МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО
«Воронежский государственный технический университет»

О.В. Курипта И.А. Наливайко
Я.В. Лынов

РАЗРАБОТКА WEB-РЕСУРСА: ПРАКТИКУМ

Утверждено учебно-методическим советом
университета в качестве учебного пособия

Воронеж 2018

УДК 004.4
ББК 32.973.3я73
К931

Рецензенты:

кафедра высшей математики и информационных технологий Воронежского государственного университета инженерных технологий ()

канд. физ - матем. наук, доцент Н.Б. Горбачева,

Российского экономического университета им. Г.В. Плеханова (Воронежский филиал)

К931 **Курипта, О.В.**
РАЗРАБОТКА WEB-РЕСУРСА: практикум / О.В. Курипта, И.А. Наливайко, Я.В. Лынов; ВГТУ. – Воронеж, 2018. – 101 с.

ISBN

В данном практикуме рассматриваются современные подходы к разработке web-ресурсов. Даются в достаточном объёме сведенья, необходимые для реализации web-ресурса.

Практикум предназначен для студентов, обучающихся по направлению подготовки 09.03.02. - «Информационные системы и технологии», а также для студентов вузов, магистрантов, интересующихся web-разработкой.

Ил. 43. Табл. 4. Библиогр.: 15 назв.

УДК 004.4
ББК 32.973.3я73

ISBN 978-5-89040-575-3

ВВЕДЕНИЕ

В современной жизни интернет стал очень важной площадкой для бизнеса. Многие компании используют его в качестве основного источника привлечения клиентов, а также для продажи каких-либо товаров или услуг. Как правило, для этого разрабатывается и применяется web-ресурс компании, перед которым, ставятся определенные цели и задачи.

Таким образом, web-технологии на сегодняшний день являются очень перспективным направлением, поскольку позволяют создавать информационные системы самой разной сложности и целевой направленности. Технологии совершенствуются и развиваются с невероятной скоростью, и предоставляют разработчикам поистине огромные возможности.

В данном пособии излагаются основные этапы реализации web-ресурса. Главной задачей предлагаемого пособия является знакомство читателей с практическими вопросами создания, как клиентских приложений web-страниц, так и серверных программ. Также затрагиваются вопросы формирования концепции развития web-ресурсов, организации его структуры и размещения в сети Интернет.

В практической части рассмотрен пример разработки web-ресурса, который можно отнести к категориям B2B (business-to-business) и B2C (business-to-consumer) и представляет собой интернет-магазин, специализирующийся на продаже строительного материала.

1. ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ РАЗРАБОТКИ WEB-РЕСУРСА

Стандартов, однозначно регламентирующих проектирование и разработку web-ресурсов нет.

Тем не менее, существует ряд документов, методик и ГОСТов, которые используются при разработке web-ресурсов.

Например, ГОСТ ИСО/МЭК 12207-99. Информационная технология. Поддержка жизненного цикла программных средств. Стандарт устанавливает общую структуру программных продуктов.

ГОСТ РИСО/МЭК 9126-93 – оценка программной продукции. Характеристики качества и руководства по их применению. Стандарт определяет шесть характеристик, которые описывают качество программного продукта [12].

Основные модели проектирования описаны в различной литературе и применяются при разработке сайтов. Существует несколько моделей проектирования web-ресурса [1].

Модель водопада определяет реализацию нескольких, выполняемых друг за другом этапов. В самом начале исходит разработка технического задания, на втором этапе – анализ и проектирование ресурса; последующие шаги - создание контента, дизайн и программирование. Окончание - финальное тестирование и приемка проекта.

Достоинства представленного подхода - простота и последовательность. Разработка ведется в заданные сроки одной командой. Вся команда делает один этап. Недостаток состоит в том, что модель предполагает точное знание того, чего хочется реализовать на сайте. Обычно наоборот, очень трудно сразу сформулировать цели, которые следует выполнить. Сложные проекты требуют командной работы, распараллеливания процессов и разбиения на подзадачи.

Другой тип модели - спиральная модель.

Данная модель использует противоположный подход, нежели модель водопада. Работа начинается с этапа «Планирование и анализ» и по часовой стрелке переходит к этапам

выполнения, тестирования полученных результатов и оценки. На следующей итерации все повторяется по новой, но уже с учетом выявленных недочетов проекта. Таким образом, пройдя несколько итераций и повторив все этапы несколько раз, проект избавляется от недостатков, обрастая дополнительными возможностями и преимуществами.

Достоинство - возможность разрабатывать проект за несколько итераций позволяет постепенно его улучшать, реализовывая различные идеи. Недостаток состоит в том, что в данной модели отсутствуют четкие критерии выполнения проекта. Это создает определенные сложности для расчета финансовых затрат на выполнение проекта.

Модель MicrosoftSolutionsFramework (MSF) сделана компанией Microsoft для своих собственных целей, но приобрела популярность и среди других разработчиков.

Все программные продукты Microsoft создаются именно по этой методологии. Модель MSF вобрала в себя лучшее из моделей, описанных выше - спиральной и водопада. Данная модель состоит из четырех этапов: анализ, планирование, разработка и стабилизация. Каждый этап заканчивается достижением определенного результата, например, после анализа проекта идет одобрение продуманной концепции. В итоге пишется определенный документ, в котором записывается результат выполнения данного этапа, для того, чтобы каждый из разработчиков четко понимал свое место в проекте и задачи, которые ему предстоит решить. Модель итерационна и при прохождении всех этапов, проект можно доработать с учетом предыдущей итерации. Но, поскольку, окончание каждого этапа четко указано, нет, как в спиральной модели, бесконечного повторения одного и того же процесса.

К достоинствам данной модели относится то, что MSF является одной из самых интересных моделей разработки и создания проектов взявшее лучшее из других моделей и отказавшись от присущих им недостатков.

Таким образом, можно в общем виде выделить следующие этапами разработки web-ресурса [9]:

1. Сбор и анализ требований по предметной области.
2. Разработка технического задания.
3. Проектирование интерфейса.
4. Создание дизайн – концепции web-ресурса.
5. Создание макетов страниц (верстка web-ресурса).
6. Создание мультимедиа.
7. Программирование (разработка функциональных инструментов) или имплементация на CMS-систему.
8. Подготовка и размещение материалов.
9. Тестирование и внесение корректировок.
10. Размещение проекта на публичной площадке.
11. Обслуживание работающего web-ресурса или его программной основы.

В зависимости от выполняемой задачи, какие – то этапы могут отсутствовать, а какие – то тесно связаны между собой.

Основным процессом разработки web-ресурса является анализ предметной области, включающий проведение необходимых исследований по формированию структуры web-ресурса и получении, в конечном счете, задания на проектирование, и предполагающий проведение работы по сбору необходимой информации в связи с особенностями предметной области.

Таким образом, чтобы разработать и реализовать полноценный web-ресурс, необходимо правильно поставить задачу. Web-ресурс как и любая другая информационная система принимает входные данные и на их основе формирует выходную информацию. К входной информации можно отнести действия (данные), которые совершает пользователь на страницах: регистрация, авторизация, оставление отзывов, покупка, продажа, формирование заявлений, подача документов и т.п. Под выходными данными подразумевается результат взаимодействия пользователя (клиента) с web-ресурсом: сформированный заказ, полученная консультация или любая другая необходимая информация.

В процессе анализа предметной области необходимо выделить ключевые сущности и на основе полученной информа-

ции построить модель базы данных (БД), спроектировать структуру, интерфейс и дизайн web-ресурса.

Логическая модель базы данных описывает понятия предметной области, их взаимосвязь, а также ограничения на данные, налагаемые предметной областью.

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но без привязки к конкретной СУБД.

База данных должна корректно отражать предметную область. Это означает, что должны выполняться следующие условия.

1. Состояние базы данных в каждый момент времени должно соответствовать состоянию предметной области.

2. Изменение состояния предметной области должно приводить к соответствующему изменению состояния базы данных.

3. Ограничения предметной области, отраженные в модели предметной области, должны некоторым образом отражаться и учитываться в базе данных.

На основе разработанной моделей предметной области и базы данных разрабатывается структура web-ресурса, то есть необходимо определить количество страницы, и их информационное наполнение: какую информацию будет отображать и принимать страница.

Проектирования интерфейса пользователя (UI) – сложная многофакторная и многовариантная задача, требующая системного подхода [10]. Цель создания эффективного эргономичного пользовательского интерфейса заключается в представлении информации максимально удобной для человеческого восприятия и структурировании отображения на дисплее таким образом, чтобы привлечь внимание к наиболее важным единицам информации.

На сегодняшний день существует многогранная концепция User eXperience (UX), которая включает множество дисциплин: интерактивный дизайн, информационную архитекту-

ру, визуальный дизайн, юзабилити и взаимодействие между человеком и компьютером. User eXperience – это опыт пользователя, опыт взаимодействия – это восприятие и ответные действия пользователя, возникающие в результате использования и/или предстоящего использования продукции, системы или услуги (ISO 9241-210).

Пользователь должен иметь возможность манипулировать объектами в среде приложения. Важно, чтобы данные объекты (графические элементы) были понятны пользователю и предоставляли информацию, что это такое и что произойдет, если выбрать или произвести действие над каким-то объектом. Концепция согласованности состоит в том, что при работе с компьютером у пользователя формируется система ожидания одинаковых реакций на одинаковые действия, что постоянно подкрепляет пользовательскую модель интерфейса.

Так же при разработке приложения должна предусматриваться в любой момент и на любом этапе работы возможность получения пользователем помощи, контекстной справки или подсказки.

Безусловно, пользователю нужно дать возможность экспериментировать в приложении (нажатие любых кнопок, изменение настроек и т.д.). Но при этом необходимо избавить его от тупиковых ситуаций: все последствия экспериментов должны быть исправимы, а в лучшем случае еще и обратимы.

Теоретически можно всё реализовать и на одной странице, однако это не всегда удобно как в разработке, так и пользователю. Поэтому страницы разрабатываются, исходя из ключевых сущностей, представленных в инфологической модели. Представленная информация не должна «ускользать» от пользователя: информации не должно быть слишком много, так как воспринимать информацию будет затруднительно, целесообразно разбить её на части, или попробовать решить проблему с точки зрения удобного дизайна. Это важно помнить при разработке проектов «справочного направления». Справедливо и обратное: объединение страниц по «общему смыслу», если страницы получаются пустоватыми, что, с точ-

ки зрения законов художественной композиции, а значит, и маркетинга, будет выглядеть не привлекательным. Это важно учитывать при разработке интернет магазинов.

