД анализ» предоставляет пользователю интерфейс для выполнения операций для анализа образовательных программ и электронных учебных курсов;
- компонент «Редактор курсов» предоставляет пользователю интерфейс для выполнения операций управления граф-планом курса (создание узлов, перемещение узлов, удаление узлов), выбора модуля для узла граф-плана, модификации атрибутов курса;
- компонент «Генератор SCORM-пакета» предоставляет пользователю интерфейс для выполнения операций для генерации

SCORM-пакета – пользователь может выбирать узлы и/или дидактические слои для генерации SCORM-пакета;

- компонент «Менеджер» обеспечивает взаимодействие системы с базой данных.

Схема базы данных системы ECoD представлена на рисунке 9.

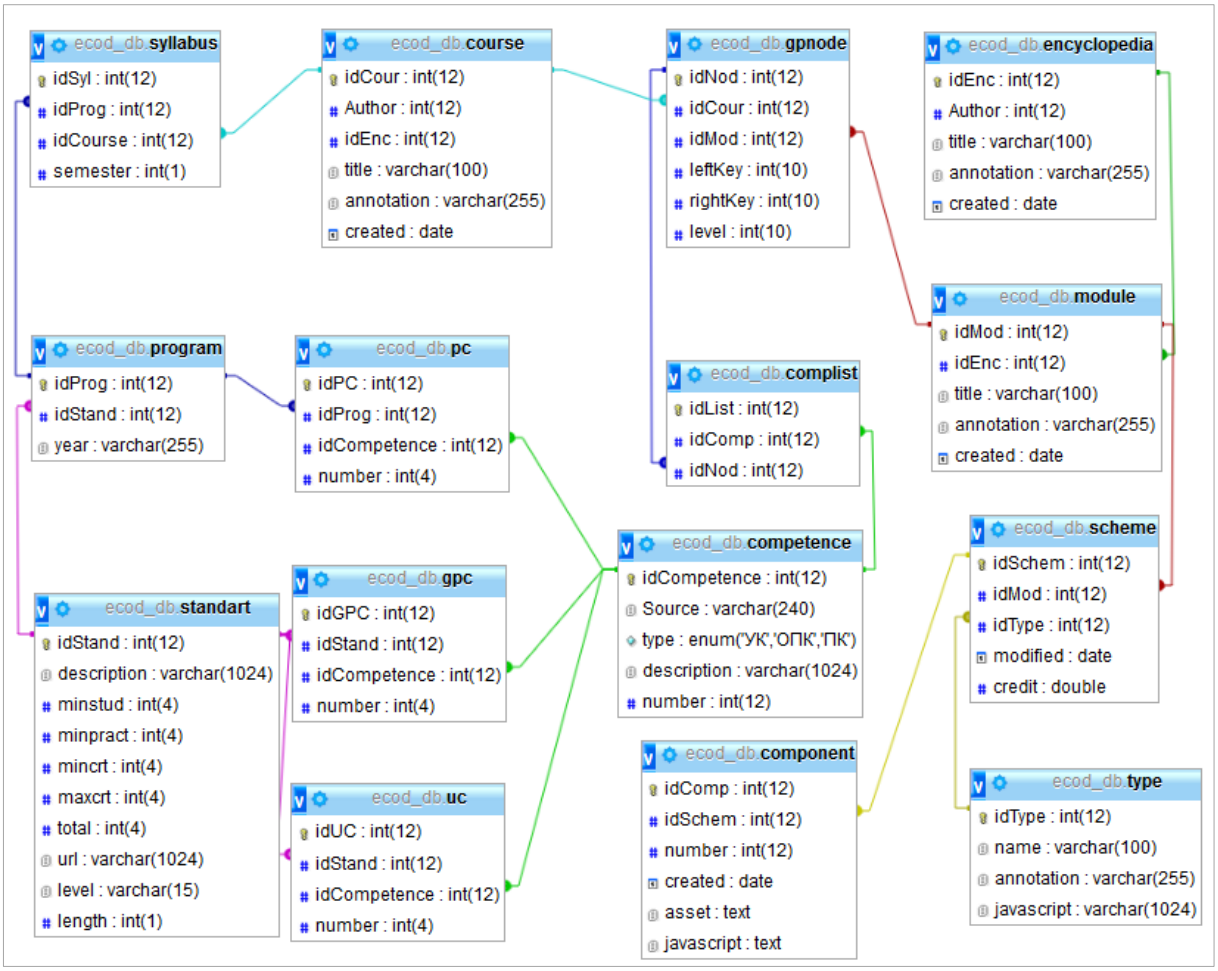


Рис. 9. Схема базы данных

Таблица Encyclopedia предназначена для хранения информации об энциклопедиях. Таблица Module предназначена для хранения информации о модулях энциклопедий. Таблица Scheme предназначена для хранения информации о схемах модулей. Таблица Component предназначена для хранения компонентов модулей. Таблица Course предназначена для хранения электронных учебных курсов. Таблица GPNode предназначена для хранения граф-планов электронного учебного курса. Информация о

пользователях хранится в таблице «User», доступ к остальным сущностям регулируется с помощью механизма прав доступа. В представленной схеме базы данных эта таблица опущена, так как данный механизм реализован во всех современных CMS (Content Management System).

Разрабатываемый редактор работает с данными, представленными в таблицах GPNode и Module. При этом в таблицу GPNode вносятся изменения, в то время как таблица Module используется для получения дополнительных данных.

Описание семантики этих двух таблиц представлено в таблицах 2 и 3.

Табл. 2. Атрибуты таблицы GPNode

№	Атрибут	Ключ	Тип	Семантика
1.	idNod	primary	uniqueid	Идентификатор узла в граф-плане
2.	idCour	foreign	uniqueid	Идентификатор курса
3.	idMod	foreign	uniqueid	Идентификатор модуля
4.	leftKey	-	int	Левый ключ узла в дереве граф-плана
5.	rightKey	-	int	Правый ключ узла в дереве граф-плана
6.	level	-	int	Уровень узла в дереве граф-плана

Табл. 3. Атрибуты таблицы Module

№	Атрибут	Ключ	Тип	Семантика
1.	idMod	primary	uniqueid	Идентификатор модуля
2.	idEnc	foreign	uniqueid	Идентификатор энциклопедии
3.	title	-	varchar	Название модуля
4.	annotation	-	varchar	Описание модуля
5.	created	-	timedate	Дата создания модуля

В таблице GPNode для применения множественной модели хранения граф-планов (Nested Sets) необходимо для каждого узла граф-плана хранить три атрибута:

- level: уровень узла;
- leftKey: левый ключ;
- rightKey: правый ключ.

Уровень узла представляет собой целое число, равное числу дуг между узлом и корнем, при этом корень дерева имеет уровень узла, равный нулю. *Левый и правый ключи* представляют собой счетчики, значения которых вычисляются при «специфическом» обходе дерева. Такой обход начинается с корня дерева и идет слева направо. Левый ключ корня равен 1, а правый – 2*n, где n – количество узлов дерева.

4.2. Проектирование редактора курсов

В ходе проектирования были определены функциональные и нефункциональные требования к разрабатываемому редактору.

Функциональные требования.

- Пользователь должен иметь возможность просматривать визуализацию граф-плана курса.
- Пользователь должен иметь возможность добавлять узлы граф-плана курса.
- Пользователь должен иметь возможность удалять узлы граф-плана курса.
- Пользователь должен иметь возможность перемещать узлы дерева в рамках граф-плана курса.
- Пользователь должен иметь возможность определять и изменять для узлов граф-плана курса сопоставляемые им учебные модули.

Нефункциональные требования.

- Веб-приложение должно быть написано на языках PHP и JavaScript.
- Веб-приложение должно корректно функционировать в последних версиях наиболее популярных браузеров.

Варианты использования

При проектировании редактора был использован язык графического описания для объектного моделирования UML. Составленная диаграмма вариантов использования приведена на рисунке 10.

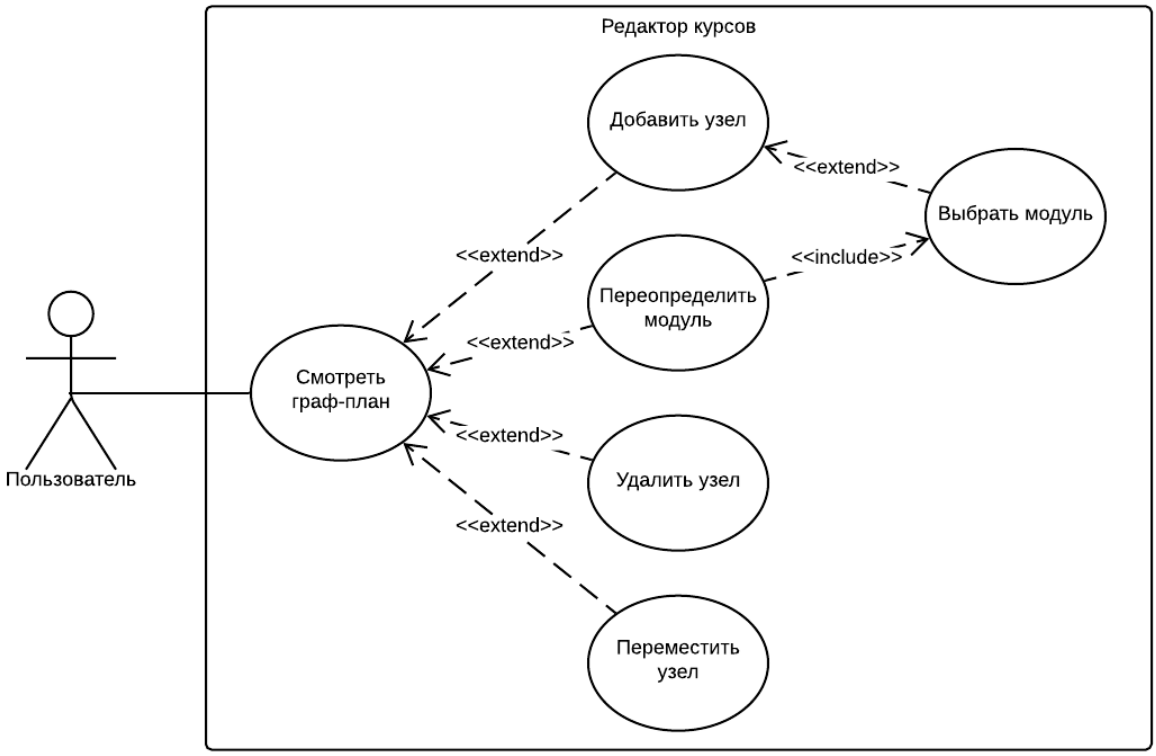


Рис. 10. Диаграмма вариантов использования

В рамках реализуемого редактора предусмотрен один актер – *пользователь*, то есть человек, который редактирует структуру курса.

