Учреждение образования «Витебский государственный технологический

университет»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование вуза)

|  |  |  |
| --- | --- | --- |
| Факультет ФПКиПК |  | Кафедра "Информационные системы и автоматизация производства" |
| «Утверждаю»:  Зав.кафедрой ***\_\_\_\_\_\_\_\_\_\_\_\_\_\_***  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. |  |  |

**З А Д А Н И Е**

**по дипломному проектированию**

Студенту  *Атрощенко Константину Анатольевичу*

1.Тема проекта РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ИНТЕРНЕТ МАГАЗИНА

Утверждена приказом по вузу от *26.03.2020 № 176-л*

2.Сроки сдачи студентом законченного проекта *2020г*

3.Исходные данные к проекту *методические указания, техническая и научная литература отечественных и зарубежный авторов.*

4.Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

|  |
| --- |
| *Введение* |
| 1. Анализ объекта   * 1. Описание предметной области   2. Построение концептуальной модели системы интернет-магазина   2. Постановка задачи  2.1 Определение требований в информационной системе интернет-магазина   * 1. Обзор и обоснование выбора инструментальных средств  1. Проектирование    1. Разработка архитектуры программного продукта    2. Проектирование структур хранения данных    3. Описание реализации вариантов использования   4. Реализация   * 1. Разработка классов информационной системы   2. Разработка интерфейса программного продукта   3. Разработка алгоритмов реализации вариантов использования  1. Интеграционное тестирование 2. Экономическое обоснование эффективности внедрения 3. Охрана труда 4. Промышленная экология 5. Ресурсосбережение   *Заключение*  ПРИЛОЖЕНИЯ  ПРИЛОЖЕНИЕ А  ПРИЛОЖЕНИЕ Б  ПРИЛОЖЕНИЕ С |

5.Перечень графического материала (с точным указанием чертежей графиков)

*Мультимедийная презентация*

6.Консультанты по проекту (с указанием относящихся к ним разделов проекта) *доцент В.Е. Казаков\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания 26.03.2020

8.Календарный график работы над проектом на весь период проектирования (с указанием сроков выполнения и трудоемкости отдельных этапов)

*Разработка файлов данных и дизайна 01.04.2020- 17.04.2020*

*Введение, теоретическая часть 18.04.2020 - 12.05.2020*

*Практическая часть, написание кода программы 13.05.2020 – 27.07.2020*

*Оформление дипломной работы 28.07.2020 – 05.08.2020*

*Представление на подпись 06.08.2020 – 21.08.2020*

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Е. Казаков

Задание принял к исполнению (дата 26.03.2020).

(подпись студента) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РЕФЕРАТ**

Дипломная работа 181 с., 61 рис., 1таблица. 21 источников, 3 приложения

.

**Слайд 1**

Уважаемы председатель, уважаемые члены Государственной аттестационной комиссии, вашему вниманию предоставляется дипломная работа на тему «ИНФОРМАЦИОННАЯ СИСТЕМА ИНТЕРНЕТ-МАГАЗИНА». Актуальность темы обусловлена необходимостью разработки программного продукта, который позволяет миллионам людей, не выходя из дома, совершают покупки в интернет-магазинах. 2020 год очень негативно сказывается на субъектах предпринимательской и государственной деятельности из – за COVID-19 тем самым повышая актуальность данной темы.

Объектом научного исследования в данной работе является интернет-магазин курсов.

В рамках работы были сформулированы следующие задачи:

1. провести анализ средств разработки и выбрать инструментальную систему для разработки интернет-магазина.
2. Разработать документно-оборотную базу данных MongoDB c помощью framework express.
3. Разработать классы информационной системы и представить их в виде структурной схемы UML, продемонстрировать при этом все сущности предметной области, атрибуты, функции и связи.
4. Реализовать вэб-интерфейс интернет-магазина.

В первой главе происходит описание предметной области, в которой рассматриваются такие платформы как:

1. CRM (Customer Ralationship Management – управление взаимоотношениями с клиентами).
2. 1С – программный продукт предназначенный для автоматизации деятельности на предпирятии.
3. ERP (Enterprise Resource Planning - управление ресурсами предприятия)
4. CRM, CMS (Content Management System – система управления контентом сайта).

Так же рассматривается «Построение концептуальной модели информационной системы интернет магазина» которая представлена на слайде 2.