После того, как определена структура страниц, необходимо приступить к разработке дизайна. Ввод информации на страницу будет обеспечиваться с помощью управляющих элементов (текстовых полей, кнопок, раскрывающихся списков, флажков и т.д.). Необходимо грамотно разместить их на странице так, чтобы пользователь сразу мог увидеть информацию, за которой «пришёл» на данный ресурс, быстро понять какую информацию от него хочет получить web - ресурс, куда её вводить и в каком виде.

Большие громоздкие инструкции для ввода ФИО и номера телефона – не лучший вариант. Наиболее важную информацию желательно поместить в центр, разнородную информацию писать в разных блоках, отделять явно выраженными заголовками. Наиболее важные (ключевые) слова можно выделить или подчеркнуть. Красивые изображения размещать там, где они уместны, притом, только такие, которые не будут резко отличаться от общего дизайна web - ресурса. Делать фон такими изображениями, которые не препятствуют быстрому восприятию информации, расположенной поверх фона. Например, фон в виде чёткой фотографии газеты будет мешать чтению информации. Стараться сохранять баланс между простотой и желанием разместить как можно больше красивых картинок/иконочек или управляющих элементов на странице с тенями и обводками.

Цветовая гамма, компоновка элементов, пиктограммы, звуки, анимация – все должно помогать пользователю при выполнении задачи, но ни в коем случае рассеивать внимание пользователя. Начальным этапом разработки пользовательского интерфейса являются создание его ассоциативной модели, после чего осуществляется проработка концептуального дизайна. Здесь необходимо разработать необходимый набор интерфейсных элементов, каждый из которых должен обла-

дать определенным цветом, формой, надписью и т. п., и все вместе они должны составлять единую систему – UX.

Меню необходимый элемент любой автоматизированной системы, позволяющий пользователю выполнять задачи внутри приложения и управлять процессом решения. Достоинство меню в том, что пользователи не должны помнить название элемента или действия, которое они хотят выполнить – они должны только распознать его среди пунктов меню.

Сообщения необходимы для направления пользователя в нужную сторону, подсказок и предупреждений для выполнения необходимых действий на пути решения задачи. Они также включают подтверждения действий со стороны пользователя и подтверждения, что задачи были выполнены системой успешно либо по каким-то причинам не выполнены. Сообщения могут быть обеспечены в форме диалога, экранных заставок и т.п.

После того, как реализованы макеты страниц со всеми элементами и блоками для отображения важной информации приступают непосредственно к разработке самих страниц с использованием языков программирования.

После разработки страниц web-ресурса, переходят к подключению базы данных и реализации их функциональной составляющей.

После того как web-ресур готов, необходимо провести валидацию html кода [13]. Html валидация**,** это важная и неотъемлемая часть практически любого web-ресурса, которая представляет собой проверку кода на наличие ошибок и его корректность. Такую проверку можно произвести с помощью практически любого html-валидатора, то есть программного продукта или онлайн-сервиса. Валидация html кода состоит из нескольких видов.

- Валидация синтаксиса. Данный вид является, одним из основных, так как от него зависит многое в процессе продвижения web-ресурса в поисковых системах, а так же корректность отображения всех элементов web-ресурса.

- Вложенность кода. Данная часть проверяет правильность закрытия всех тэгов, их порядок и т.п.
- Валидация Document Type Definition. Данная проверка аналогична проверке на грамотность, т.е. проверяет правильность написания всех тэгов, атрибутов и прочих элементов кода, а также элементов внутри определенного кода.
- Посторонние коды. Позволяет выявить ошибки в посторонних кодах.

Так же немало важным, а даже самым основным для получения прибыли при использовании web-ресурса является его продвижение в сети Интернет. Именно продвижение web-ресурса в поисковых системах является одним из самых эффективных методов привлечения потенциальных клиентов, и для этого применяются следующие инструменты: поисковая оптимизация (SEO), контекстная реклама, медийная (баннерная) реклама, маркетинг в социальных медиа, вирусная реклама, email-маркетинг. Рассмотрим эти инструменты интернет-маркетинга более подробно.

Первым инструментом интернет-маркетинга является поисковая оптимизация (Search Engine Optimization – SEO), которую можно определить, как комплекс мер по внутренней и внешней оптимизации для поднятия позиций web-ресурса в результатах выдачи поисковых систем по определенным запросам пользователей с целью привлечения потенциальных клиентов.

На положение ресурса в выдаче поисковых систем влияют внешние и внутренние факторы. Внешние факторы определяют релевантность ресурса на основании цитируемости его внешними ресурсами, а также их авторитетности, в зависимости и вне зависимости от текста цитирования. Основными внутренними факторами могут быть названы: доменное имя; заголовок web-страницы и ключевые слова в нем, которые должны совпадать с формой запроса пользователя в поисковой системе; информационное наполнение web-ресурса и наличие в нём ключевых слов; количество гиперссылок на

веб-ресурсе; HTML-разметка; мета-теги: «title», «description», «keywords».

Внутренняя оптимизация направлена на улучшение web-ресурса в целом, повышение его пользы для пользователя. К внутренней оптимизации относится работа над структурой ресурса, его дизайном и юзабилити (usability), повышение качества информационного наполнения web-ресурса и облегчение его восприятия. Также проводится работа с HTML-кодом: заголовками web-страницы и мета-тегами. Мета-теги – это специальные элементы заголовков HTML-документов, несущие служебные функции. В интернет-маркетинге применяется определенный набор мета-тегов, которые оказывают влияние на эффективность индексации ресурса поисковыми системами. Наиболее важен мета-тег «description», так как именно он отображается под URL web -ресурса в поисковой выдаче, в нем содержится расширенная информация, раскрывающая суть заголовка. Вторым инструментом интернет-маркетинга является контекстная реклама, размещаемая в поисковой выдаче и соответствующая своим содержанием поисковому запросу, то есть релевантная ему. Контекстная реклама может быть представлена в виде текста, баннера, видеоролика, содержащего гиперссылку.

Преимущество контекстной рекламы – очень быстрый результат. Число обращений в организацию значительно возрастает в достаточно короткие сроки. Это происходит за счет того, что контекстная реклама в первую очередь ориентирована на людей, которые готовы к совершению покупки, но, как следствие, этот вид рекламы практически не влияет на людей, которые не готовы к совершению покупки. Так же контекстная реклама будет низко эффективна для web-ресурса, который имеет плохую оптимизацию. Выделяют поисковую и тематическую контекстную рекламу. Поисковая реклама выводится среди результатов поиска и отбирается в зависимости от поисковых запросов. Такого вида реклама эффективна в случае открытия нового офиса продаж, проведения мероприятия или краткосрочной акции. Тематическая реклама встраивается

в страницы web-ресурсов, входящих в рекламную сеть. Подбор рекламных объявлений происходит в зависимости от тематики web-ресурса либо на основе истории поиска пользователя.

Следующим инструментом интернет-маркетинга является медийная реклама, она представляет из себя текстово-графические рекламные материалы, которые размещаются на web-ресурсах, являющихся рекламными площадками. Чаще всего медийная реклама используется в форме баннерной рекламы. Баннер – это рекламное изображение, имеющее фиксированный размер, в некоторых случаях, содержащее анимацию, и при этом выполняющее роль гиперссылки на тот или иной веб-ресурс. Баннерная реклама позволяет значительно повысить качество и привлекательность рекламного сообщения. Тем не менее, баннер не обязательно должен содержать гиперссылку, он может содержать полезную для потребителя информацию. Данный вид рекламы достаточно универсален и эффективен.

Четвертым инструментом интернет-маркетинга является маркетинг в социальных медиа, которых представляет собой комплекс мероприятий, которые направлены на увеличение посещаемости web-ресурса путем привлечения пользователей из социальных сетей, тематических блогов или сообществ. Данный вид маркетинга включает в себя два направления: оптимизацию в социальных медиа – Social Media Optimization (SMO) – и продвижение в социальных медиа – Social Media Marketing (SMM).

Оптимизация в социальных медиа очень сильно похожа на поисковую оптимизацию, но направлена не на поисковые системы, а на социальные сети и блоги. В рамках SMO, необходимо создание интересного, уникального контента, работа над дизайном web -ресурса и его юзабилити usability).

Продвижение в социальных медиа позволяет точечно воздействовать на целевую аудиторию, выбирать площадки, где эта аудитория представлена в большей степени, а также применять наиболее подходящие способы коммуникации с

ней, при этом в наименьшей степени затрагивая не заинтересованных в этой рекламе людей.

Не менее важным является следующий инструмент интернет-маркетинга – вирусный маркетинг, который представляет собой собирательное название самых различных методов распространения рекламы, передающейся в прогрессии, близкой к геометрической. Основные распространители данной информации являются ее получатели. Рекламное сообщение, как правило должны быть ярким, творческим и очень необычным.

Успех вирусного маркетинга основывается на том, что получающий информационное сообщение человек, должен быть уверен, что оно исходит от незаинтересованного лица. Основной причиной этого является огромное недоверие к рекламной информации, а также мнение, что реклама преувеличивает реальные качества товара.

Шестым и, пожалуй, последним инструментом интернет-маркетинга является email-маркетинг. Он служит для прямой коммуникации между организацией и потребителями, потенциальными и реальными. Так, основная цель email-маркетинга – увеличение лояльности потребителей к организации, а также увеличение новых и повторных продаж. Данный вид маркетинга имеет ряд положительных сторон: очень низкая стоимость, сбор базы потенциальных и реальных потребителей, осуществление коммуникации между организацией и потребителем, получение необходимых действий от подписчиков (комментарии, оформленные заказы, заявки, скачанные бесплатные материалы).

Однако разумнее всего использовать данные инструменты интернет-маркетинга в совокупности.

Самым главным достоинством web-ресурса является то, что нет необходимости устанавливать экземпляры приложения на несколько компьютеров. Все, что требуется для полноценной работы – это браузер, поставляемый вместе с операционной системой, и доступ в сеть Интернет [15].

В случае с web-приложением, лежащем на сервере, все задачи связанные с администрированием приложения и его масштабированием, решаются централизованно на стороне сервера.

Web-приложения не требовательны к ресурсам и к аппаратной платформе. Это значит, что нет никакой разницы, сколько мегабайт оперативной памяти установлено на устройстве пользователя, и под управлением какой операционной системы он работает.