Пользователь может *смотреть граф-план*. При запуске приложения формируется визуализация граф-плана курса, с помощью которой выбираются цели для остальных доступных пользователю операций.

Пользователь может *добавить узел* – добавить некоторый элемент как дочерний к одному из существующих. При этом пользователь может также *выбрать модуль*, который будет ассоциирован с добавляемым узлом или добавить элемент без указания модуля.

Пользователь может *переопределить модуль* – изменить ссылку на модуль, хранящуюся в конкретном узле.

Пользователь может *удалить узел* – стереть узел из структуры курса вместе со всеми его дочерними узлами.

Пользователь может *переместить узел* – сделать один из узлов курса дочерним по отношению к другому узлу.

В приложении А приведена спецификация для представленных вариантов использования.

Проектирование интерфейса

В ходе проектирования дизайна веб-приложения был создан эскиз интерфейса. Эскиз приведен на рисунке 11.

Цели операций будут выбираться с помощью мыши в рабочей области программы. Как правило, требуется выбрать две цели. Основная цель выбирается нажатием левой кнопкой мыши, а дополнительная цель выбирается нажатием правой кнопкой мыши. В нижней части экрана будет расположено меню работы с курсом. Меню должно включать в себя блоки, перечисленные далее.

- Выбранный модуль – панель, позволяющая выбрать модуль, который будет ассоциирован с добавляемым или изменяемым узлом.
- Формулировка операции – текст, при наведении мыши на кнопку кратко поясняющий выбранную операцию.
- Добавить дочерний узел – добавить узел как дочерний к выбранному.
- Изменить номер модуля – редактировать ссылку на модуль, хранящуюся в узле.
- Переместить узел – сделать выбранный узел дочерним для другого выбранного узла.
- Удалить узел – удалить выбранный узел вместе с дочерними узлами.

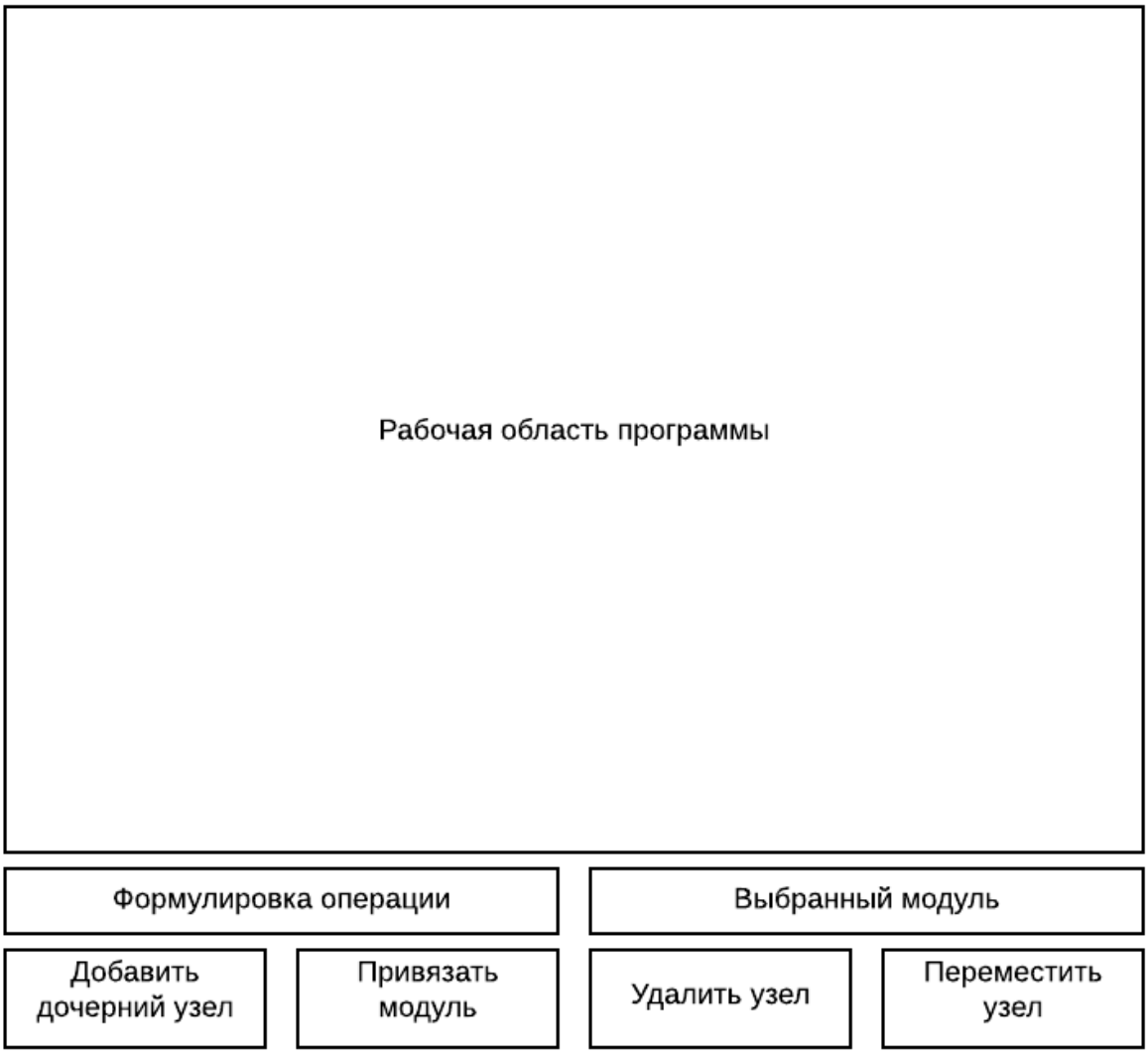


Рис. 11. Эскиз интерфейса приложения

Архитектура приложения

Архитектура веб-приложения включает в себя два компонента: компонент «Менеджер» и компонент «Визуализация дерева». Компонент «Менеджер» представляет собой набор функций для получения данных о структуре курса, а также для ее изменения. Компонент «Визуализация дерева» обеспечивает построение граф-плана курса и его вывод на экран, а также работу пользовательского интерфейса на стороне клиента. Архитектура приложения представлена на рисунке 12.

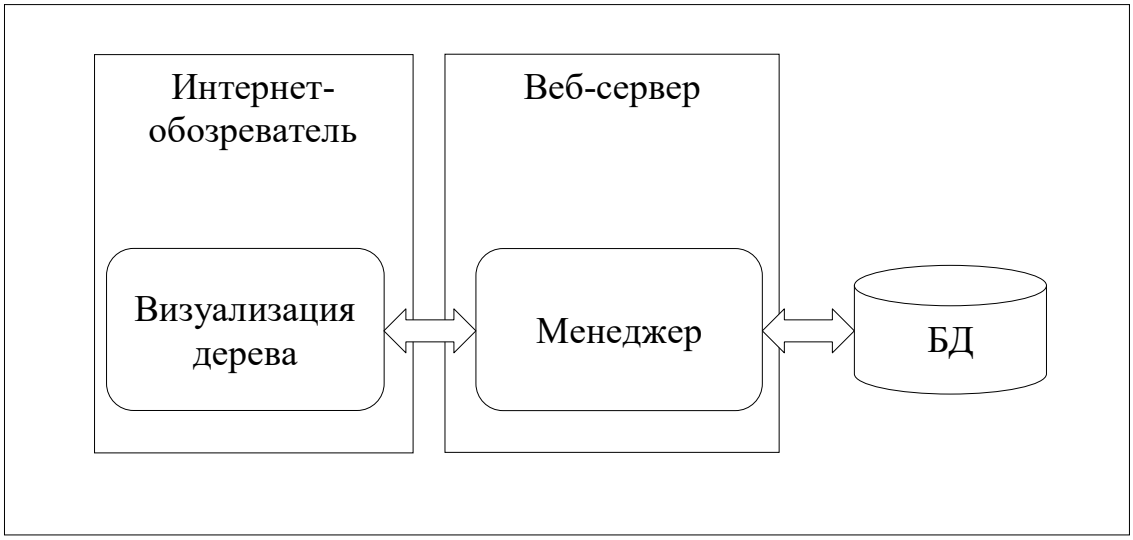


Рис. 12. Архитектура редактора курсов

К основным операциям, выполняемым компонентом «Менеджер», относятся перечисленные далее.

- Получение информации о структуре курса и ее сохранение. После получения необходимой информации о структуре курса из базы данных система передает необходимые данные для построения граф-плана компоненту визуализации дерева.
- Добавление узла. Операция выполняется с помощью функции, получающей как параметр ID узла, дочерним к которому будет добавленный узел.
- Переопределение модуля. Операция выполняется с помощью функции, получающей как параметры ID узла, в котором требуется переопределить модуль и номер модуля, который будет записан в этот узел.
- Удаление узла. Операция выполняется с помощью функции, получающей как параметр ID удаляемого узла. Если у удаляемого узла есть дочерние узлы, они также будут удалены.
- Перемещение узла. Операция выполняется с помощью функции, получающей как параметры ID перемещаемого узла и ID узла, дочерним к которому станет перемещаемый узел.

Прочие операции, выполняемые компонентом «Менеджер», являются вспомогательными.

Компонент «Визуализация дерева» выполняет основные операции, перечисленные далее.

- Вывод граф-плана курса на веб-страницу. Для этой операции используются данные, полученные от компонента выполнения запросов.
- Выбор узла или узлов, с которыми будет производиться операция. Выбор можно произвести на визуализированном граф-плане курса с помощью курсора мыши.
- Отображение надписи, поясняющей выбранную операцию при наведении курсора мыши на одну из кнопок. При некорректности параметров операции надпись должна отражать ошибку.
- Блокировка операций с некорректными параметрами и отображение сообщения об ошибке.

Таким образом, для разрабатываемого приложения была разработана диаграмма вариантов использования, определены функциональные и нефункциональные требования, спроектирован интерфейс и составлена схема архитектуры.