**Слайд 2**

Данная модель отражает связь параметров. Для полноты восприятия поставленной задачи, в приложении А.1, рассмотрены три ключевые сущности, а именно: курсы, корзина и заказы.

Курсы – это страница на которой содержится множество карточек курсов или же по-другому карточек товаров. Данные карточки курсов состоят:

1. Заглавие т.е. название курса.
2. Цена курса.
3. Картинка или логотип курса.
4. Описание курса.

Поля «id Пользователя» и «id Курса» от пользователя скрыты. С помощью данных полей происходит вся логика т.е. добавление, редактирование и удаление курса. А также происходит отображение по правам доступа т.е. кнопка «Редактировать» появляется только у тех пользователей, которые являются владельцами.

Так же имеются кнопки:

1. Открыть.
2. Редактировать.
3. Купить.

При нажатии на кнопку «Открыть» происходит открытие дельного описания курса. При нажатии на кнопку редактировать происходит перенаправление на страницу «Редактировать». И кнопки «Купить» при нажатии на которую происходить добавление данного курса в корзину.

При добавлении одного или нескольких курсов в корзину происходит формирование таблицы.

Таблица состоит из трех колонок:

1. Название.
2. Количество.
3. Действие.

В первую колонку записывается название курса. Во второй, с помощью счетчика формируется количество добавленного курса. А в третьем происходит действие т.е. удаление при котором происходит срабатывание счетчика и уменьшения количества заказываемого курса или же вообще его удаление.

Так же имеется счетчик цены, который показывает пользователю общую стоимость покупки, которая в свою очередь формируется в зависимости от действий пользователя т.е. добавления либо удаления.

При нажатии на кнопку «Сделать заказ» происходит формирование заказа и перенаправление на страницу «Заказы».

Страница «Заказы» состоит из карты заказа в которую входит:

1. Id заказа.
2. Имя пользователя.
3. Email.
4. Дата заказа.
5. Список курсов и их количество.
6. Цена заказа.

Данная концептуальная модель отображает главные функции приложения, которые необходимо реализовать.

**Проведение анализа средств разработки рассмотрено в главе «Обзор и обоснование выбора инструментальных средств».**

**Слайд 3**

Для разработки «информационной системы интернет-магазина» были использованы следующие инструментальные средства:

1. Редактор кода **Visual Studio Code** (<https://code.visualstudio.com/>)
2. Платформа **NodeJS** – это программная платформа, основанная на движке v8 (транслирующем JavaScript в машинный код) превращая JavaScript из узкоспециализированного языка в язык общего назначения. Поверх него написана оболочка на C++ которая позволяет NodeJS запускаться где угодно.
3. JavaScript. Одна из главных причин популярности Node.js такова: для нее программируют на знакомом большинству программистов языке – JavaScript. Раньше, как минимум нужно было в совершенстве знать как минимум два языка программирования: JavaScript для клиентской части и что – то еще, например PHP или Ruby – для серверной.
4. **Framework Express**(<https://expressjs.com/>) Express самый популярный фреймворк для Node.js.
5. **MongoDB** (<https://account.mongodb.com/>) она подобна Node.js славится своей быстротой работы и масштабируемостью.
6. **Mongoose**([**https://mongoosejs.com/**](https://mongoosejs.com/)) позволяет определить структуру данных в самом приложении, а не в БД. Таким способом Mongoose обеспечивает изящное моделирование объектов MongoDB для NodeJS. Mongoose облегчает управление подключениями к базе данных MongoDB, а так же сохранение и чтение данных. Благодаря Mongoose приобретается возможность добавления проверки данных на уровне схемы, гарантируя, что в БД можно сохранять только корректные данные.
7. Git (<https://github.com/>) – Система контроля версий и управление исходным кодом.
8. **Heroku**(<https://www.heroku.com/>) - один из ведущих провайдеров хостинга для приложений Node.js, и у него имеется великолепный бесплатный пакет, которым я и воспользуюсь.
9. **SendGrid**(<https://app.sendgrid.com/>) В PHP есть функция mail которая позволяет с помощью сервера отправлять email на любой почтовый адрес. Но в Node.js это не так. В Node.js для того что бы отправлять какие либо email нужно пользоваться сторонними сервисами.
10. **Webpack**(<https://webpack.js.org/>) – это статический модульный сборщик.
11. **Babel** (<https://babeljs.io/>) - это JavaScript компилятор, но в отличии от классических компиляторов которые переводят код в машинный код, babel переводит EcmaScript2015+ (самая свежая спецификация) в более старую спецификацию. Однако на этом он не останавливается. В своем проекте я использую препроцессор Sass. И аналогично тому как Babel компилирует современные спецификации JavaScript в более старые, он компилирует исходный код Sass в обычную каскадную таблицу стилей css.
12. **Handlebars** (<https://handlebarsjs.com/>) – это шаблонный процессор, который динамически генерирует HTML страницу. Handlebars генерирует HTML используя структуру JSON и запускает ее через шаблон. Конечно, вызовы AJAX могут возвращать фрагменты HTML, которые можно вставить в DOM как есть, но Handlebars на стороне клиента дает возможность получать результаты вызова AJAX в виде данных в формате JSON и форматировать их для нужд приложения. Шаблоны написаны в обычном HTML и набиты заполнителями. Заполнители позволяют при необходимости вводить данные.
13. **SCSS**(<https://sass-lang.com/>) – является языком надстройкой над языком разметки css т.е. это лишь компилированный язык для каскадной таблицы стилей.