Также стоит отметить, что использование web-приложения упрощается переход к новым версиям, а также не возникают проблемы обратной совместимости. В случае обновления web-приложения все изменения, внесенные в него, поступают ко всем пользователям своевременно и без временных затрат.

И наконец, web-приложения позволяют своим пользователям быть по-настоящему мобильными. По сути, пользователь может работать в сети, сохранять результаты своей работы на сервере и, в случае необходимости, иметь к ним доступ там, где есть выход в Интернет.

2. ОСНОВНЫЕ ЭТАПЫ РАЗРАБОТКИ СЕРВЕРНОЙ ЧАСТИ WEB-РЕСУРСА

Для разработки серверной части web-ресурса (далее проект) необходимо установить сервер. Предлагается применить локальный сервер **OpenServer**.

OpenServer – это портативная серверная платформа и программная среда, включающая в себя набор серверного программного обеспечения, удобный, многофункциональный продуманный интерфейс, обладающая мощными возможностями по администрированию и настройке компонентов. Платформа широко используется с целью разработки, отладки и тестирования web-проектов, а так же для предоставления web-сервисов в локальных сетях [14].

OpenServer работает как в стационарном, так и портативном режиме. Достоинства OpenServer: легкость установки; огромная функциональность, которая достигается благодаря массе всевозможных настроек; стабильное обновление системы; частые обновления повышающие производительность и скорость работы; бесплатная лицензия.

OpenServer включает в себя все необходимые модули для хранения, разработки и тестирования web-ресурса:

- SPanel;
- Apache;
- Nginx;
- MySQL;
- PostgreSQL;
- FTP FileZilla;
- PHP;
- ImageMagick;
- Bind;
- Git;
- PHPMyAdmin.

После установки и запуска сервера, станет доступно управление локальным сервером. Необходимо открыть интер-

нет браузер и прописать в адресной строке: **http://127.0.0.1(или localhost)/openserver/phpmyadmin.** В результате откроется страница авторизации на сервере. Затем выбирается язык, в поле пользователь прописывается «root» (без кавычек). Поле для ввода пароль оставляем пустым. Затем нажать на кнопку «ОК», чтобы перейти на главную страницу.

Для создания базы данных (БД) в **phpMyAdmin**, необходимо на главной странице перейти во вкладку «Базы данных», далее ввести в текстовое поле название базы данных без пробелов и нажать на кнопку «Создать» (рис. 2.1).

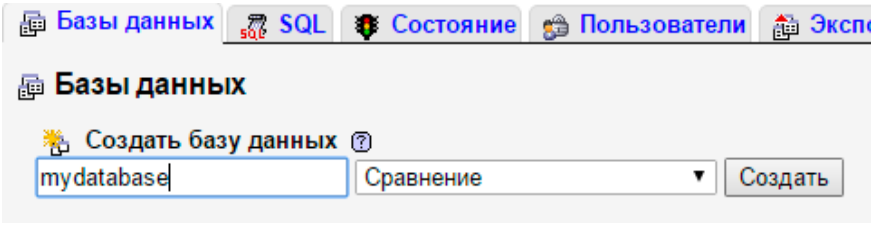

Рис. 2.1. Создание базы данных в phpMyAdmin

В результате в списке баз данных появится новая запись. Для работы с новой созданной базой данных, необходимо кликнуть по ее названию в списке.

Рассмотрим процесс разработки таблиц, представленных на рис. 2.2.

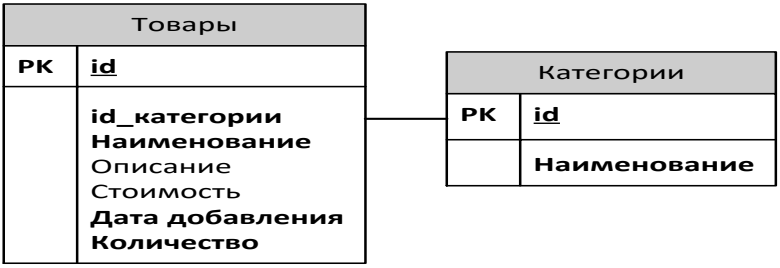

Рис. 2.2. Пример таблиц

Для создания таблицы в текущей БД необходимо ввести название таблицы и количество полей (столбцов) в текстовые поля, как это показано на рис. 2.3, и нажать на кнопку «ОК».

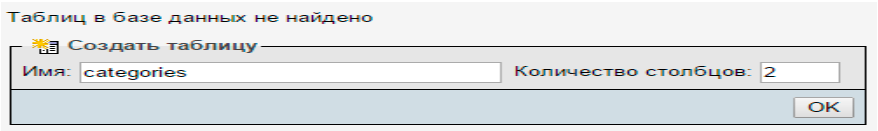

Рис. 2.3. Пример создания таблицы в phpMyAdmin

В результате появиться страница с созданием структуры таблицы, куда добавляются имена для полей как это сделано на рис. 2.4.

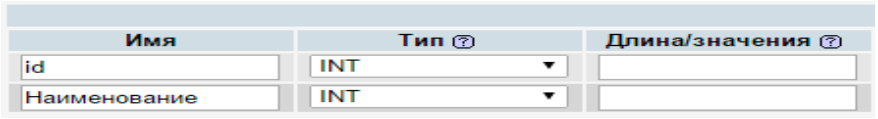

Рис. 2.4. Добавление полей в таблицу

Следующим пунктом идет «Тип». Основные и наиболее часто используемые типы данных в **MySQL** приведены в табл. 2.1.

Для поля «**id**» определяется тип **INT**, а для «Наименования» тип **VARCHAR** длиной в 255 символов (рис. 2.5).

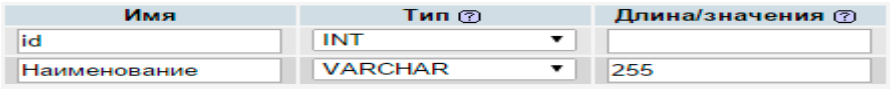

Рис. 2.5. Установка типов полей в таблице

Необходимо обратить внимание на следующие поля структуры, представленные на рис. 2.6. Если у атрибута «**Null**» стоит галочка, то это означает, что значение в текущем поле таблицы может быть пустым, а если галочки нет, то в

этом поле обязательно должно находиться значение (т.е. является обязательным).

Таблица 2.1

Типы данных языка MySQL

Название типа	Описание
INT	Целочисленный тип диапазоном от -2147483648 до 2147483647
FLOAT	Малое число с плавающей точкой. Допустимые значения: от -3,402823466E+38 до -1,175494351E-38
DOUBLE	Число с плавающей точкой удвоенной точности. Допустимые значения: от -1,7976931348623157E+308 до -2,2250738585072014E-308
VARCHAR	Строка переменной длины в диапазоне от 1 и до 255 символов
TEXT	Строка с максимальной длиной 65535 символов
DATE	Представляет собой дату в интервале от '1000-01-01' до '9999-12-31'
TIME	Хранит время в интервале от '-838:59:59' до '838:59:59'
DATETIME	Хранит комбинацию даты и времени в интервале '1000-01-01 00:00:00' до '9999-12-31 23:59:59'
BOOLEAN	Хранит малое целочисленное значение, где 0 является ложью, а все остальное является истинной
DECIMAL(M[,D])	Число с плавающей точкой, которое хранится в виде строки, для него необходимо определить длину (M) и кол-во символов дробной части (D). Если D – не указанно, то число является целым.

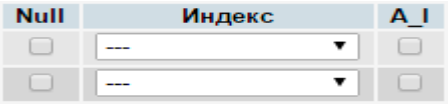

Рис. 2.6. Атрибуты полей таблицы

В поле атрибута «Индекс» можно выбрать первичный ключ, ограничение уникальности (значение в поле будет всегда уникальным, добавление поля со значением которое уже присутствует в таблице, приведет к ошибке) и индексацию (для быстрого поиска).

Атрибут «**A_I**» (**Auto Increment**) устанавливает для текущего поля счетчик.

В таблицы «Категории» для поля **id** устанавливаем Индекс «**Primary Key**» (первичный ключ) и **A_I** (автоинкремент) и нажимаем на кнопку «Сохранить».

После этого в списке таблиц в базе данных должна появиться первая запись (как на рис. 2.7).

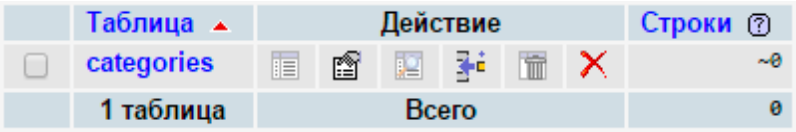

Рис. 2.7. Список таблиц в БД

Для создания таблицы «Товары» проводятся те же самые операции, что и для таблицы «Категории». В результате должна получиться структура подобная той, что представлена на рис. 2.8.

После того, как две таблицы созданы, необходимо их связать.

В списке таблиц щелкаем по названию таблицы «Товары», переходим на вкладку «Структура», справа от структуры есть различные действия: редактировать, удалить, назначить первичный ключ, сделать поле уникальным и добавление ин-

декса, по которому нужно кликнуть один раз напротив поля «**id_категории**» (все действия представлены на рис. 2.9).

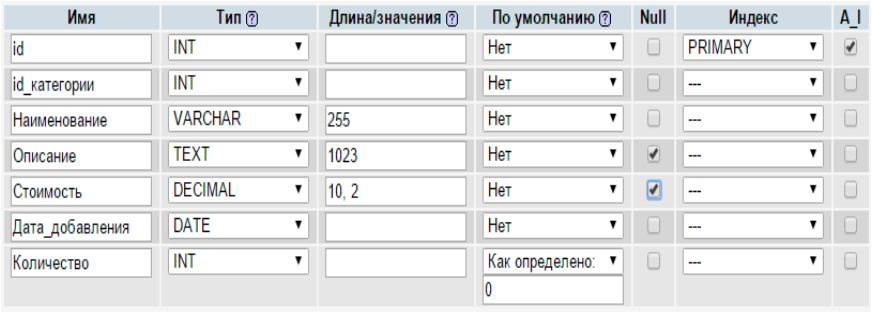

Рис. 2.8. Атрибуты полей таблицы «Товары»

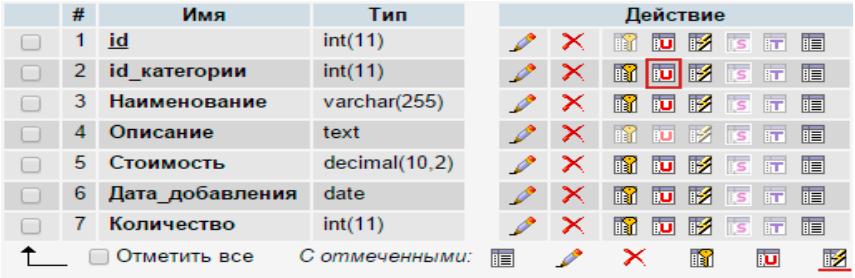

Рис. 2.9. Действия с таблицами в phpMyAdmin

Также можно выделить сразу несколько полей, нажать на действие снизу и это действие выполнится сразу для нескольких полей. Далее нужно кликнуть по пункту «Связи» (рис. 2.10) и связать поле «**id_категории**» у таблицы «Товары», с полем «**id**» у таблицы «Категории» (рис. 2.11).