5. РЕАЛИЗАЦИЯ

В качестве клиентского приложения может использоваться любой стандартный интернет-обозреватель. Согласно статистике за март 2020 года, наиболее популярными браузерами для ПК являются Google Chrome (68,11%) и Mozilla Firefox (9,25%) [10].

В качестве веб-сервера использовалась сборка WampServer. В данную сборку входят такие компоненты, как:

- веб-сервер Apache;
- СУБД MySQL;
- интерпретатор PHP.

Файл приложения имеет 1047 строк кода. Из них:

- 81 строка – разметка HTML;
- 774 строки – реализация компонента «Вычислитель» на языке PHP;
- 192 строки – реализация компонента «Визуализация дерева» на языке JavaScript.

Реализация базы данных

Данные о структуре граф-плана хранятся в таблице под названием gpnode. Внешний вид таблицы в СУБД MySQL показан на рисунке 13. На рисунке 14 приведен код создания данной таблицы для используемой СУБД.

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1	**idNod** 🔑	int(11)			No	*None*		AUTO_INCREMENT
☐	2	**idCour** 🔗	int(11)			No	*None*		
☐	3	**idMod** 🔗	int(11)			Yes	*NULL*		
☐	4	**leftKey**	int(11)			No	*None*		
☐	5	**rightKey**	int(11)			No	*None*		
☐	6	**level**	int(11)			No	*None*		

Рис. 13. Вид таблицы gpnode в СУБД MySQL

```
CREATE TABLE IF NOT EXISTS `gpnode` (
  `idNod` int(11) NOT NULL AUTO_INCREMENT,
  `idCour` int(11) NOT NULL,
  `idMod` int(11) DEFAULT NULL,
  `leftKey` int(11) NOT NULL,
  `rightKey` int(11) NOT NULL,
  `level` int(11) NOT NULL,
  PRIMARY KEY (`idNod`),
  KEY `idCour` (`idCour`),
  KEY `idMod` (`idMod`)
) ENGINE=InnoDB AUTO_INCREMENT=385 DEFAULT CHARSET=utf8 COL-
LATE=utf8_unicode_ci;
```

Рис. 14. Код, создающий таблицу gpnode в СУБД MySQL

Данные о модулях хранятся в таблице под названием module. Внешний вид таблицы в СУБД MySQL показан на рисунке 15. На рисунке 16 приведен код создания данной таблицы для используемой СУБД.

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1	**idMod** 🔑	int(11)			No	*None*		AUTO_INCREMENT
☐	2	**idEnc**	int(11)			Yes	*NULL*		
☐	3	**title**	varchar(100)	utf8_unicode_ci		Yes	*NULL*		
☐	4	**date**	date			Yes	*NULL*		
☐	5	**annotation**	varchar(100)	utf8_unicode_ci		Yes	*NULL*		

Рис. 15. Вид таблицы module в СУБД MySQL

```
CREATE TABLE IF NOT EXISTS `module` (
  `idMod` int(11) NOT NULL AUTO_INCREMENT,
  `idEnc` int(11) DEFAULT NULL,
  `title` varchar(100) COLLATE utf8_unicode_ci DEFAULT NULL,
  `date` date DEFAULT NULL,
  `annotation` varchar(100) COLLATE utf8_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`idMod`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 COL-
LATE=utf8_unicode_ci;
```

Рис. 16. Код, создающий таблицу module в СУБД MySQL

Реализация компонента «Менеджер»

Компонент «Менеджер» реализован на языке PHP, так как предназначен для работы на сервере (управление данными).

Операция получения информации о структуре курса и ее сохранения реализована в виде функции edInit, листинг которой представлен на рисунке 17. Функция выполняется при запуске веб-приложения.

```
function edInit($conn)
{
	$pointers = loadPointers();
	$pointers = padPointers($conn, $pointers);
	savePointers($pointers);
	if(isset($_REQUEST['course'])) $selCourse = $_REQUEST['course'];
	else $selCourse = 1;
	$sql = "select n1.idNod, n1.idCour, n1.leftKey, n1.rightKey,
n1.idMod, (select idNod from gpnode n2 where n2.idCour = n1.idCour and
n2.leftKey < n1.leftKey and n2.rightKey > n1.rightKey order by
n2.rightKey - n1.rightKey asc limit 1) as parent, n2.title from gpnode n1
left join module n2 on n1.idMod = n2.idMod where n1.idCour = $selCourse
group by n1.idNod order by n1.leftKey";
	$result = $conn->query($sql);
	if ($result->num_rows > 0)
	{
		echo("<div hidden>");
		$total = 0;
		while($row = $result->fetch_assoc())
		{
			echo(" Course: <span id = 'cid" . strval($total) . "'>"
. $row["idCour"] . "</span>");
			echo(" ID: <span id = 'id" . strval($total) . "'>" .
$row["idNod"] . "</span> LK: <span id = 'lk" . strval($total) . "'>" .
$row['leftKey'] . "</span> RK: <span id = 'rk" . strval($total) . "'>" .
$row['rightKey'] . "</span> TITLE: <span id = 'tl" . strval($total) .
"'>" . $row['title'] . "</span>");
			echo(" ParentID: <span id = 'pid" . strval($total) .
"'>" . $row["parent"] . "</span>");
			if(isset($pointers[$row["idNod"]]))
			{
				echo(" Pointer destination: <span id = 'pdid" .
strval($total) . "'>");
				if (FETCH_CHILD($row["idNod"]))
echo(FETCH_CHILD($row["idNod"]));
				else echo("NULL");
				echo("</span>");
			}
			echo("<br>");
			$total++;
		}
		echo(" Total: <span id = 'total'>" . $total .
"</span><br></div>");
	}
}
```

Рис. 17. Листинг функции получения информации о структуре курса и ее сохранения

Параметр $conn в этой и всех последующих функциях хранит соединение с базой данных MySQL.

Операция добавления узла реализована в виде функции newNode. Прототип функции:

function newNode($conn, $parentID, $modID)

- $parentID – ID узла, дочерним к которому будет добавляемый узел.
- $modID – номер модуля, с которым будет ассоциирован добавляемый узел. Если модуль не был указан, этот параметр хранит строку «NULL».

Диаграмма деятельности алгоритма добавления узла приведена на рисунке 18.

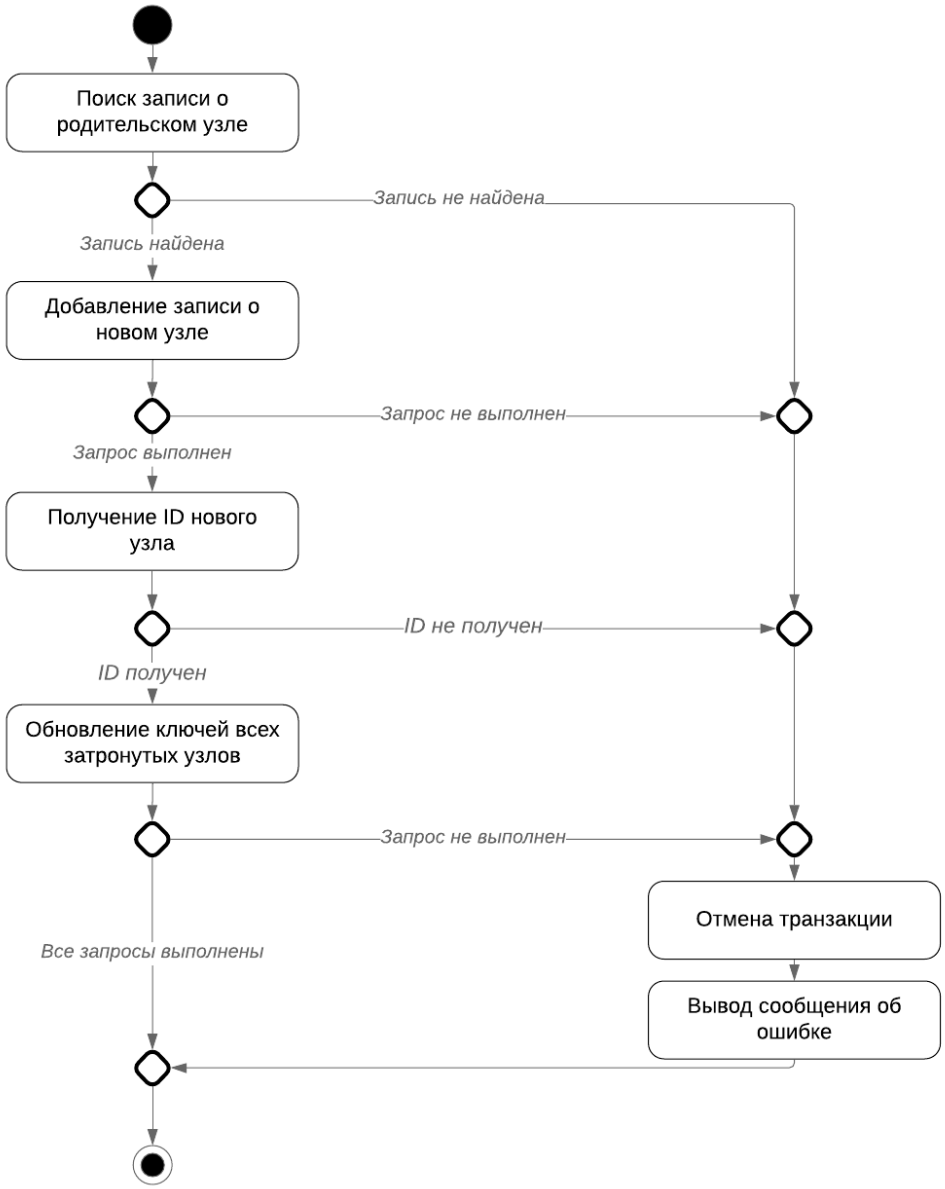


Рис. 18. Диаграмма деятельности алгоритма добавления узла

Операция переопределения модуля реализована в виде функции changeModID. Прототип функции:

function changeModID($conn, $ID, $modID)

- $ID – ID узла, в котором будет переопределен модуль.
- $modID – номер модуля, с которым будет ассоциирован узел.

Операция удаления узла реализована в виде функции deleteNode. Прототип функции:

function deleteNode($conn, $ID)

- $ID – ID удаляемого узла.