**Проектирование БД MongoDB**

Mongoose был специально создан в качестве объектно-документного средства моделирования (Object-Document Modeler (ODM)) MongoDB для приложений NodeJS. Один из ключевых принципов – возможность управления моделью данных изнутри приложения. Не нужно возится непосредственно с базами данных, внешними фреймворками или реляционными подпрограммами отображения – можно просто описать модель данных, находясь в комфортных условиях своего приложения.

1. В MongoDB каждая запись в БД называется документом.
2. В MongoDB набор документов называется коллекцией.
3. В Mongoose описание документа называется схемой.
4. Каждая отдельная сущность данных в схеме называется путем.

**Слайд 4**

В MongoDB структурной единицей является BSON – документ.

Структурной единицей которого в MongoDB является хранилище ключей и значений. В хранилище ключей и значений, имеются ключи, указывающие на аморфные значения.

Определившись со структурным делением данных, нужно понять, как ими манипулировать. В РСУБД предлагаются произвольные запросы и операции соединения. В MongoDB произвольные запросы так же поддерживаются, но соединение не поддерживается.

**Слайд 5**

Рассмотрим на примере модели courses в MongoDB как может выглядеть схема Mongoose. Как видно, схема очень напоминает, сами данные. Она описывает имя для каждого пути данных и содержащихся в нем тип данных. В данном примере я просто объявляю title и img как строковое значение, price числовое, description объявляю объектом, userId обращаюсь к схеме пользователя в которой указан уникальный id пользователя.

В схеме Mongoose я не объявляю путь \_id. \_id – это уникальный идентификатор, первичный ключ для каждого документа. MongoDB автоматически генерирует этот путь при создании каждого документа и присваивает ему уникальное значение objectId. Базовая структура описания пути представляет собой название пути со следующим за ним объектом свойств. И так путь схемы состоит из двух частей – названия пути и объекта свойств. Модель в Mongoose – это скомпилированная версия схемы. Однократно скомпилированный, отдельный экземпляр модели отображается непосредственно на отдельный документ в БД. Именно благодаря этому взаимному однозначному соответствию модель может создавать, читать, сохранять и удалять данные.

**Слайд 6**

**Разработка классов информационной системы.**

**Слайд 7 добавление курса**

Рассмотрим информационную систему интернет магазина согласно разработанной архитектуре MVC.

Страница «Добавить курс» состоит из формы, в которую входят следующие поля: title, price, img, description. При заполнении данных полей происходит валидация формы. Если формы заполняются не должным образом, то пользователь получает сообщение с ошибкой.

При заполнении и отправке форм происходит сверка полей в модели курса, в которой указано обязательны ли данные поля к заполнению. Так же происходит сверка id пользователя который создает курс.

При удачном сценарии происходит добавление курса в базу данных с последующим перенаправлением пользователя на страницу «Курсы».

**Слайд 8 Редактирование курса**

Пользователь посылает get запрос, происходит аутентификация пользователя. Далее происходит проверка query параметра, который отвечает за доступ пользователя к редактированию курса. И если по данному запросу не будет найдено заданное поле allow, то будет происходить перенаправление на страницу «Главная».

Далее производится поиск id курса после чего происходит перенаправление на страницу «Редактировать курс». Если же нет, то происходит перенаправление на страницу «Курсы».