Рис. 2.10. Пункт «Связи» в меню

Здесь можно дать имя ограничению внешнего ключа и выставить стратегии для обеспечения целостности данных для обновления и удаления. После нажатия на кнопку сохранить будет создана связь между таблицами.

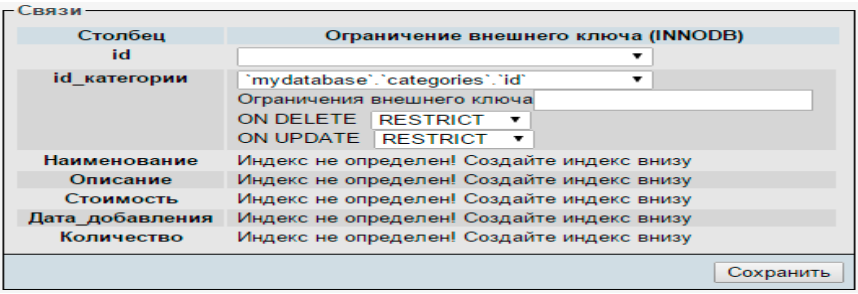

Рис. 2.11. Связывание двух таблиц в phpMyAdmin

Для того чтобы добавить запись в таблицу, нужно в списке таблиц базы данных выбрать нужную (если записей в базе данных нет, то начинать нужно с главных таблиц), выбираем таблицу «Категории» после этого перейти на вкладку «Вставить» и заполнить все необходимые данные. Поле с автоинкрементном заполнять не нужно (рис. 2.12).

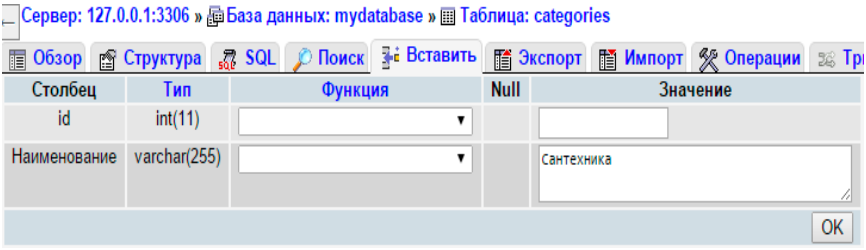

Рис. 2.12. Заполнение полей таблицы БД на phpMyAdmin

После заполнения необходимых полей нужно кликнуть по кнопке «ОК» и запись будет успешно добавлена.

Аналогичным образом заполняется таблица «Товары» (рис. 2.13).

Для просмотра записей в таблице нужно перейти во вкладку «Обзор». В результате появятся записи, которые хранятся в таблице (рис. 2.13). Записи можно редактировать, копировать и удалять. Процессы редактирования и копирования, практически не отличаются от добавления записи, а если кликнуть по значку «удалить», то появится диалоговое окно с подтверждением, при нажатии кнопки «ОК» запись будет безвозвратна удалена. Пример записи в таблице представлен на рис. 2.14.

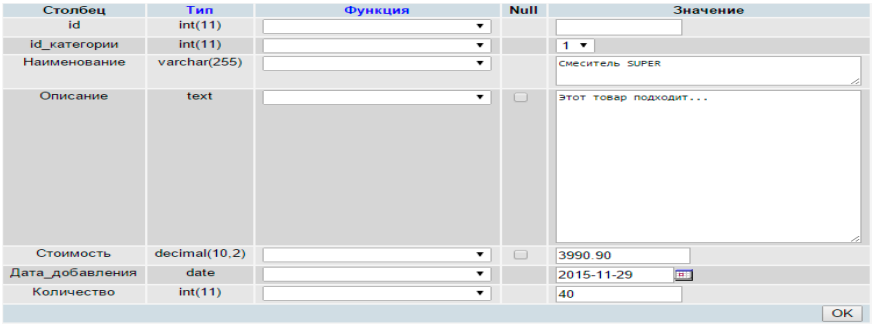

Рис. 2.13. Заполнение таблицы «Товары»

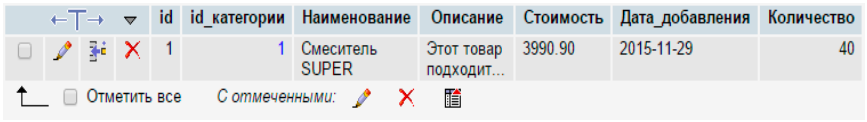

Рис. 2.14. Просмотр записей в таблице на phpMyAdmin

Взаимодействие БД и страниц web-ресурса предлагается реализовать с помощью серверного языка **PHP**.

Рассмотрим основные положения при работе с языком PHP [2,6].

Для того чтобы написать исполняемый **PHP** код, интерпретатору надо явно указать где начинается исполняемый код.

Для этого служит следующий блок ***<?php //Тут должен находится PHP код ?>*** в него уже не получится просто так написать теги, т.к. код внутри него относиться уже к **PHP,** а не к **HTML**. Справедлива и обратная ситуация **PHP** код не получится написать вне этого блока, т.к. интерпретатор тогда воспримет его обычным **HTML** кодом.

Комментарии

В PHP есть многострочные комментарии:

```
/* Все что написано внутри этого
блока будет проигнорировано  */
```

и однострочные комментарии:

```
//Комментарий 1 в стиле C/C++
# Комментарий 2 в стиле Unix
```

Оператор вывода

Для того чтобы вывести какое либо значение необходимо использовать оператор echo <значение>, например:

```
<?php $message = "HELLO"; ?>
<p>
    <?php echo $message;?> WORLD!
</p>
```

Так же этот код можно написать проще с помощью операции конкатенации:

```
<?php
      $message = 'HELLO';
      echo '<p>'.$message. ' WORLD! </p>';
?>
```

Здесь операция "." (точка) выполняет функцию склеивание строк.

В большинстве случаев при использовании знака двойной и одинарной кавычки разница не принципиальна, однако

если вовнутрь строки надо поместить значение переменной или специального символа \r,\n\t и т.п., то необходимо использовать знак двойной кавычки:

```
$message = 'HELLO';
echo "<p>$message WORLD! </p>";
```

Объявление переменных

Язык PHP – не типизирован (динамической типизации), т.е. не требует от разработчика указания типа переменной. Для того чтобы объявить переменную нужно перед ее именем поставить знак "$", и в дальнейшем при использовании этой переменной перед ней всегда должен стоять этот знак, например:

```
$a = 10;          // целый тип (integer)
$b = 2.134;       //   вещественный   тип   данных
(float, double)
$c = true   // логический тип (boolean)
$d = "Hello";     // строковый тип (string)
```

Язык PHP самостоятельно определяет тип переменной в зависимости от значения, которое она содержит, но это значение можно менять в ходе выполнения программы:

```
$a = 10;          // был целый тип
$a = "Hello";  // стал строковым
$a = 5.8;   // а за тем стал вещественным
```

Создание массивов

```
$arr = array(
    <ключ 1> => <значение 1>,
    <ключ 2> => <значение 2>,
    …
);
```

Если ключи не определять, то PHP определяет их самостоятельно начиная с самого большого числового индекса, с шагом +1. Примеры создания одномерных массивов:

```
$arr1 = array(1, 2, 3, 4);
$arr2 = array(0 => "A", 2 => "B", 4 => "C");
$arr3 = array("color" => "blue", "size" => 20,
"price" => 2.15);
```

Обращение к элементам массива:

```
echo $arr1[2];        //Выведет: 3
echo $arr2[4];        //Выведет: C
echo $arr3["color"]; //Выведет: blue
```

С версии PHP 5.4 доступна сокращенная запись объявления массива:

```
$arr = [
    <ключ 1> => <значение 1>,
    <ключ 2> => <значение 2>,
    …];
```

Пример создания одномерных массивов, с применением сокращенной записи:

```
$arr1 = [1, 2, 3, 4];
$arr2 = [0 => "A", 2 => "B", 4 => "C"];
$arr3 = [color" => "blue", "size" => 20, "price" =>
2.15];
```

Примеры создания многомерных массивов:

```
$arr1 = array(
   array(1, 2, 3),
   array(4, 5, 6)
);
$arr2 = array(
   0 => array(10 => 1, 2, 3),
   "A" => true
);
```

```
$arr3 = array(
   "parameters" => array(
      "color" => "green",
      "size" => array("width" => 5,
"height" => 4)
   ),
   "price" => 2.15
);
```

Обращение к элементам многомерного массива:

```
echo $arr1[1][0];                          //Выведет: 4
echo $arr2[0][11];                         //Выведет: 2
echo $arr3["parameters"]["size"]["height"]; //Выведет: 4
```

Для быстрого просмотра содержимого массива, можно использовать функцию **print_r(<массив>)**:

```
<?php
$tovar = array(
   "parameters" => array(
      "color" => "green",
      "size" => array("width" => 5, "height" => 4)
   ),
   "price" => 2.15
);
$tovar[] = "END";
echo "<pre>";
print_r($tovar);
echo "</pre>";
```

Результат работы кода представлен на рис. 2.15.