Операция перемещения узла реализована в виде функции moveNode. Прототип функции:

function moveNode($conn, $mID, $tID)

- $mID – ID перемещаемого узла.
- $tID – ID узла, дочерним к которому станет перемещаемый узел.

Получение идентификатора корневой вершины граф-плана курса реализовано в виде функции GRAPH_PLAN. Прототип функции:

function GRAPH_PLAN($courseid)

- $courseID – ID целевого курса.

Для упрощения выполнения составных операций были добавлены функции, работающие с так называемым внутренним курсором, который является ссылкой, указывающей на один из дочерних узлов каждого узла. Состояние внутренних курсоров хранится в файле pointers.txt, расположенном на сервере, в формате

Nod:ID:Ptr:Chn: [Ch1 [Ch2 [...]]]

где ID – идентификатор узла, Ptr – позиция указателя, Chn – количество дочерних узлов, Ch1 – номер дочернего узла 1, Ch2 – номер дочернего узла 2 и так далее.

Получение указателя на текущую дочернюю вершину узла node реализовано в виде функции FETCH_CHILD. Прототип функции:

function FETCH_CHILD($node)

- $node – ID целевого узла.

Перемещение внутреннего курсора на первую дочернюю вершину узла node реализовано в виде функции RESET_CHILD. Прототип функции:

function RESET_CHILD($node)

- $node – ID целевого узла.

Перемещение внутреннего курсора на следующую дочернюю вершину узла node реализовано в виде функции NEXT_CHILD. Прототип функции:

function NEXT_CHILD($node)

- $node – ID целевого узла.

Перемещение курсора на предыдущую дочернюю вершину узла node реализовано в виде функции PRIOR_CHILD. Прототип функции:

function PRIOR_CHILD($node)

- $node – ID целевого узла.

Для добавления новой дочерней вершины узла node используется функция INSERT_CHILD. Прототип функции:

function INSERT_CHILD($node)

- $node – ID целевого узла.

Удаление дочерней вершины узла node, на которую указывает курсор реализовано в виде функции DELETE_CHILD. Прототип функции:

function DELETE_CHILD($node)

- $node – ID целевого узла.

Связывание узла граф-плана node с модулем module реализовано в виде функции SET_LINK. Прототип функции:

function SET_LINK($node, $module)

- $node – ID целевого узла.
- $module – ID целевого модуля.

Получение доступа к модулю энциклопедии курса, который ассоциирован с узлом граф-плана node реализовано в виде функции GET_LINK. Прототип функции:

function GET_LINK($node)

- $node – ID целевого узла.

Вспомогательные функции

Помимо функций, выполняющих основные операции, в компонент входят некоторые вспомогательные функции.

Операция обновления ключей в затронутой добавлением узла части дерева реализована в виде функции updateParentsN. Прототип функции:

function updateParentsN($conn, $counter, $protect)

- $counter – левый ключ добавленного узла.
- $protect – ID добавленного узла.

Операция обновления ключей в затронутой удалением узла части дерева реализована в виде функции updateParentsD. Прототип функции:

function updateParentsD($conn, $counter)

- $counter – левый ключ удаленного узла.

Операция удаления узла, не имеющего детей, реализована в виде функции deleteCLNode. Прототип функции:

function deleteCLNode($conn, $ID)

- $ID – ID удаляемого узла.

Таким образом, вспомогательные функции компонента являются составными частями основных функций.

Реализация компонента «Визуализация дерева»

Операция вывода граф-плана курса на веб-страницу реализована с помощью библиотеки Vis.js. Библиотека представляет древовидные структуры с помощью класса network, конструктор которого принимает параметры container, data и options.

Параметр container – элемент HTML, который будет содержать визуализацию дерева.

Параметр data – содержит информацию об узлах и связях между ними и некоторые их свойства. Одним из свойств является shape, свойство, определяющее внешний вид конкретного узла. К вариантам внешнего вида относятся: ellipse (эллипс), circle (круг), database (база данных), box (прямоугольник) и др.

Параметр options – содержит различную информацию об оформлении. К этой информации относятся: layout (параметры размещения узлов), interaction (параметры взаимодействия с узлами) и др.

Операция выбора узла или узлов реализована с помощью событий (Event), привязанных к элементу, отображающему граф-план курса. Для этого используется функция network.on, вызывающая некоторую функцию при возникновении события. К возможным событиям относятся: selectNode (выбор узла с помощью левой кнопки мыши), deselectNode (снятие выбора с узла), oncontext (вызов контекстного меню) и др.

Листинг функций, привязанных к событиям, приведен на рисунке 19.

Данные функции, помимо сохранения номера выбранного узла, участвуют в формировании поясняющей надписи. Помимо них, надпись формируется с помощью функций, вызываемых с помощью событий onmouseover и onmouseout, привязанных к кнопкам операций. К таким функциям относится функция addHover, листинг которой приведен на рисунке 20.

Прочие привязанные к кнопкам операций функции построены по тому же принципу. Помимо формулирования операции, они выполняют также функцию блокировки операций с некорректными параметрами. В случае, если операция не может быть выполнена, область, содержащая формулировку операции, подсвечивается красным, а кнопка блокируется.

При попытке нажатия на кнопку пользователь получит сообщение о неверных параметрах для операции.

```
network.on("selectNode", function (params)
{
	document.getElementById("target1").innerHTML = params.nodes[0];
	document.getElementById("addid").value = params.nodes[0];
	document.getElementById("modid").value = params.nodes[0];
	document.getElementById("delid").value = params.nodes[0];
	document.getElementById("movid").value = params.nodes[0];
});
network.on("deselectNode", function (params)
{
	document.getElementById("target1").innerHTML = "(left click a
node)";
	document.getElementById("addid").value = "";
	document.getElementById("modid").value = "";
	document.getElementById("delid").value = "";
	document.getElementById("movid").value = "";
});
network.on("oncontext", function (params)
{
	params.event.preventDefault();
	if(tNode)
	{
		tNode.color = {background: '#97C2FC', border: '#2B7CE9'};
		nodes.update(tNode);
	}
	if(this.getNodeAt(params.pointer.DOM))
	{
		tNode = nodes.get(this.getNodeAt(params.pointer.DOM));
		tNode.color = {background: '#55ff55', border: '#119911'};
		nodes.update(tNode);
		document.getElementById("target2").innerHTML =
this.getNodeAt(params.pointer.DOM);
		document.getElementById("mtgid").value =
this.getNodeAt(params.pointer.DOM);
	}
	else
	{
		document.getElementById("target2").innerHTML = "(right click
a node)";
		document.getElementById("mtgid").value = "";
	}
});
```

Рис. 19. Листинг функций для событий

```
function addHover()
{
	if(!isNaN(parseInt(document.getElementById("cmbMod").value)))
	{
		document.getElementById("amodid").value = docu-
ment.getElementById("cmbMod").value;
	}
	if(!isNaN(parseInt(document.getElementById("amodid").value)))
	{
		document.getElementById("01").innerHTML = "Add a child node
(module: " + par-
seInt(document.getElementById("amodid").value).toString() + ") to node
";
	}
	else
	{
		document.getElementById("01").innerHTML = "Add a child node
to node ";
	}
	document.getElementById("12").innerHTML = ".";
	document.getElementById("23").innerHTML = "";
	document.getElementById("target2").style.visibility = "hidden";
	if(!isNaN(parseInt(document.getElementById("addid").value)))
	{
		document.getElementById("msgCtnr").style.backgroundColor =
"#66ff66";
	}
	else
	{
		document.getElementById("msgCtnr").style.backgroundColor =
"#ff6666";
		document.getElementById("addbtn").addEventListener("click",
blockElem);
	}
}
```

Рис. 20. Листинг функции addHover

Вспомогательные функции

Помимо перечисленных функций, выполняющих основные операции, компонент содержит некоторые вспомогательные функции.

Для блокирования кнопок при неверных параметрах используется функция blockElem. Прототип функции:

function blockElem(event)

- event – событие, вызвавшее функцию.

Для более удобного визуального представления узлов, содержащих длинную строку, используется функция downToSize. Прототип функции:

function downToSize(string)

- string – строка, которую нужно отформатировать.

Внешний вид веб-приложения приведен на рисунке 21.

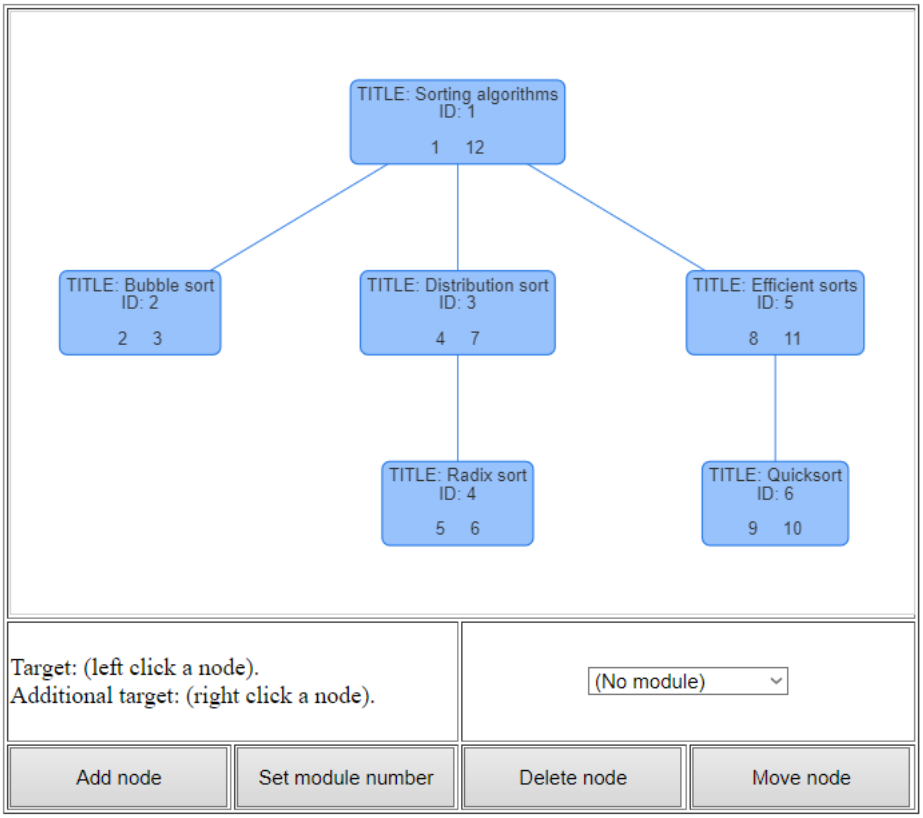


Рис. 21. Внешний вид веб-приложения

Таким образом, для реализации приложения были выбраны подходящие инструменты, после чего была выполнена реализация всех компонентов приложения в соответствии с определенными ранее функциональными и нефункциональными требованиями.