При перенаправлении пользователя происходит аутентификация пользователя. И если пользователь не зарегистрирован в системе будет происходить перенаправление на страницу «Регистрации и авторизации». И лишь при успешной аутентификации будет происходить перенаправление ни страницу «Редактировать курс».

Страница «Редактировать курс» состоит из формы, в которую входят следующие поля: title, price, img, description. При заполнении данных полей происходит валидация формы. Если формы заполняются не должным образом, то пользователь получает сообщение с ошибкой. При успешном прохождении валидации происходит замена старых значений полей на новые, после чего происходит перенаправление на страницу «Курсы».

**Слайд 9 Удаление курса**

Пользователь посылает get запрос, происходит аутентификация пользователя. Далее через модель курса происходит сверка \_id курса и id пользователя, после чего происходит удаление курса из БД с последующим перенаправлением на страницу «Курсы».

**Слайд 10 Добавление курса в корзину**

В представлении с помощью кнопки «Купить» предаю id того курса который пользователь хочет приобрести. В контроллере card.js импортирую класс Router из библиотеки express и сразу для удобства работы заношу данный класс в переменную router.

Далее обращаюсь к переменной router, вызываю метод post, где первым параметром указываю маршрут ‘/add’ а вторым параметром указываю callback функцию.

После чего в точке входа в приложение, entry point, в файле index.js я импортирую и регистрирую данный маршрут.

В теле функции создаю переменную course т.к. действие асинхронное я обращаюсь к оператору await, к модели Course вызываю метод getById в параметрах которого прописываю искомый req.body.id. Так как корзина привязана к каждому пользователю, т.е. для каждого пользователя она уникальна я получаю пользователя, с помощью объекта req.user.

и теперь смотрим способ, который предоставляет mongoose для того, чтобы более удобно выстраивать подобные логические цепочки.

Перехожу в models user.js. У меня есть userSchema которую я могу расширить. В конце документа, перед экспортом модуля обращаюсь к userSchema. к его объекту methods. далее здесь я могу определить другой любой метод который вынесет логику прямо в объект пользователя. И называю этот метод addToCart = addToCard это метод который нужно определить как функцию, но здесь есть один нюанс. Здесь обязательно использовать ключевое слово function(){} т.е. конечно я могу задать функцию через стрелочную функцию. Но здесь важно использовать ключевое слово function потому, что, я буду здесь работать с ключевым словом this.  
Когда я создаю функцию через стрелочную функцию т.е. без ключевого слова function, контекст this я не могу использовать. Я не смогу с ним работать. Поэтому это критично важно.

И внутри этой функции я буду прописывать логику, которая позволит добавлять элемент в корзину.  
В параметр function я принимаю объект (course).

Данный метод req.user.addToCart(course); является асинхронным, т.е. он будет что то сохранять в БД. Поэтому к нему также применяю оператор await. Для того что бы NodeJS подождал пока выполнится данный req.user.addToCart(course) запрос.

И вот теперь необходимо реализовать логику, которая позволит добавить этот курс в корзину. И в случае, если в корзине лежит такой курс, то тогда нужно просто увеличить количество.

Перехожу в models user.js и продолжаю писать.

Для начало, мне потребуется, обратится к корзине cart и забрать у нее массив items. Именно в данном массиве лежат те элементы у которых есть свойство count и courseId.

Создаю массив const items и просто клонирую массив items т.е. обращаюсь к this.cart.items создаю новый массив this.cart.items. Для этого я использую Spreed оператор.

Для логичности переименовываю переменную на clonedItems которая принимает новый клонированный массив.

Дальше нужно найти в данном массиве clonedItems тот курс с которым я работаю, а точнее найти его индекс const idx =. Далее я обращаюсь к clonedItems. и у этого массива я могу вызвать метод findIndex() для того что бы найти индекс того или иного элемента, я прописываю условие где на каждой итерации я получаю объект findIndex(c =>{}).

Далее в callback мне необходимо вернуть условие. У каждого элемента есть поле coursed, поэтому обращаюсь к return c.courseId. Но здесь стоит учитывать, что courseId объект типа Schema.Types.ObjectId. И для того что бы его сравнивать с другими строками, необходимо вызывать метод toString(). Это очень важный момент так как могут возникнуть определенные проблемы.

И сравниваю return c.courseId.toString() === с тем курсом который был передан в параметры function (course), т.е. course.\_id. и также привожу к строке.