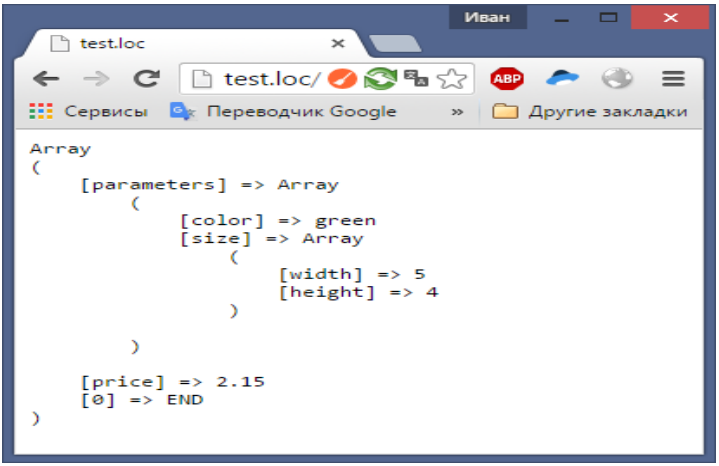

Рис. 2.15. Вывод массивов на экран

Условный оператор

```
if (<условие 1>) {
   //Блок выполнится, если <условие 1> выполняется
}
[else if (<условие 2>) {
   //Блок выполнится, если <условие 1> не выполня-
ется,
   //а <условие 2> выполняется
}]
[else {
   //Блок кода выполнится, если <условие 1> и
<условие 2> не выполняются
}]
```

Тернарный оператор

```
<результат> = (<условие>) ? <1> : <2>
```

Пример, программа напечатает «Нет»:

```
$res = (5 > 6) ? "Да" : "Нет";
echo $res;
```

Циклы

Цикл **while**:

```
while (<условие>) { //Цикл будет выполняться пока
условие истинно
         //Тело цикла}
```

Цикл **do while**:

```
do{
         //Тело цикла, как минимум 1 раз будет вы-
полнено
} while (<условие>); //Цикл будет выполнятся пока
условие истинно
```

Цикл **for**:

```
for (<1>; <2>; <3>){
    //Тело цикла <4>
}
```

Цикл for делится на 4 блока.

Шаг 1: Блок <1> - выполняется всего один раз в самом начале. В этой части прописывается: вызов функции, создание переменной и т.п., а можно и вовсе оставить его пустым.

Шаг 2: Блок <2> - если значение в нем ложно (**false**) то цикл не выполняется и программа переходит к выполнению следующего оператора, расположенного после цикла, если же значение блока <2> является значением отличным от 0 (**null**), то цикл переходит к шагу 3.

Шаг 3: На этом шаге будет выполнено тело цикла – блок <4>.

Шаг 4: Цикл выполняет все, что прописано в блоке <3> и переходит к **Шагу 2**.

В рассмотренных блоках можно писать как любой **PHP** код, так и вовсе оставить все значения пустыми в результате получится бесконечный цикл:

```
for (;;) {}
/* На этом месте программа зависнет, так как будет
выполняться бесконечный цикл */
```

Цикл **foreach**:

```
foreach (<массив> as [<ключ> =>] <значение>) {
   //Тело цикла
}
```

Цикл проходит по каждому элементу массива, и значение каждого элемента кладет в переменную <**значение**>, а ключ в переменную <**ключ**>. Таким образом, можно легко перебрать значения массива и вывести все его ключи и значения, но значения в самом массиве изменить не получится, т.к. в переменных <**ключ**> и <**значение**> хранятся лишь копии.

Для того чтобы можно было изменять значения в самом массиве, перед названием переменной <значение>, необходимо поставить знак взятия адреса "**&**", но при этом необходимо очищать внутри цикла эту временную переменную:

```
foreach (<массив> as &<значение>) {
   //Тело цикла
   unset(<значение>); //Удаляем временную перемен-
ную
}
```

Если этого не сделать то в программе будут возникать всякого рода ошибки, при работе с массивом (<**массив**>).

Команды break и continue в циклах

Команда **break** прерывает выполнение цикла:

```
/* Заметьте что в цикле while и do while мы не мо-
жем оставить условие пустым в отличие от for */
while (true){
    //Выполнится всего 1 раз
    break; //Выход из цикла
}
```

Команда continue, в циклах **while** и **do while** переходит к проверке условия. В цикле **for** перейдет к блоку <3>:

```
for($i = 0;; $i++)
{
      if ($i == 5) continue;
      if ($i == 10) break;
      echo $i." ";
}
```

Программа напечатает следующее: «0 1 2 3 4 6 7 8 9».

В цикле **foreach** переходит к следующему элементу массива:

```
$arr = Array(8, 6, 6, 3, 2, 5);
foreach($arr as $v)
```

```
{
    if ($v == 6) continue;
    echo $v." ";
}
```

Программа напечатает следующее: «8 3 2 5».

Исключения

Исключение – это вид прерывания, вызываемый при возникновении серьезной ошибки во время выполнения программы, которая приводит к невозможности дальнейшей обработки программы ее базового алгоритма.

Вызов исключения:

```
throw new Exception(<строка сообщения>);
```

Обработка исключения

```
try {
    /* Блок кода в котором может возникнуть исключение */
}
catch(Exception $e) {
    /* Блок кода, который выполнится в случае возникновения
       исключения в блоке try */
}
[finally {
    /* Блок кода, который выполнится в любом случае */}]
```

Пример работы:

```
try{
    throw new Exception("Ошибка! в блоке try ");
}
catch(Exception $e)
{
    echo $e->getMessage();
}
finally{
```

```
    echo "конец.";
}
```

В результате выведется: «Ошибка! в блоке try конец».
Код **$e->getMessage()** возвращает текст сообщения.
Создание функций

```
function <имя функции>([<параметры>]){
    //Тело функции
}
```

Функция может принимать параметры и возвращать значение, притом одна функция может вернуть всего одно значение или вообще ничего не вернуть. Для выхода из функции и возврата значения используется команда:

```
return <значение>;
```

Пример:

```
function myMax($a, $b) {
    if ($a > $b) {
        return $a;
    }
    else {
        return $b;
    }
}
echo myMax(10, 432);
```

Результат работы программы: «432».
Инструкции включения
Если необходимо подключить один файл в другой, то можно воспользоваться одной из следующих функций:

- **include "путь к файлу"** – подключает указанный файл в текущее место, в случае если файл будет не найден, то вернет **WARNING** (предупреждение);

- **require ("путь к файлу")** – функция аналогично **include** подключает файл, но в случае если файл будет не найден, то вернет **FATAL ERROR** и выполнение **PHP** кода будет прервано;
- **require_once ("путь к файлу")** – функция действует аналогично **require**, но делает это в случае, если данный файл не был включен раньше, то есть позволяет избежать дублирования включаемых данных.

Наиболее важные PHP функции [2,6,7]

1. isset(<имя переменной>) – проверяет, была ли ранее определена и инициализирована переменная с таким именем, результат выполнения true или **false**. Пример:

```
echo (isset($a) ? "TRUE " : "FALSE ");
$a = 8;
echo (isset($a) ? "TRUE " : "FALSE ");
```

В результате будет выведено на печать: FALSE TRUE.

2. exit([<строка>] || [<число>]) – прекращает работу текущего кода, если в качестве параметра была передана строка, то перед завершением будет выведено сообщение, а если число (0 - 254), то значение не выводится и будет использовано как статус выхода.

Пример:

```
exit;     // нормальное завершение программы
exit();   // нормальное завершение программы
exit(0); // 0 – нормальное завершение программы
exit(1); // завершение программы с ошибкой
exit("В приложении произошла ошибка!");
```

Также есть эквивалентная ей функция **die**().

3. count(<массив>) – возвращает количество элементов в массиве.

4. strlen(<строка>) – возвращает длину строки.

5. time() – возвращает текущее время (количество секунд, прошедших с начала Эпохи Unix (The Unix Epoch, 1 января 1970 00:00:00 GMT) до текущего времени.

6. define(<имя константы>, <значение константы>) – определяет константу в программе.

7. header() – применяется для отправки **HTTP** заголовка. Она должна быть вызвана до первого вывода **HTML** содержимого.

Пример:

```
/* Пересылает пользователя на страницу яндекса */
header("Location: http://yandex.ru/");

/* Устанавливает кодировку и тип файла для текущей
страницы */
header("Content-Type: text/html; charset=utf-8");
```

8. preg_match(<строка шаблон>, <проверяемая строка>) – проверяет строку на соответствие регулярному выражению (шаблону), функция возвращает 1, если параметр <**строка шаблон**> соответствует переданному параметру <**проверяемая строка**>, 0 если нет, или **FALSE** в случае ошибки.

9. md5(<строка>) – возвращает **MD5**-хэш строки.

10. sha1(<строка>) – возвращает **SHA1**-хэш строки.

Примеры работы функций **md5** и **sha1**:

```
$str = "Hello World";
echo md5($str)." <br /> ".sha1($str);
```

В результате получатся две строки вида:

```
b10a8db164e0754105b7a99be72e3fe5
0a4d55a8d778e5022fab701977c5d840bbc486d0
```

11. trim(<строка>) – возвращает строку с удаленными пробелами в начале и в конце строки.

12. in_array(<значение>, <массив> [,<учитывать регистр (true/false)>]) – проверяет, присутствует ли указанное значение в массиве, результатом работы функции будет **true** или **false**.

Следующим этапом, рассмотрим процесс соединения web – страниц с базой данных с помощью языка PHP.

Для соединения с сервером БД можно использовать функцию:

mysqli_connect(<хост>, <имя пользователя>, <пароль>, <имя БД>) – после выполнения функция вернет идентификатор соединения. Для локального компьютера и пользователя по умолчанию параметры будут следующими ("**ocalhost**", "**root**", "", "*<имя БД>*").

После подключения и выбора БД можно отправлять различные запросы и получать данные. Для отправки запроса существует функция:

mysqli_query(<идентификатор соединения>, <запрос>) – в случае неудачного запроса вернет ложь. В случае запросов **SELECT**, **SHOW**, **DESCRIBE**, **EXPLAIN**, функция вернет указатель на результат, который в дальнейшем может быть обработан.

Для работы с результатом запроса существует множество функций, рассмотрим наиболее применяемые функции:

1. mysqli_num_rows(<указатель на результат>) – функция возвращает количество полученных строк.

В случае если необходимо узнать количество строк затронутых запросами (**INSERT**, **UPDATE**, **DELETE**), необходимо использовать **mysqli_affected_rows**().

2. mysqli_fetch_assoc(<указатель на результат>) – функция возвращает ассоциативный массив следующей строки результата запроса. Функция имеет внутренний указатель, так что при вызове первый раз она вернет первую строку, при вызове второй раз, вторую строку и так до тех пор, пока не достигнет конца. В случае если строк больше нет, функция вернет ложь.

3. mysqli_fetch_array(<указатель на результат>) – аналогична функции mysqli_fetch_assoc(), только кроме ассоциативного массива вернет еще и массив с численными индексами.