6. ТЕСТИРОВАНИЕ

Разработанный редактор курсов был протестирован на интернет-обозревателях Google Chrome версии 83.0.4103.61 и Mozilla Firefox версии 76.0.1.

Тестирование визуализации дерева

На использованных браузерах было протестировано десять различных структур курса. Все граф-планы были построены корректно.

На рисунке 22 приведена таблица gpnode, содержащая одну из использованных структур курса. На рисунке 23 также приведена таблица module, содержащая названия использованных модулей. На рисунке 24 приведен внешний вид приложения при запуске на браузере Google Chrome, а на рисунке 25 – на браузере Mozilla Firefox.

idNod	idCour	idMod	leftKey	rightKey	level
1	1	1	1	12	0
2	1	2	2	3	1
3	1	3	10	11	1
4	1	4	5	6	2
5	1	5	4	9	1
6	1	6	7	8	2

Рис. 22. Содержимое таблицы gpnode

idMod	idEnc	title	date	annotation
1	*NULL*	Sorting algorithms	*NULL*	*NULL*
2	*NULL*	Bubble sort	*NULL*	*NULL*
3	*NULL*	Distribution sort	*NULL*	*NULL*
4	*NULL*	Radix sort	*NULL*	*NULL*
5	*NULL*	Efficient sorts	*NULL*	*NULL*
6	*NULL*	Quicksort	*NULL*	*NULL*

Рис. 23. Содержимое таблицы module

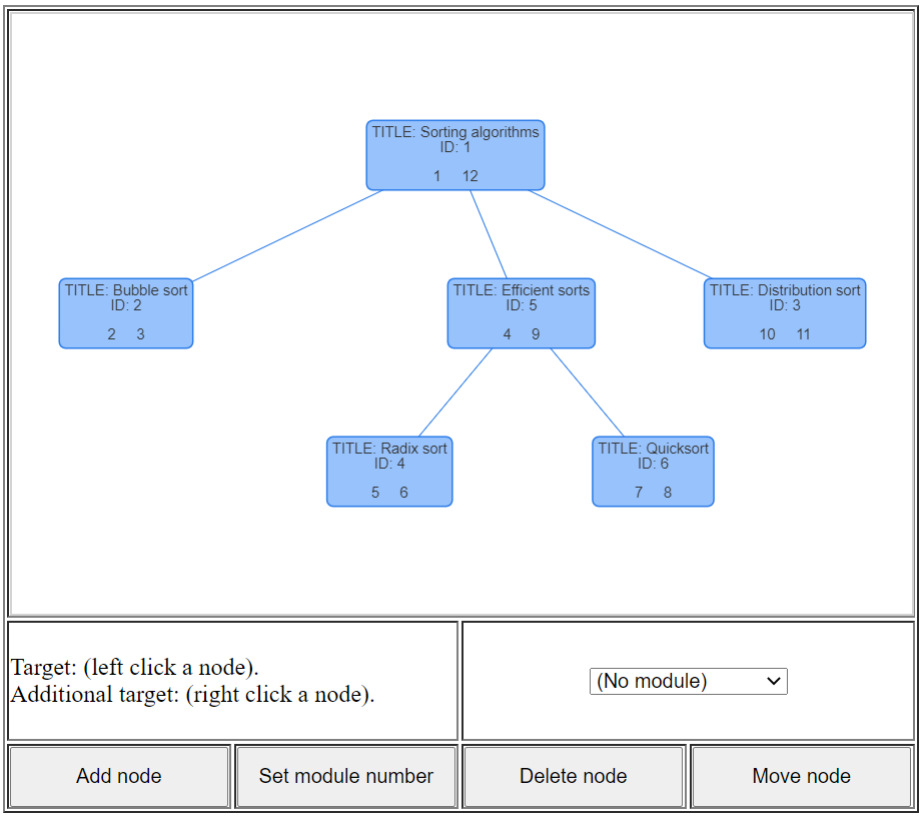


Рис. 24. Визуализация на браузере Google Chrome

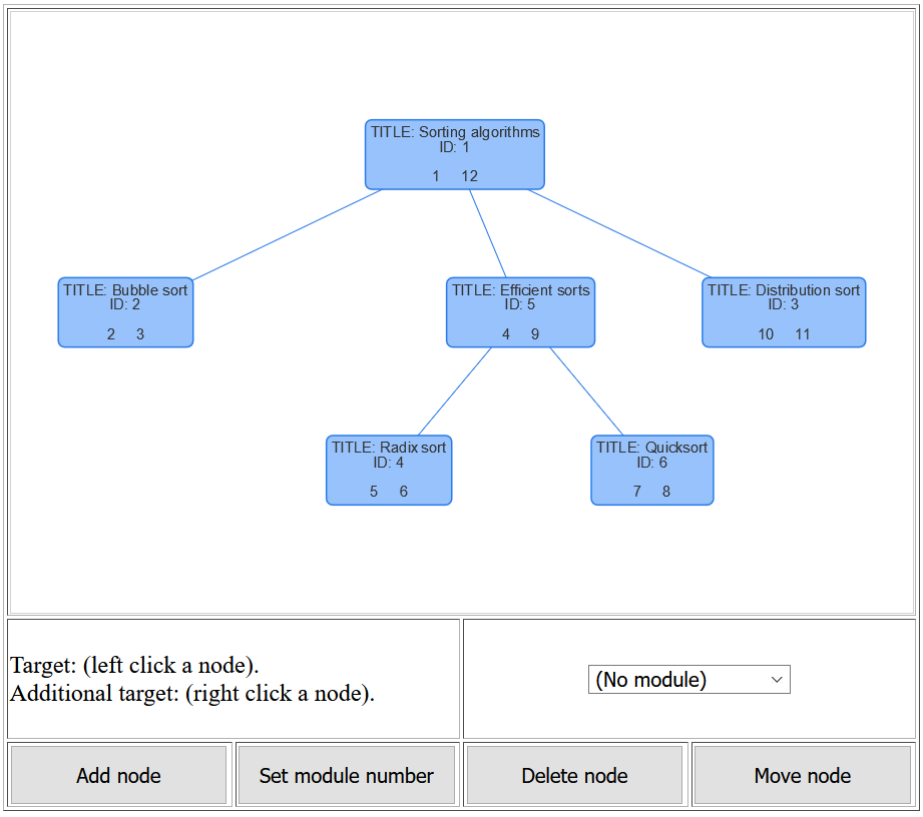


Рис. 25. Визуализация на браузере Mozilla Firefox

Помимо информации о структуре дерева, приложение также получает список существующих учебных модулей, которые отображает в виде выпадающего списка. Содержимое этого списка в браузере Mozilla Firefox приведено на рисунке 26.

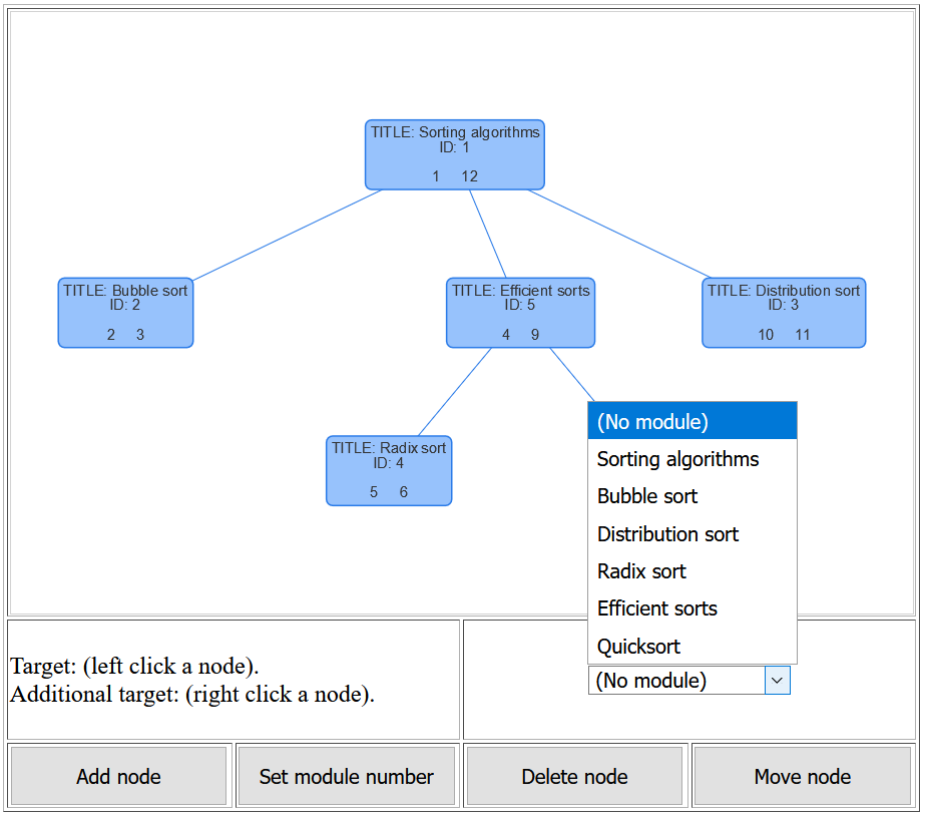


Рис. 26. Список учебных модулей

Тестирование добавления узлов

На каждом используемом браузере в структуру курса были добавлены десять узлов. Все узлы были добавлены корректно.

На рисунке 27 приведен внешний вид приложения перед добавлением дочернего узла к узлу с идентификатором 6, а на рисунке 28 – визуализация, построенная приложением после выполнения операции. В данном тесте использовался браузер Mozilla Firefox.