И теперь у меня есть потенциальный id, позиция данного курса в массиве, которая есть в корзине. Т.е. если данный idx найдется т.е. >= 0, то значит я уже несколько раз добавлял этот курс и по этому нужно увеличить количество.  
А если же курс не нашелся то тогда его нужно добавить в корзину.

И для этого пишу следующую логику. В случае, когда idx будет равняться -1, то тогда будет происходить добавление элемента т.е. если idx >= 0 это означает что в корзине уже есть такой курс и счетчик должен увеличится. Внутри тела условия у cloneItems[idx]. обращаюсь по индексу с которым работаю, далее обращаюсь к его полю count = и в него я заношу новое значение которое должно увеличится от предыдущего clonedItems[idx].count + 1.

Иначе если idx равен минус одному я обращаюсь к clonedItems и с помощью метода push добавляю новый элемент где по сути нужно описать два поля count и courseId. courseId: будет равняться idx который сюда передавал т.е. course.\_id. Ну а количество по умолчанию count: 1.

Сокращаю код. Далее я обращаюсь к this.cart = {} и в нее заношу значение items: clonedItems. Таким образом я избавляюсь от создания одной переменной. Но более того если бы мы назвали переменную clonedItemsпросто items, то тогда нам бы с этим былобы просто работать. Меняю все clonedItems на items

И теперь когда ключ и значение совпадают this.cart = { items: items }; я могу просто написать this.cart = { items }; т.е. просто опустить значение. И последнее что необходимо сделать в методе addTocart это вернуть return this.save() Обязательно вызываю данный метод.

Отображение корзины.

В контроллеру card.js и в router.get первым параметром добавляю маршрут ‘/’. Вторым параметром принимает callback функцию. В теле функции создаю переменную user которой присваиваю значение объекта req.user в котором содержится id пользователя т.к нужно подождать пока данный пользователь найдется я принимаю оператор await.

Задача заключается в том что бы получить корзину, а корзина находится в модели пользователя. Для получения корзины в объекте req.user я вызываю метод populate c помощью которого я вытаскиваю из модели пользователя из объекта cart массив items в котором содержится объект courseId. Данное обращение я помещаю в параметр метода populate(‘cart.items.courseId’) и для того что бы все работало вызываю метод exectPopulate.

Теперь значения находящиеся в courseId необходимо привести в надлежащий вид для того что бы front-end понимал как с этим работать. В контроллере card.js В get запросе добавляю еще одно поле courses: обращаюсь к модели user.

Далее я хочу сформировать массив который буду передавать на front-end. Для этого создаю функцию function mapCartItems в параметры которой заношу cart. В get запрос передаю результат работы этой функции const courses = mapCartItems(user.cart);

В теле функции mapCartItems я возвращаю новый массив cart.items.map() в параметры помещаю callback функцию где на каждой итерации будет получен некоторый объект у которого все данные относятся к конкретному курсу. они лежат в поле c.courseId. и что бы убрать эту всю метадату нужно обратиться к его полю \_doc. И использую ES6 ...c.userId.\_doc для того что бы развернуть данный курс, и что касается счетчика просто указываю count: c.count.

Далее для корректного отображения цены создаю функцию computerPrice. В параметры передаю массив courses. И в get запросе в поле price передаю результат работы данной функции computerPrice.

Для реализации функции computerPrice, что лежит в массиве courses, необходимо умножить цену на количество курсов и сложить это все со всеми курсами которые есть в массиве, то получиться финальная цена. Для этой задачи я использую метод который называется reduce который есть у массивов, который выполняет задачу в одну строку. Данная функция первым параметром принимает в себя callback функцию, а вторым параметром принимает целочисленное стартовое значение ноль.

В callback первым параметром принимаю объект total который на каждой итерации будет увеличиваться, а вторым принимаю объект course. И в теле callback логика простая. Она будет просто возвращать переменную return total += и будем ее увеличивать на значение course.price \* course.count.

**Слайд 11 Удаление курса из корзины**

Удаление из корзины является асинхронным действием т.к. все данные хранятся в БД.

В контроллере card.js обращаюсь к объекту router и вызываю метод delete. Первым параметром принимаю маршрут ‘/remove/:id’. Вторым параметром принимаю асинхронную callback функцию в параметры которой принимаю req, res. В теле функции обращаюсь к оператору await, у объекта req.user вызываю метод, который я создам позже, removeFromCart в параметры которого принимаю req.params.id. После этого на front-end необходимо вернуть объект корзины в том же формате в котором она была до этого. Для того что бы клиентский JS смог заново динамически отобразить страницу.