Для быстрого просмотра содержимого массива удобно применять функцию **print_r(<массив>)**, она выводит структуру массива и его содержимое.

После того как работа с БД завершена, необходимо закрыть соединение, для этого применяется функция:

4. mysqli_close(<идентификатор соединения>) – в случае успешного закрытия соединения вернет истину, в противном случае ложь. Рассмотрим разработку кода простого взаимодействия web – страниц с БД:

```
/* Посылаем заголовок о кодировке текущей страницы
Внимание! заголовки должны посылаться до первого
вывода из файла
Если используется кодировка UTF-8 (как в данном
случае), то ее нужно использовать без BOM */
header('Content-type: text/html; charset=utf-8');

$db = mysqli_connect("localhost", "root", "", " my-
database");
//Устанавливаем кодировку для БД
mysqli_query($db, "SET NAMES utf8");

$data = mysqli_query($db, "SELECT * FROM prod-
ucts");
$rowCount = mysqli_num_rows($data);

echo "Запрос вернул $rowCount строк <br />";

//Если запрос вернул хотябы одну строку
if($rowCount > 0)
{
      //Получаем первую строку
      $row = mysqli_fetch_assoc($data);

      do
      {
            echo "--------------------------------
```

```
----------<br />";
            echo                                "Наименование:
".$row['Наименование']."<br />";
            echo "Описание:  ".$row['Описание']."<br
/>";
            echo                                "Стоимость:
".$row['Стоимость']."<br />";

            //Получаем след строку
            $row = mysqli_fetch_assoc($data);
      }
      while($row);
}
mysqli_close($db);
```

Результат выполнения кода представлен на рис. 2.16

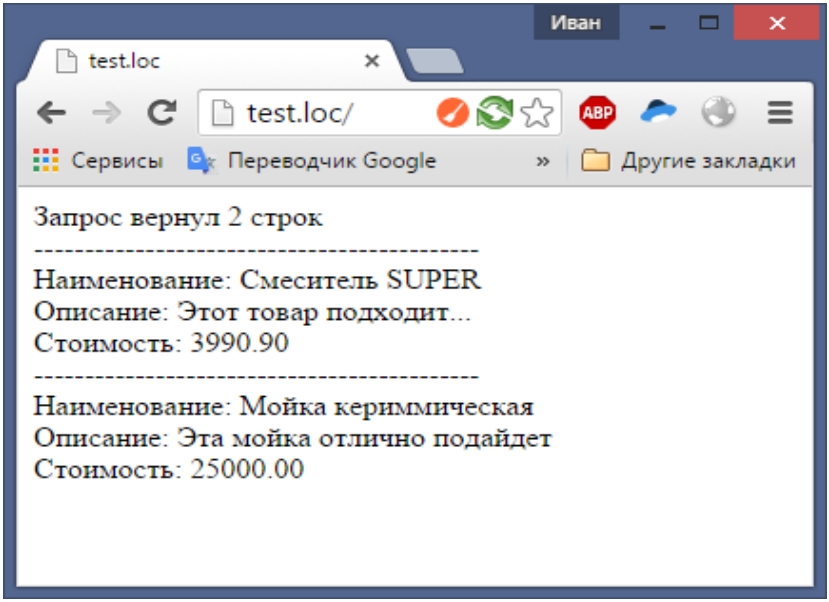

Рис. 2.16. Результат взаимодействия web-страницы с БД

3. РАЗРАБОТКА ИНТЕРНЕТ – МАГАЗИНА ПРОДАЖИ СТРОИТЕЛЬНЫХ МАТЕРИАЛОВ

В данном практикуме рассмотрен пример разработки web-ресурса, который можно отнести к категориям B2B (business-to-business) и B2C (business-to-consumer) и представляет собой интернет-магазин, специализирующийся на продажи строительного материала.

Цель разработки данного проекта – автоматизировать процесс торговли, через сеть Интернет, что позволит привлечь новых клиентов, повысить продажи (соответственно увеличится прибыль), сделать процесс приобретения товара более быстрым и удобным.

Для разработки серверной части и для организации взаимодействия web-ресурса с базой предлагается применить программный комплекс Open Server [14].

Для разработки клиентской части (интерфейса) web-ресурса, предлагается применить тестовый редактор, поддерживающий язык гипертекстовой разметки HTML, CSS и язык JavaScript.

Также необходимо установить операционную систему совместимую свыше перечисленными программными компонентами.

3.1. Анализ предметной области и проектирование базы данных

Прежде, чем приступить к разработке web-ресурса, необходимо провести анализ предметной области, а именно что представляет собой процесс продажи товаров.

Покупатель приходит в магазин, изучает ассортимент товаров и услуг, предоставляемый данным магазином, выбирает интересующие его товары и обращается к продавцу для совершения покупки, который оформляет заказ товаров со склада или витрины.

Таким образом, можно выделить ключевые сущности для разработки базы данных [2,11].

Первая в данном процессе сущность – это покупатель. Для того чтобы отделить информацию о покупках одного клиента от другого, необходимо разграничить клиентам доступ с парольной защитой. Поскольку магазин будет работать удалённо от клиентов, то возникает проблема доставки товара до покупателя, а из этого следует, что магазину нужна информация о месте жительства покупателя, его фамилия, имя, и какая-либо контактная информация, например, электронная почта, которая выступит в качества логина для авторизации, и телефон (табл. 3.1).

Таблица 3.1

Покупатель

№ п/п	**Имя поля**	**Тип поля**
1	E-mail	Текстовый
2	Пароль	Текстовый
3	Имя	Текстовый
4	Фамилия	Текстовый
5	Телефон	Текстовый
6	Почтовый адрес	Текстовый

Следующим, объектом процесса приобретения товара является, ассортимент. В отличие от реального магазина, где можно ознакомиться с ассортиментом, либо изучая витрины, либо консультируясь с продавцом, интернет - магазин дает возможность увидеть ассортимент всего склада. Ассортимент товара на складе, как для работника магазина, так и для покупателя через интернет удобнее будет, просмотреть, если его разместить в определённом порядке, например, разбить и отсортировать по категориям (табл. 3.2).

Следующая сущность – «Корзина». Здесь пользователь магазина имеет возможность добавить весь интересующий его товар в корзину, а затем, просмотрев свой выбор, оформить заказ, купив товар, представленный в корзине (табл. 3.3).

Таблица 3.2

Товар

№ п/п	Имя поля	Тип поля
1	Наименование товара	Текстовый
2	Описание	Текстовый
3	Цена	Денежный
4	Количество товара на складе	Целочисленный
5	Категория	Текстовый

Таблица 3.3

Корзина

№ п/п	Имя поля	Тип поля
1	Товар	Текстовый

На основе выше рассмотренной информации построим инфологическую модель базы данных (рис. 3.1)

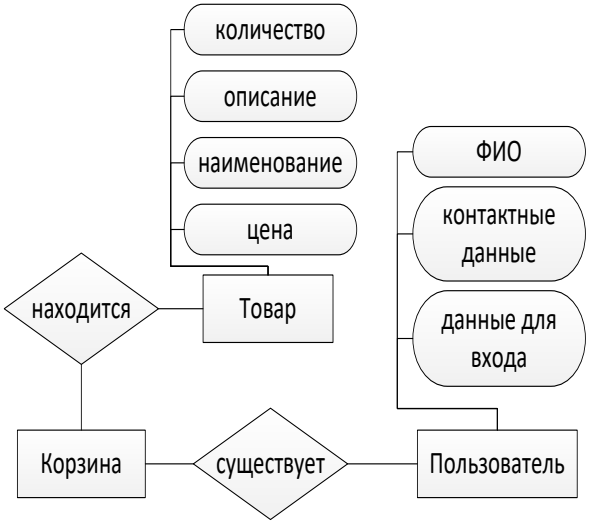

Рис. 3.1. Инфологическая модель базы данных

Логическая модель базы данных представлена на рис. 3.2.

В физической модели (рис. 3.3) выделена таблица «Категории товаров» из таблицы «Товары», так как один и тот же товар может принадлежать к разным категориям.

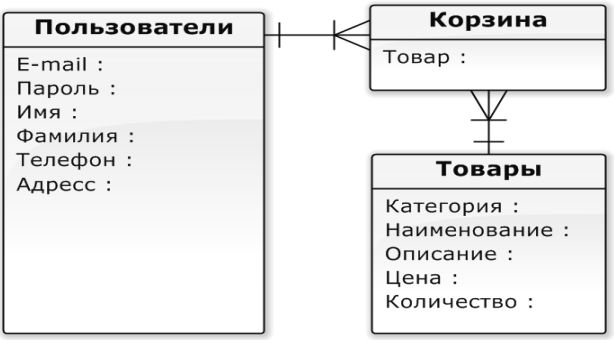

Рис. 3.2 Логическая модель БД

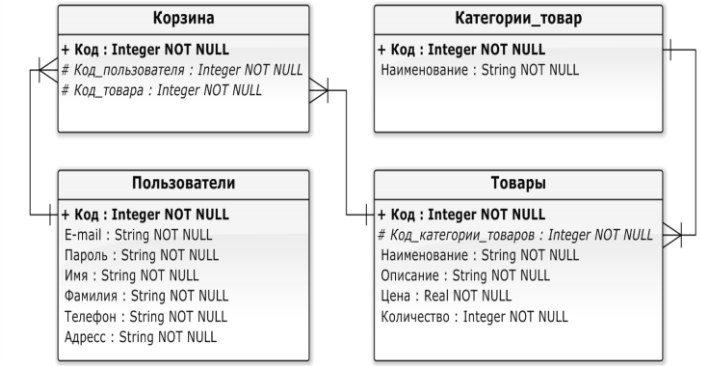

Рис. 3.3 Физическая модель БД

3.2. Разработка структуры web-ресурса

Прежде, чем проектировать интерфейс web-ресурса, необходимо определить его структуру и количество страниц.

Исходя из разработанной инфологической модели, можно выделить следующие страницы: «Каталог товаров», «Стра-

ница пользователя», «Страница управления корзиной», и обязательно страницы «Регистрации» и «Авторизации». Кроме того, как и любому web-ресурс, потребуется «Главная» страница, связующая все остальные страницы воедино.

На главной странице необходимо расположить меню, которое будет осуществлять доступ к любым другим страницам ресурса. Данное меню необходимо так же расположить на каждой страницы web-ресурса, поэтому его необходимо разработать и выделить в отдельный блок, и подключить к остальным страницам.