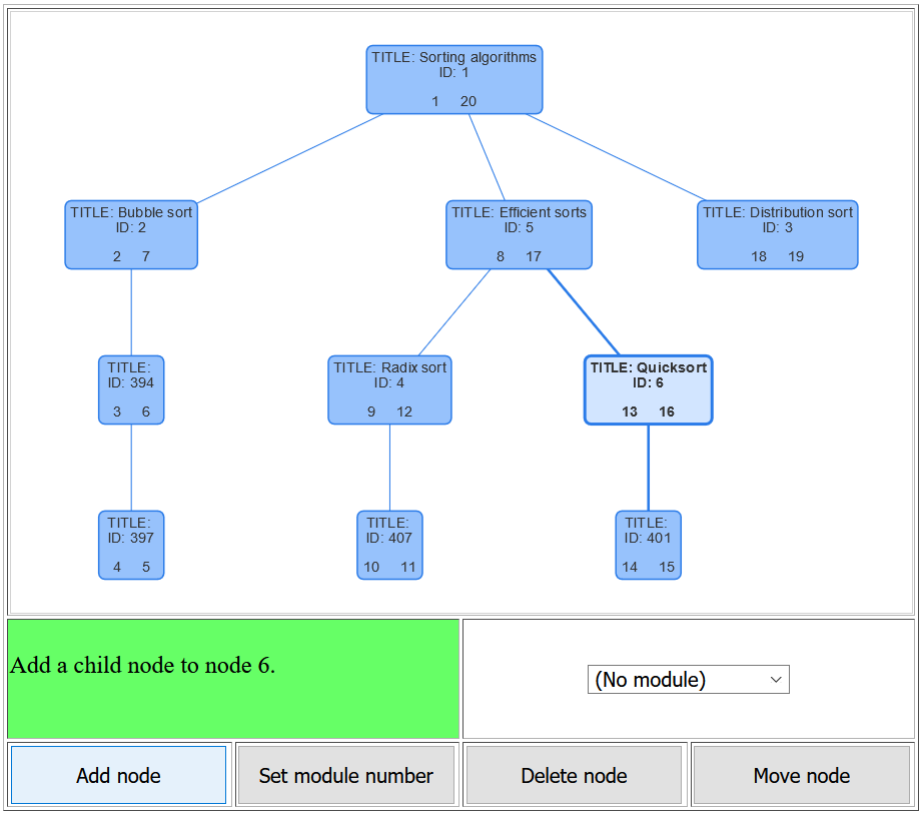


Рис. 27. Вид приложения перед добавлением дочернего узла

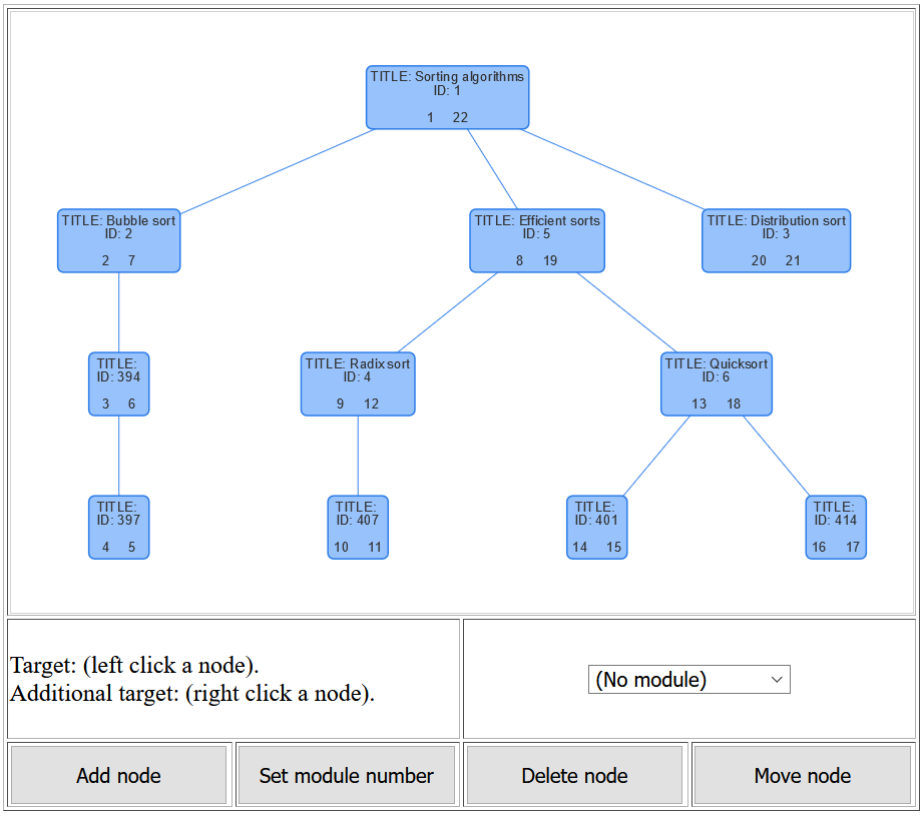


Рис. 28. Вид приложения после добавления дочернего узла

Тестирование переопределения модуля

На каждом из используемых браузеров были двадцать раз переопределены модули узлов. В ходе тестов не было выявлено ошибок.

На рисунке 29 приведен внешний вид приложения перед изменением модуля, ассоциированного с узлом с идентификатором 6. Модуль узла был изменен с модуля под названием «Quicksort» на модуль под названием «Bubble sort». На рисунке 30 приведен внений вид приложения после выполнения операции. В данном тесте использовался браузер Google Chrome.

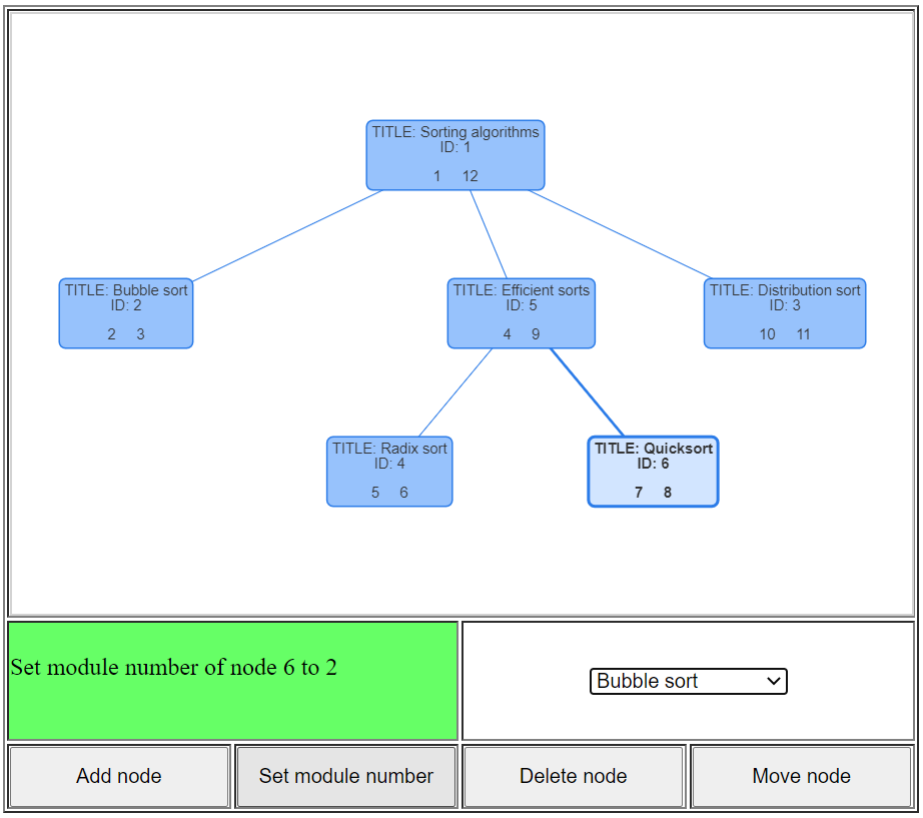


Рис. 29. Вид приложения перед переопределением модуля

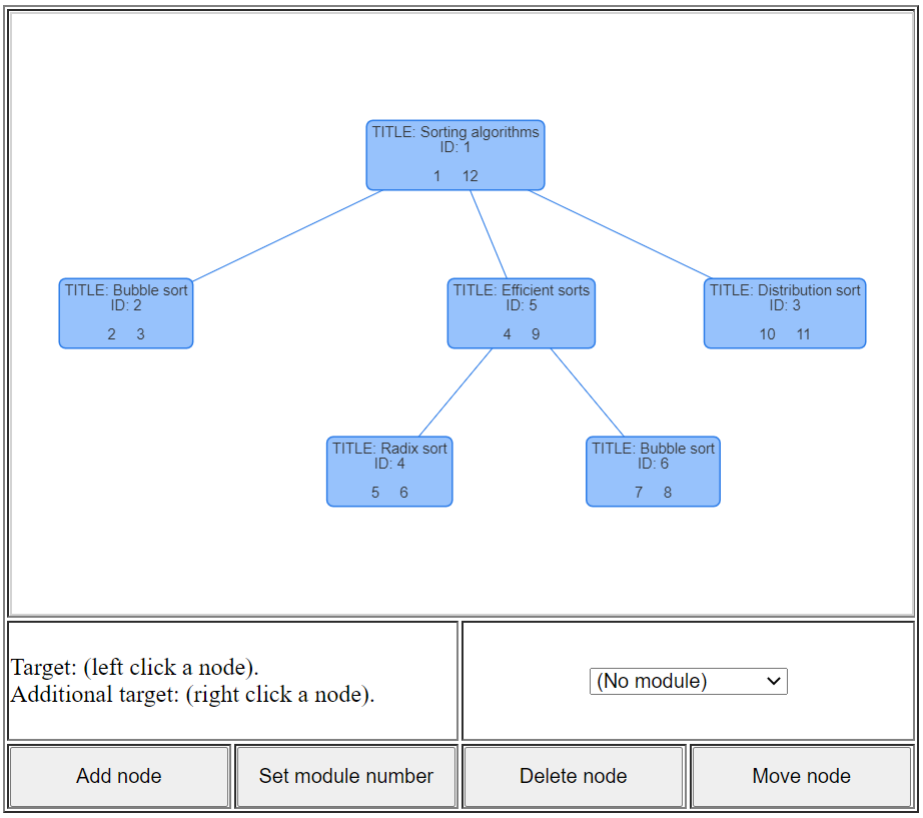


Рис. 30. Вид приложения после переопределения модуля

Тестирование удаления узла

На каждом из используемых браузеров было десять раз выполнено удаление узла. Все операции были проведены корректно.