Опять создаю объект пользователя const user = await далее обращаюсь к req.user вызываю метод populate('') в котором так же обращаюсь к cart.items.courseId, потому что из id его необходимо превратить в полноценный объект с курсами, и после этого вызываю метод exectPopulate().

Теперь у меня есть пока что пустой объект cart,  res.status(200).json(cart);.

Создаю объект const cart. Так же отдельно нужно создать переменную const courses  которую я буду получать из функции mapCartItems(user.cart) из user.cart. И теперь эти курсы я могу передавать в объект cart. Ну а price  я получаю из функции computePrice(course) куда я передаю course.

Теперь реализую метод removeFromCart куда я передаю id того курса, а именно [req.param.id](http://req.param.id/), который необходимо удалить. Для этого я вызываю модель пользователя. Перехожу в models user.js и здесь, перед module.exports что в конце документа, обращаюсь к userSchema, его мотоду methods и определяю новый метод который называется removeFromCart. Это такая же функция, которую обязательно определять через ключевое слово function. Которая принимает в свой параметр  id курса.

По сути для начало здесь так же нужно клонировать массив const items = [...this.cart.items], с помощью spred оператора обращаюсь к this.cart.items. Создаю новый массив для того что бы не было потенциальных мутаций. Далее я как обычно создаю переменну idx т.е. это потенциальный индекс того курса в массиве items который я должен найти. Но в методе removeFromCart этот индекс будет постоянно определяться, потому что, если нет элемента на front - end, его соответсвенно нет и в карточке. Поэтому здесь ошибка практически не возможна только если кто-то специально будет отсылать не правильные id отдельными запросами.  В const idx = я обращаюсь к items. методу findIndex где на каждой итерации получаю объект c. Далее я возвращаю условие по которому нахожу данный индекс т.е. я сравниваю c.courseId.toString() === id.toString().

Далее в условии проверяю если items[idx]. count эквивалентен одному т.е. один курс добавлен в корзину, тогда его необходимо удалить, а иначе срабатывает счетчик items[idx].count--. После срабатывания счетчика, сразу же переопределяю объект this,cart ={items}. После чего возвращаю this.save() тем самым говоря что пока items в корзине не обновятся дальнейшее действие происходить не будет.

Что касается логики if то здесь я переопределяю массив items = items.filter((c) => c.courseId.toString() !== id.toString()). Я просто удаляю с помощью метода filter элементы т.е. существующий id в корзине c.courseId.toString() я сравниваю с id.toString()который поступил.

Единственная проблема в данной функции заключается в том, что массив items определен через ключевое слово const. Для того что бы было все в норме нужно const заменить на let.

Однако сейчас хоть вся логика и реализована, удаление не будет происходить из-за конфликта id и \_id с нижнем подчеркиванием которое использует MongoDB. Для решения этой проблемы необходимо произвести трансформацию данных на клиенте. Перехожу в models courses.js.

Для того что бы исправить такую ситуацию. Когда id не правильно определяется.

Перерехожу models course.js.

Первым делом я обращаюсь к объекту courseSchema. далее я вызову метод который назыается method() и в его параметрах я могу описать тот метод который я хочу использовать. Первым параметром прописываю toClient т.е. это тот момент когда у нас будет трансформация данных к клиенту.  
Вторым параметром я передаю функцию через ключевое слово function(){}. И в теле функции создаю переменную const course через ключевое слово this. а оно доступно если используется ключевое слово function, и далее вызываю метод toObject() для того что бы получить объект данного курса.  
И по сути теперь необходимо сделать трансформацию. У объекта [course.id](http://course.id/) = course.\_id. И далее для того что бы много трафика не летоло я удаляю лишнюю информацию delete course.\_id. Ну и после этого мне нужно просто вернуть этот курс.

Второй момент заключается в том что кода происходит populate в контроллере card.js поле в …с.courseId.\_doc по умолчанию будет прописываться \_id. Для исправление данной ошибки я прописываю значение id а ключ задаю c.courseId.id. После чего логика удаления курса полностью готова.