В рамках данного проекта не будут созданы страницы «Страница пользователя» и «Страница управления корзиной», вместо этого будет выводиться информация о товарах в корзине и данные пользователя из выпадающих блоков в главном меню при помощи языка JavaScript и технологии AJAX [3,15].

Кроме того, в отдельный блок будет выделено меню выбора каталогов товаров, чтобы можно было его разместить как на странице каталога товаров, так и на главной странице.

В случае успешной регистрации пользователя не нужно будет показывать форму для ввода данных, поэтому данную форму тоже выделяем в отдельный блок.

На странице «Каталог товаров» будут выводиться данные товара в виде отдельных блоков для каждого наименования товара, поэтому, этот блок так же выделяется в отдельный блок.

На странице «Каталог товаров» будет выводится список товаров, где каждая запись представляет собой одинаковые по структуре элементы, которые логически можно вынести в отдельный блок и в последствии использовать на других страницах web - ресурса.

На рис. 3.4 приведена структурная схема страниц web - ресурса (выделены жирной границей) и отдельных подключаемых блоков, которые входят в данные страницы (выделены пунктирной границей). Данная схема показывает только лишь те файлы, которые отвечают за интерфейс ресурса.

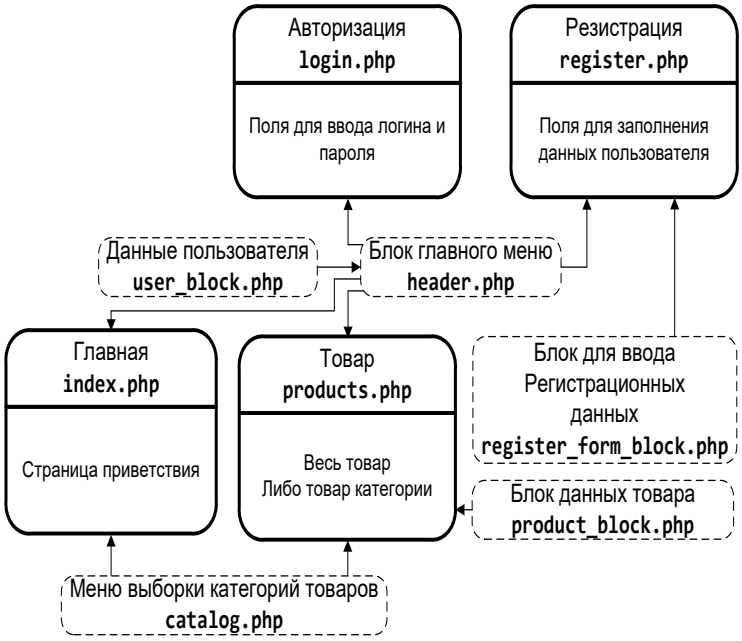

Рис. 3.4. Структура страниц web-ресурса

Определим информационное наполнение страниц web – ресурса.

1) **header.php** – главное меню ресурса: ссылки на все страницы, кнопка выхода пользователя, кнопка регистрации, выпадающие блоки корзины и информации об авторизованном пользователе;

2) **catalog.php** – меню выбора категорий товаров: ссылка на страницу товаров с передачей данных о выборке товаров по категории методом **GET**;

3) **index.php** – главная страница: приветственное сообщение, информация о web – ресурсе;

4) **login.php** – страница авторизации: поле для ввода логина, пароля, кнопка авторизации;

5) **register.php** – страница регистрации: форма для ввода регистрационных данных, сообщение об успешной регистрации, либо об ошибках;

6) **register_form_block.php** – форма для ввода регистрационных данных: поля для ввода логина, пароля, адреса, телефона, фамилии, имени, кнопка регистрации;

7) **products.php** – страница ассортимента товара: блоки каждого вида товара, меню выбора категорий товара, строка поиска товара;

8) **product_block.php** – блок товара: наименование, цена, информация о наличии товара на складе, количество, описание, категории, артикул;

9) **user_block.php** – данные пользователя: фамилия, имя, почта, телефон, адрес, кнопка выхода пользователя.

На рис. 3.5 представлена структурная схема функционального взаимодействия файлов web – ресурса, где серым цветом показаны файлы страниц, бледно-серым – включаемые блоки с **HTML** кодом, а белые блоки с пунктирной границей – файлы с кодом **PHP**, которые и реализуют функциональность web-ресурса вместе с **JavaScript**.

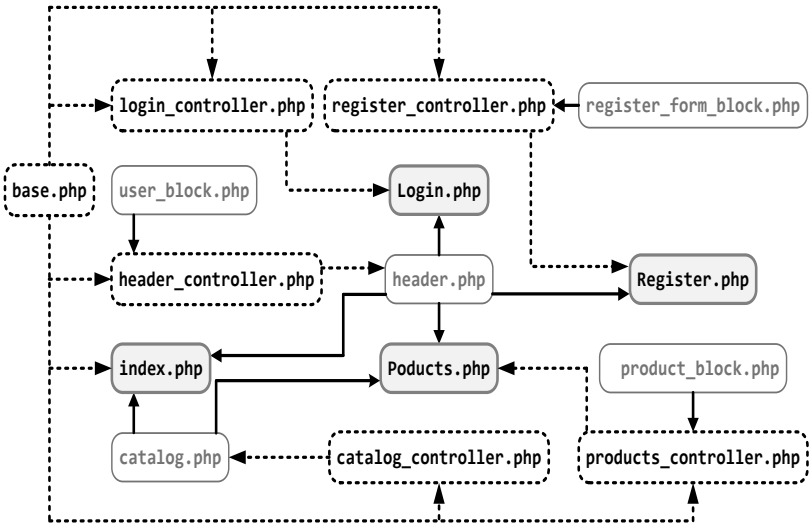

Рис. 3.5. Структурная схема функционального взаимодействия файлов web-ресурса

Схема включения файлов JavaScript для функциональности web-ресурса приведена на рис. 3.6, где файлы с **PHP** и **HTML** кодом изображены блоками аналогичными на рис.3.5.

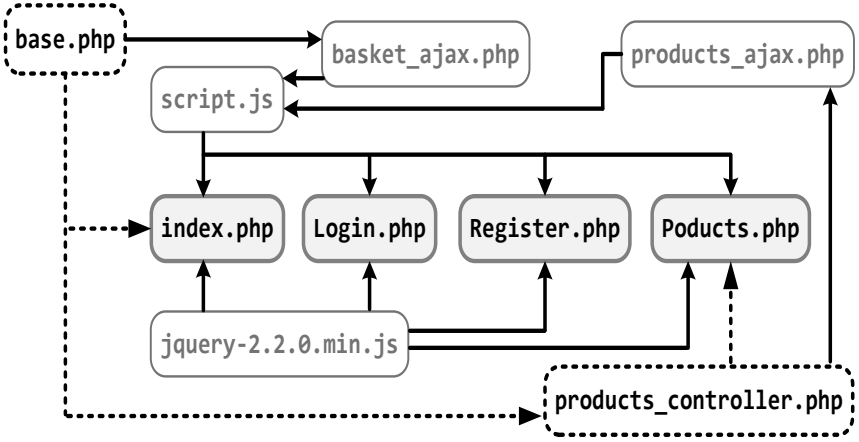

Рис. 3.6. Схема включения файлов JavaScript

Для наглядности реализации серверной логики были разработаны файлы, имеющие имена аналогично основным страницам с припиской «_controller» (postfix), включающие в себя код, позволяющий отделить большую часть серверной логики от интерфейса страниц.

Таким образом, за функциональность web-ресурса отвечают файлы:

- **header_controller.php** – обеспечение динамичности главного меню web-ресурса в зависимости от авторизации пользователя;
- **login_controller.php** – авторизация пользователя;
- **register_controller.php** – регистрация пользователя;

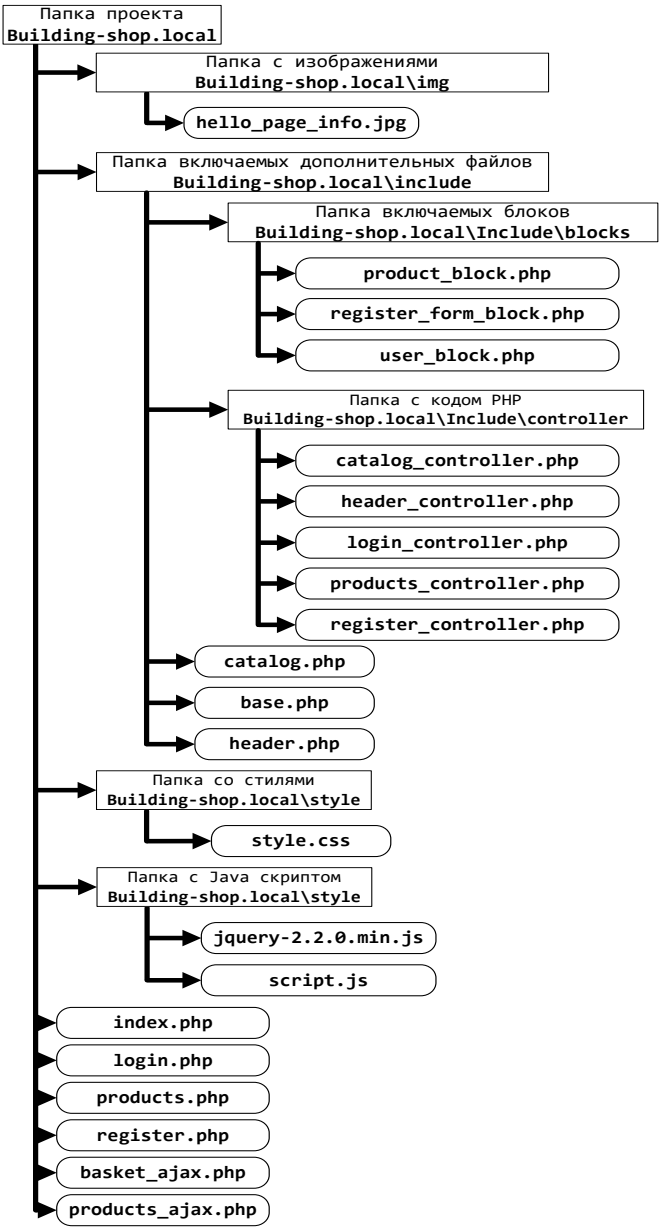

Рис. 3.7. Физическая структура web-ресурса

- **products_controller.php** – вывод списка товаров на складе;
- **catalog_controller.php** – обеспечение динамичности меню выбора категорий товаров (открыто ли меню всегда, или открывается при наведении на заголовок меню курсора мыши);
- **jquery-2.2.0.min.js** – библиотека **jQuery**, предоставляющая, упростить работу с DOM деревом и AJAX запросами;
- **script.js** – обеспечивает логику работы интерфейса (отправка асинхронных запросов на добавление/удаление товаров в корзине, открытие/закрытие главного меню и т.п.);
- **basket_ajax.php** – функции для работы с корзиной;
- **product_ajax.php** – обеспечение поиска товаров.