На рисунке 31 приведен внешний вид приложения перед удалением узла с идентификатором 420. Удаляемый узел имеет два дочерних узла с идентификаторами 421 и 422, которые были удалены вместе с ним. На рисунке 32 приведен внешний вид приложения после выполнения операции удаления. В данном тесте использовался браузер Google Chrome.

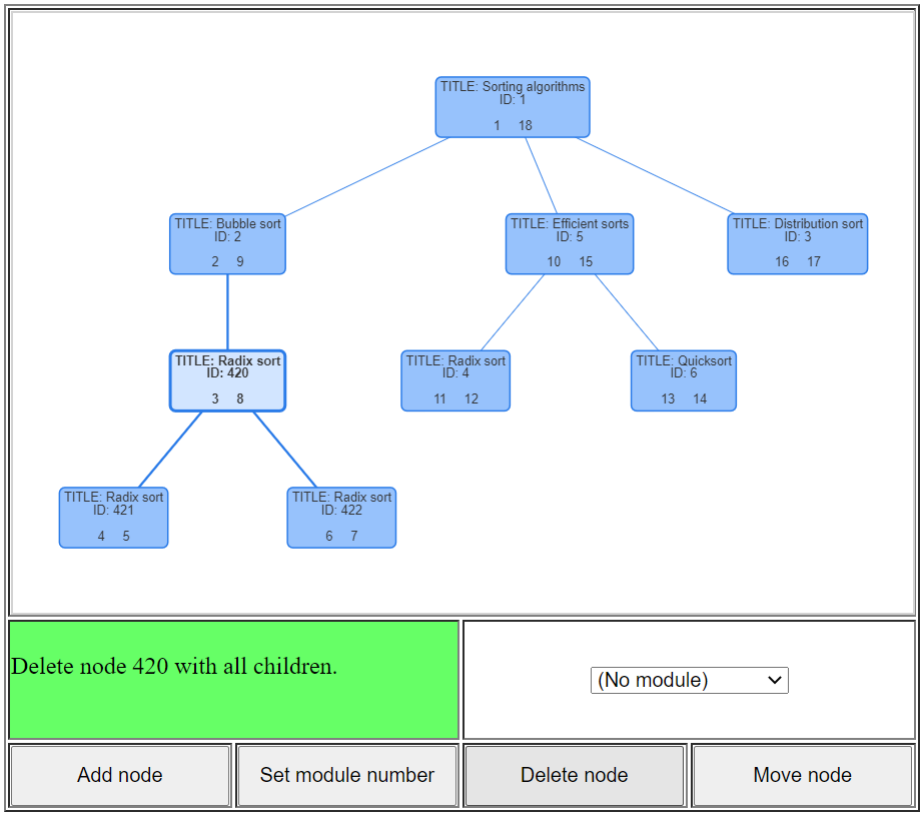


Рис. 31. Вид приложения перед удалением узла

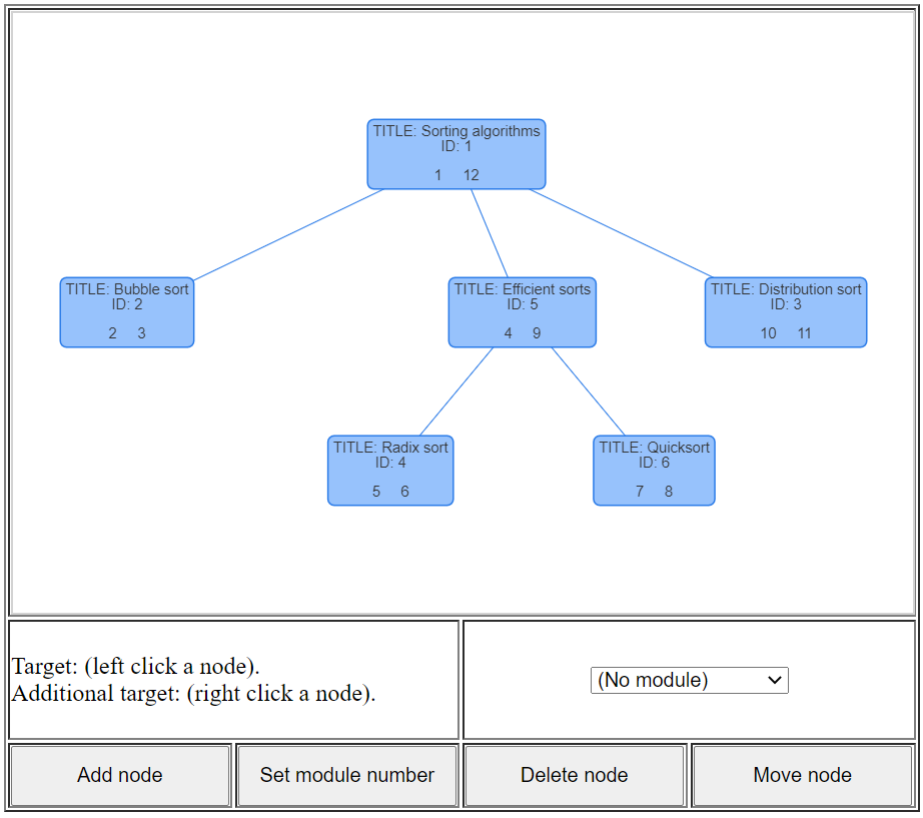


Рис. 32. Вид приложения после удаления узла

Тестирование перемещения узла

На каждом из используемых браузеров было десять раз выполнено перемещение узла. В ходе выполнения тестов не было обнаружено ошибок.

На рисунке 33 приведено окно приложения перед перемещением узла с идентификатором 439 из позиции дочернего по отношению к узлу с идентификатором 4 в позицию дочернего по отношению к узлу с идентификатором 6. Вместе с узлом 439 были перемещены его дочерние узлы с идентификаторами 442 и 445. На рисунке 34 приведено окно приложения после выполнения операции по перемещению узла. В данном тесте использовался браузер Google Chrome.

Таким образом, основные функции приложения были протестированы на современных версиях наиболее популярных браузеров. В ходе тестирования не было выявлено ошибок в работе программы. В двух протестированных браузерах есть незначительные различия в отображении интерфейса, в том числе несколько отличающийся цвет границ редактора и кнопок, а также различные шрифты. Результаты тестирования позволяют сделать вывод, что приложение способно корректно выполнять свои основные функции на браузерах Google Chrome и Mozilla Firefox.