**Слайд 12 добавление заказа**

Страница заказов имеется у каждого пользователя. Для этого в представлении views в файле card.hbs создаю еще одну форму. В данную форму помещаю кнопку “Сделать заказ”. В форме прописываю атрибут action=”/orders” в котором указываю обрабатываемый путь и вторым атрибутом указываю каким методом будет обрабатываться данная форма method =”POST”.

Так же в представлении navbar.hbs добавляю ссылку на данную страницу.

В контроллере создаю файл orders.js и далее в главном файл index.js подключаю и регистрирую данный маршрут.

Обращаюсь к router, его методу get. Первым параметром создаю маршрут ‘/’. Вторым параметром принимаю callback функцию в параметрах которой находятся req, res. Обращаюсь к функции res.render. Первым параметром добавляю обрабатываемы путь к основному созданному маршруту. Вторым параметром принимаю объект, состоящий из полей в которые входит ключ и его значение т.е. isOrder: true и title: ‘Заказы’.

Теперь в модели представления создаю данную страницу orders.hbs.

После обрабатываю POST запрос на данную страницу. Первым параметром добавляю маршрут ‘/’. Вторым параметром принимаю асинхронную callback функцию параметры которой принимают req и res. В теле функции пока делаю перенаправление на res.redirect(‘orders’).

Перехожу в models и создаю orders.js. В файле импортирую mongoose и из него достаю класс Schema и конструктор model. Далее создаю переменную const orderSchema которой присваиваю новый объект класса new orderSchema. И после этого с помощью конструкции model.exports где регистрирую новую модель model('Order', orderSchema).

В orderSchema передаю поля которые будут присутствовать у каждого из заказов. Определяю массив courses:[] где каждый элемент будет являться объектом. Добавляю объект course со следующими полями: type: Object, required: true т.е. задаю тип объект, и делаю данное поля обязательным. Так же указываю объект count, счетчик который будет считать количество курсов. В данном объекте указываю поля содержащие ключи и значения, а именно: type: Number, required: true.

Далее описываю пользователя совершивший заказ. В объект user задаю имя пользователя name : String. Создаю объект userId который является своего рода референцией на модель пользователя. Прописываю поля: type: Schema.Types.ObjectId, и указываю ссылку откуда достать этот объект ref: ‘User’ и после чего указываю что userId является обязательным.

Далее создаю объект даты в котором прописываю следующие поля: type: Date, default: Date: now т.е. указываю что тип является датой, по умолчанию берется дата текущего момента.

Теперь на страницу order.js я импортирую модель Order.

И далее прописываю логику в контроллерах.

Получение данных заказов.

Для создания заказа необходимо получить все данные, которые есть в корзине. Для этого в контроллере routes order.js в post запросе создаю переменную const user. Обращаюсь к оператору await который ждет пока будет получен пользователь req.user вызываю метод populate в параметры которого передаю (‘cart.items.courseId’) и для того что бы объекты курсов превратить в объекты добавляю execPopulate(). Тем самым я получаю наполненный объект пользователя, где есть все объекты курсов.

Однако нужно привести к человеческому формату. Создаю переменную const users, обращаюсь к user.cart.items.map(). В параметрах map указываю callback функцию в которой на каждой итерации получаю items т.е. i . В callback оборачиваю объект в круглые скобки, и после этого мне не обходимо писать return. И я просто возвращаю count: i.count, а объекту course присваиваю новый экземпляр course: {…i.courseId.\_doc}. Через Spreed оператор разворачиваю данный объект.

После получения читаемого формата курсов я создаю новый объект const Order = new Order в объект которого передаю два обязательного параметра. Это объект user у которого есть поле name:req.user.name и поле userId: req.user.

Сами курсы я передаю как courses: courses, но так как ключ и значение совпадают пишу просто courses.

И после этого необходимо подождать пока создастся новый заказ await order.save(). Но когда заказ создан это означает что корзину необходимо почистить. Поэтому здесь так же вызоваю метод await req.user. который будет называться к примеру clearCart().

Перехожу models user.js.

userSchema.metods.clearCart = function(){}

И внутри этой функции я почищу this.cart = {items:[]} и скажу что items является пустым массивом. Далее я все сохраняю return this.save(). После успешной очистки происходит перенаправление на страницу orders.hbs.

Теперь эти данные необходимо отобразить на странице «Заказы». И для этого я формирую шаблон. Заголовок страницы называю «Мои заказы». Все заказы относятся к какому – то определенному пользователю т.е. пользователь не может посмотреть заказы другого пользователя.