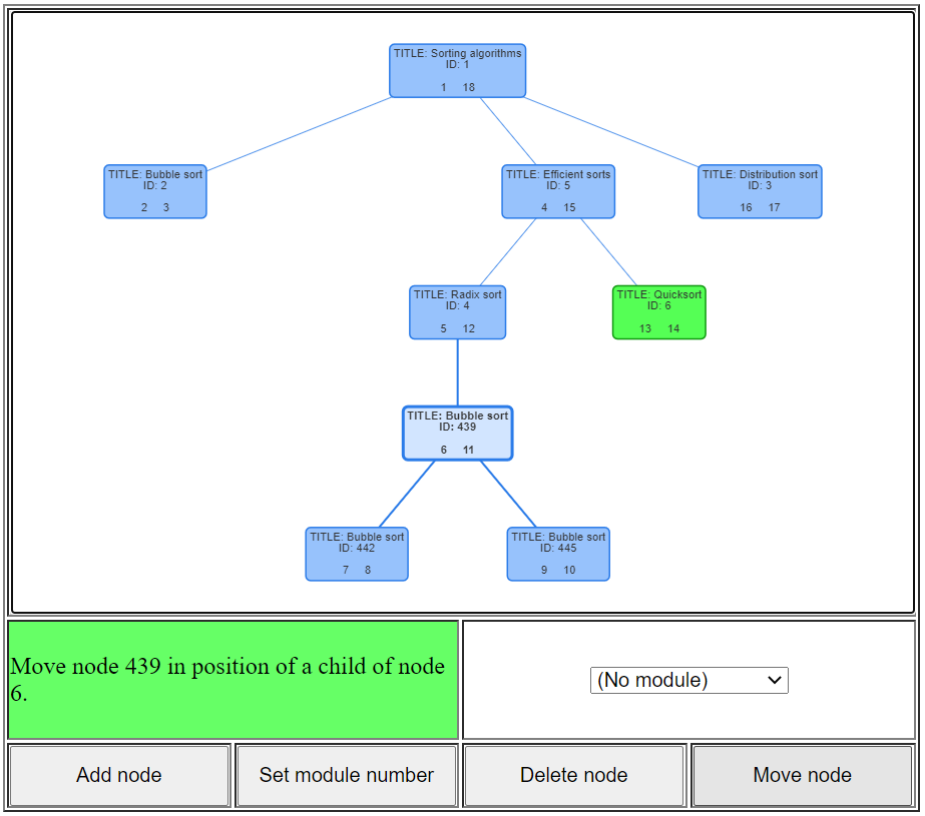


Рис. 33. Вид приложения перед перемещением узла

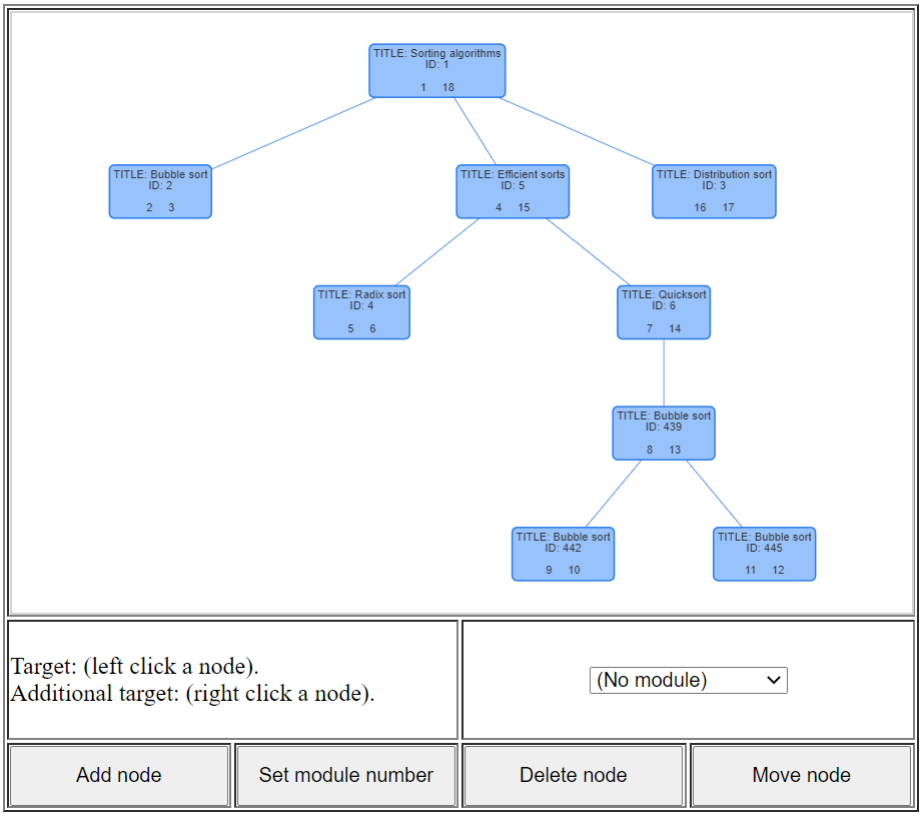


Рис. 34. Вид приложения после перемещения узла

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была изучена структурно-иерархическая дидактическая модель электронного обучения. Далее был проведен обзор моделей представления деревьев в реляционных СУБД, а также обзор средств визуализации древовидных структур в рамках веб-приложения. После выбора наиболее подходящих для поставленной задачи модели представления и средства визуализации было выполнено проектирование редактора курсов для программной системы ECoD и его реализация. Редактор был протестирован и передан заказчику.

По результатам работы была написана статья «Редактор электронного учебного курса для структурно-иерархической дидактической модели» [6], принятая к публикации в журнале «Наука ЮУрГУ 2019», а также статья «Application of a Hierarchical Approach for Determining an Individual Study Trajectory» [12], принятая к публикации на международной научной конференции «FarEastCon – 2019».

В настоящий момент производится интеграция разработанного веб-приложения в систему ECoD. В будущем планируется продолжать разработку и улучшение веб-приложения, в частности реализовать поиск учебных модулей для сопоставления с узлами граф-плана, работу с образовательными компетенциями и др.

ЛИТЕРАТУРА

1. Богданов Д. В. Оптимальный способ хранения и обработки древовидных структур в базах данных // Программные продукты и системы, 2009. № 1. С.140-142.

2. ГОСТ 33245-2015 (ISO/IEC TR 29163-1:2009) Информационные технологии (ИТ). Эталонная модель распределенного объекта контента (SCORM®) 2004 3-я редакция. Часть 1. Обзор. Версия 1.1. URL: http://docs.cntd.ru/document/1200127254 (дата обращения: 10.05.2020 г.).

3. Жигальская Н.С. Моделирование дидактической структуры электронных учебных комплексов // Вестник Южно-Уральского государственного университета. Серия «Математическое моделирование и программирование». 2008. № 27(127). Вып. 2. С. 4-9.

4. Жигальская Н.С. Модель вариантов использования универсальной среды электронного обучения UniCST // Инновационные технологии обучения: проблемы и перспективы: Материалы всерос. науч.-метод. конф. (29-30 марта 2008 г., Липецк) -Липецк: Изд-во ЛГПУ, 2008. С. 204-207.

5. Маликов А. В. Ориентированные графы в реляционных базах данных // Доклады ТУСУР, 2008. № 2(18). Ч. 2. С. 100-104.

6. Силкина, Н.С., Глизница М.Н. Редактор электронного учебного курса для структурно-иерархической дидактической модели // Материалы 71-й научной конференции "Наука ЮУрГУ". Челябинск: Издательский центр ЮУрГУ, 2019. С. 343-352.

7. Силкина Н.С., Соколинский Л.Б. Система UniCST – универсальная среда электронного обучения // Системы управления и информационные технологии. 2010. № 2. С. 81-86.

8. Силкина Н.С., Соколинский Л.Б. Структурно-иерархическая дидактическая модель электронного обучения // Вестник Южно-Уральского государственного университета. Серия «Вычислительная математика и информатика». 2019. Том 8, № 4.

9. Токмянин В.В. Дистанционное образование: зависимость качества от формы обучения // Теория и практика образования в современном мире: материалы междунар. науч. конф. (г. Санкт-Петербург, февраль 2015 г.). – СПб.: Реноме, 2015. – С. 370-372.

10. Desktop Browser Market Share Worldwide | StatCounter Global Stats URL: http://gs.statcounter.com/browser-market-share/desktop/worldwide (дата обращения: 24.05.2020 г).

11. Dracula Graph Library. URL: https://www.graphdracula.net (дата обращения: 16.03.2020 г.).

12. Gliznitsa, M. Application of a Hierarchical Approach for Determining an Individual Study Trajectory / M.. Gliznitsa, N.. Silkina //2019 International Multi-Conference on Industrial Engineering and Modern Technologies, FarEastCon 2019.

13. Ivanova, O.N., Silkina N.S. Competence-Oriented Model of Representation of Educational Content // Proceedings of the 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO'2017, Opatija, Croatia, May 22 26, 2017. IEEE, 2017. P. 791–794.

14. Joe Celko. Trees in SQL. Some answers to some common questions about SQL trees and hierarchies, 2004. URL: http://www.ibase.ru/files/articles/programming/dbmstrees/sqltrees.html (дата обращения: 02.06.2020 г).

15. SCORM 2004 4th Edition. Content Aggregation Model (CAM). Advanced Distributed Learning (ADL) Initiative. 2009.

16. SCORM 2004 4th Edition. Run-Time Environment (RTE). Advanced Distributed Learning (ADL) Initiative. 2009.

17. SCORM 2004 4th Edition. SCORM Users Guide for Programmers. Advanced Distributed Learning (ADL) Initiative. 2011.

18. SCORM 2004 4th Edition. Sequencing and Navigation (SN).Advanced Distributed Learning (ADL) Initiative. 2009.

19. Sigma js. URL: http://sigmajs.org (дата обращения: 16.03.2020 г.).

20. Treant.js – javascript library for drawing tree diagrams. URL: https://fperucic.github.io/treant-js/ (дата обращения: 10.01.2020 г.).

21. Vis.js. URL: https://visjs.org (дата обращения: 10.01.2020 г.).

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А. СПЕЦИФИКАЦИЯ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Спецификация вариантов использования (ВИ) системы приведена в таблицах А.1 – А.6.

Табл. А.1. Спецификация ВИ «Смотреть граф-план»

Прецедент: Смотреть граф-план
ID: 1
Краткое описание: Визуализация граф-плана курса.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: отсутствуют
Основной поток: 1. Прецедент начинается при запуске приложения. 2. Приложение формирует визуализацию граф-плана на основе данных, полученных из базы данных.
Постусловия: 1. Визуализация граф-плана выводится в рабочую область программы.
Альтернативные потоки: I. Ошибка при обращении к базе данных. 1. Вывести сообщение об ошибке.

Табл. А.2. Спецификация ВИ «Добавить узел»

Прецедент: Добавить узел
ID: 2
Краткое описание: Добавление нового узла курса.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Выбран целевой узел граф-плана
Основной поток: 1. Прецедент начинается при нажатии кнопки «Добавить дочерний узел». 2. Приложение изменяет структуру курса, добавляя к выбранному узлу дочерний узел. 3. Если в меню приложения был выбран модуль, то ссылка на этот модуль записывается в добавленный узел.
Постусловия: 1. В структуру курса в базе данных был добавлен новый узел.
Альтернативные потоки: I. Ошибка при обращении к базе данных. 1. Вывести сообщение об ошибке. II. Не выбрана цель для действия. 1. Вывести сообщение об ошибке.

Табл. А.3. Спецификация ВИ «Переопределить модуль»

Прецедент: Переопределить модуль
ID: 3
Краткое описание: Изменение ссылки на учебный модуль в узле курса.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: 1. Выбраны узел и номер модуля
Основной поток: 1. Прецедент начинается при нажатии кнопки «Изменить номер модуля». 2. Приложение вставляет ссылку на модуль, выбранный в меню приложения в узел, выбранный на граф-плане.
Постусловия: 1. Ссылка на модуль в выбранном узле указывает на выбранный модуль.
Альтернативные потоки: I. Ошибка при обращении к базе данных. 1. Вывести сообщение об ошибке. II. Не выбрана цель для действия. 1. Вывести сообщение об ошибке.

Табл. А.4. Спецификация ВИ «Удалить узел»

Прецедент: Удалить узел
ID: 4
Краткое описание: Удаление узла курса.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: 1. Выбран целевой узел граф-плана
Основной поток: 1. Прецедент начинается при нажатии кнопки «Удалить узел». 2. Приложение удаляет из структуры курса выбранный узел и все дочерние по отношению к нему узлы.
Постусловия: 1. Выбранный узел и все его дочерние узлы удалены из структуры курса в базе данных.
Альтернативные потоки: I. Ошибка при обращении к базе данных. 1. Вывести сообщение об ошибке. II. Не выбрана цель для действия. 1. Вывести сообщение об ошибке.

Табл. А.5. Спецификация ВИ «Переместить узел»

Прецедент: Переместить узел
ID: 5
Краткое описание: Перемещение узла курса.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: 1. Выбраны целевые узлы граф-плана
Основной поток: 1. Прецедент начинается при нажатии кнопки «Переместить узел». 2. Приложение перемещает выбранный узел из текущей позиции в структуре курса в позицию, дочернюю по отношению к узлу, выбранному как дополнительная цель.
Постусловия: 1. Выбранный узел является дочерним для узла, выбранного как дополнительная цель.
Альтернативные потоки: I. Ошибка при обращении к базе данных. 1. Вывести сообщение об ошибке. II. Не выбрана цель для действия. 1. Вывести сообщение об ошибке.

Табл. А.6. Спецификация ВИ «Выбрать модуль»

Прецедент: Выбрать модуль
ID: 6
Краткое описание: Выбор учебного модуля.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: 1. Использование панели «Выбранный модуль».
Основной поток: 1. Прецедент начинается при использовании панели «Выбранный модуль». 2. Пользователь выбирает из списка один из имеющихся учебных модулей.
Постусловия: 1. Выбран учебный модуль, который может использоваться в других операциях.
Альтернативные потоки: Нет