Но перед этим необходимо реализовать метод get в контроллере order.js. Внутри callback оборачиваю все в try catch. Далее получаю список всльзователя. Создаю переменную const orders и в ней обращаюсь к модели await Order и вызываю метод find() где в объекте прописываю условие. Условие такое что если user.userId совпадает с req.user.\_id то тогда это заказы данного пользователя. Через структуру user.userId я указываю в модели order.js на объект user.

Т.е. я просто указываю, где искать идентификатор пользователя. Ну а что касается req.user.\_id, то это наш текущий id. Далее обращаюсь к методу populate в параметры которого вношу user.userId.

Далее обращаюсь к функции res.render, с помощью которого формирую заказы. Для этого в объекте второго параметра добавляю поле которое передаю на front-end массив orders который трансформирую с помощью orders.map() в параметры которой передаю callback функцию в которой возвращаю объект. Где по сути разворачиваю массив o.\_doc. Дальше считаю цену всего заказа price: o.courses.reduce() в параметры которой принимаю callback. Данная функция callback принимает два параметра, total – какое-то тотальное значение, и значение c – какой – то конкретный курс.

Здесь единственно что немного отличается структура данных. return total += теперь у объекта **c** есть поле count, c.count где храниться количество курсов которые приобрел пользователь умноженное на **c** и дальше нужно перейти в объект course. и только после этого обратиться к полю .price.

Далее с помощью handlebars вывожу все на front-end.

Подводя итог, можно отметить, что рассматриваемый проект имеет фактическое применение в сфере продаж. Таким образом поставленные задачи для разработки дипломного проекта были выполнены. Спасибо за внимание.

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
|  |  |

[1. Анализ объекта **Ошибка! Закладка не определена.**](#_Toc48659172)

[1.1. Описание предметной области. **Ошибка! Закладка не определена.**](#_Toc48659173)

[1.2. Построение концептуальной модели информационной системы интернет магазина. **Ошибка! Закладка не определена.**](#_Toc48659174)

[2. Постановка задачи. **Ошибка! Закладка не определена.**](#_Toc48659175)

[2.1. Опреденление требований к информационной системе интернет магазина. **Ошибка! Закладка не определена.**](#_Toc48659176)

[2.2. Обзор и обоснование выбора инструментальных средств **Ошибка! Закладка не определена.**](#_Toc48659177)

[3. Проектирование **Ошибка! Закладка не определена.**](#_Toc48659178)

[3.1. Разработка архитектуры програмного продукта **Ошибка! Закладка не определена.**](#_Toc48659179)

[3.2. Проектирование структур хранение данных **Ошибка! Закладка не определена.**](#_Toc48659180)

[3.3. Описание реализации вариантов использования **Ошибка! Закладка не определена.**](#_Toc48659181)

[4. Реализация **Ошибка! Закладка не определена.**](#_Toc48659182)

[4.1. Разработка классов информационой системы **Ошибка! Закладка не определена.**](#_Toc48659183)

[4.2 Разработка интерфейса программного продукта **Ошибка! Закладка не определена.**](#_Toc48659184)

[4.3 Разработка алгоритмов реализации вариантов использования **Ошибка! Закладка не определена.**](#_Toc48659185)

[5. Интеграционное тестирование **Ошибка! Закладка не определена.**](#_Toc48659186)

[6. Экономическое обоснование эффективности внедрения **Ошибка! Закладка не определена.**](#_Toc48659187)

[7. Охрана труда **Ошибка! Закладка не определена.**](#_Toc48659188)

[8. Промышленная экология **Ошибка! Закладка не определена.**](#_Toc48659189)

[9. Ресурсосбережение **Ошибка! Закладка не определена.**](#_Toc48659190)

[Заключение **Ошибка! Закладка не определена.**](#_Toc48659191)

[Список использованной литературы **Ошибка! Закладка не определена.**](#_Toc48659192)

[Приложения **Ошибка! Закладка не определена.**](#_Toc48659193)

[ПРИЛОЖЕНИЕ А **Ошибка! Закладка не определена.**](#_Toc48659194)

[ПРИЛОЖЕНИЕ Б **Ошибка! Закладка не определена.**](#_Toc48659195)

[ПРИЛОЖЕНИЕ С………………………………………………………………………………………………………………………… **Ошибка! Закладка не определена.**](#_Toc48659196)