Так же разработан файл **base.php**, включающий в себя код для работы с базой данных и набор базовых функций для работы с сервером и применяемый для работы всех страниц web – ресурса.

Физическая структура web-ресурса представлена на рис. 3.7.

3.3. Макетирование и дизайн страниц web-ресурса

Следующим этапом разработки проекта является дизайн и макетирование страниц web-ресурса в соответствии с техническим заданием. Дизайн разрабатывается обычно с помощью графических пакетов. В данной работе внимание уделено именно реализации web-страниц, поэтому дизайн будет самым простым. На основе разработанного дизайна и макета страниц осуществляется их реализация при помощи **HTML** и **CSS [3,4,5]**. На рис. 3.8 и 3.9 представлены макеты «Главной страницы» и «Каталог товаров».

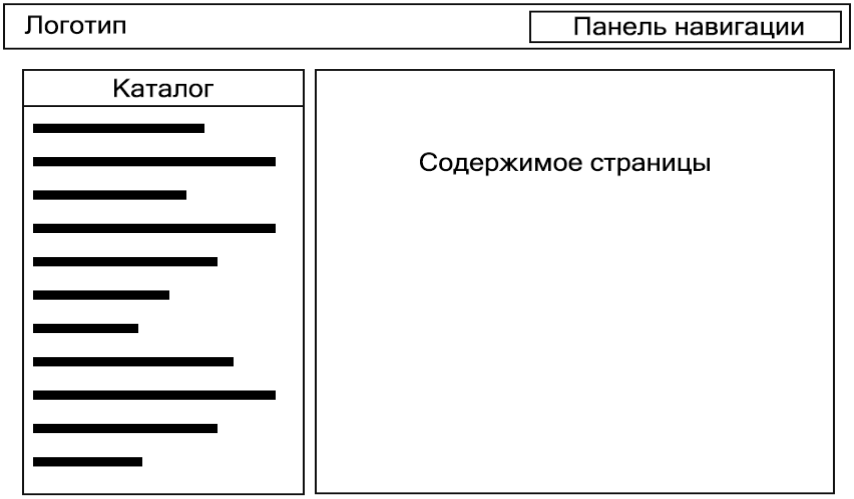

Рис. 3.8. Макет «Главной страницы»

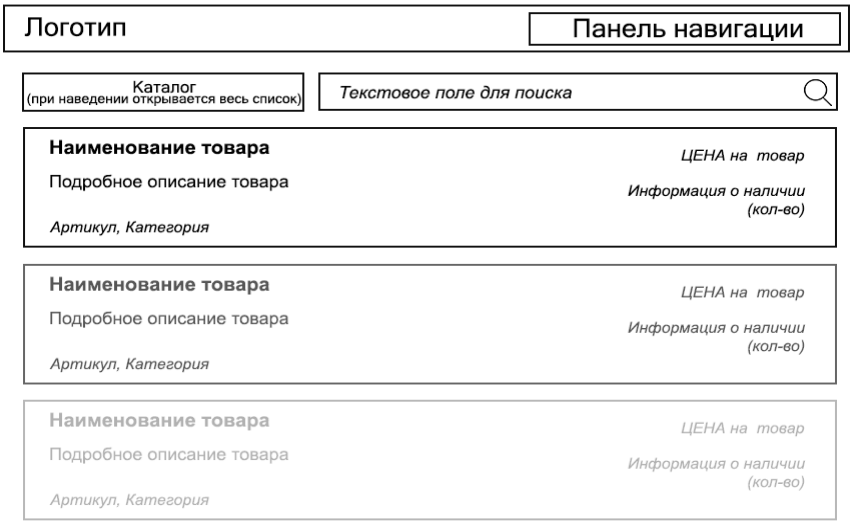

Рис. 3.9. Макет «Каталог товаров

3.4. Реализация web-ресурса

Для удобства реализации проекта, как выше уже было отмечено, применяется инструментальное средство **OpenServer,** которое избавляет от установки и настройки необходимого серверного ПО (web-сервера, MySQL, PHP). Необходимо перейти в папку с установленным сервером **OpenServer**'ом, затем в **domains** создать папку с проектом «**building-shop.local**» (в данном случае). Постфикс «**.local**» предназначен для логического обозначения, о том что web-ресурс находится на локальном сервере. Теперь, в папке с проектом создается файл «**index.php**». Это главная страница, и сервер будет автоматически запускать файл с этим названием. **Обязательно для каждого файла проекта необходимо указать кодировку UTF-8 без BOM** для корректного чтения сервером файлов проекта. Файлы проекта можно редактировать в любом текстовом редакторе, поддерживающем данную кодировку [2,14].

3.4.1 Разработка web-страниц

Разработка главной страницы index.php

Дизайн главной страницы представлен на рис. 3.10. Блок основного меню размещен сверху страницы. Причем, в зависимости от статуса пользователя, ему будут доступны, разные страницы и блоки главного меню (табл. 3.4). Блок списка категорий товара расположен сбоку – слева.

Таблица 3.4

Распределение доступа страниц в зависимости от состояния авторизации пользователя

Пользователь авторизован	**Пользователь не авторизован**
Главная	Главная
Товары	Товары
Пользователь	Регистрация
Корзина	Авторизация

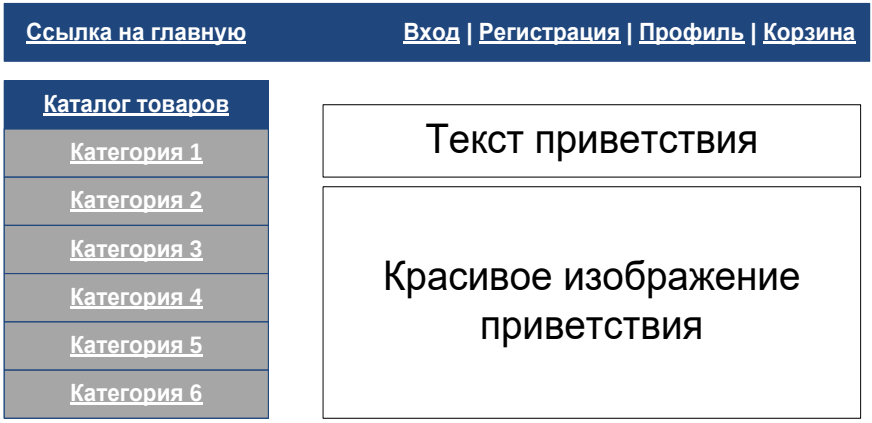

Рис. 3.10. Макет главной страницы

В соответствии с разработанным макетом главной страницы пишется **html** код (листинг 3.1).

Листинг 3.1 – HTML код index.php

```
    <!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <link       rel="stylesheet"       type="text/css"
href="style/style.css" />
  <script                      type="text/javascript"
src="/script/jquery-2.2.0.min.js"></script>
  <script                      type="text/javascript"
src="/script/script.js"></script>
  <title>Главная</title>
</head>

<body>
  <!-- Подключаем блок основного меню-->
  <?php require("include/header.php"); ?>
  <!-- Основной блок сайта-->
  <div class="wrapper">

    <!-- Левый блок сайта-->
    <div class="left_bar">
```

```
      <?php
        //Подключаем блок категорий товаров
        require("include/catalog.php");
      ?>
    </div>
    <!-- Основная информация главного окна-->
    <div class="content_lb">
            <!-- Приветственный текст-->
      <h1 class="h_title">Добро Пожаловать!</h1>
      <p>
        BuildingShop - международная компания-
ритейлер, специализирующаяся на продаже товаров для
строительства, отделки и обустройства дома, дачи и
сада. BuildingShop помогает людям во всем мире бла-
гоустроить жилье и улучшить качество жизни.
      </p>
      <!-- Приветственное изображение-->
      <img                          class="img_center"
src="/img/hello_page_info.jpg" />
    </div>
  </div>
</body>
</html>
```

Обратите внимание, что **CSS** код прописывается в отдельном файле **style.css**, и размещен в папке **styles [8,10].**

Для файлов **header.php** и **catalog.php**, где хранится код блока основного меню проекта и блок меню выбора категорий товаров соответственно, расположены в папке **include**, т.к. они не являются полноценными web- страницами, а являются лишь включаемыми в них блоками, соответственно листинг 3.2 и листинг 3.3.

Листинг 3.2 – HTML код header.php

```
<div class="header">
  <div class="header_content">
    <!-- Название магазина и значок-->
    <a href="/" class="logotip">
      ⚒ BuildingShop
    </a>
    <!-- Верхний блок сайта-->
```

```
    <div class="top_menu">
      <div class="user_top_block">
        <!-- Специальный "значок корзины"-->
        <div class="user_avatar_min">⊛</div>

        <div class="user_block_content">
          <div
class="user_block_name">Корзина</div>
          <!-- Здесь будет информация о пользовате-
ле-->
          <div            class="hidden_user_block"
id="userBasket">
            <!-- Данный блок будет использован поз-
же в "script.js" -->
          </div>
        </div>
      </div>
      <!-- Ссылки на страницы входа и регистрации--
>
      <div class="user_top_block">
        <!-- Специальный "значок настроек"-->
        <div class="user_avatar_min">⊙</div>
        <!-- Информация о пользователе-->
        <div class="user_block_content">
          <!—Позже здесь будут ссылки на регистра-
цию или авторизацию-->
        </div>
      </div>
    </div>
  </div>
</div>
```

Значок ⊙ - это специальный юникод символ, который позволяет приукрасить панель главного меню.

Поскольку на данном этапе программируется только дизайн, да и с базой данных пока нет увязки, то в главном меню нет реализации ссылок на страницу авторизации или регистрации в зависимости от того, авторизован пользователь, или нет.

Код для меню категорий товаров в **catalog.php** представлен в листинге 3.3, где прописана ссылка на страницу то-

варов **products.php**, а не в главном меню **header.php**, т.к. планируется сделать выпадающий список категорий товаров при наведении указателя мыши на ссылку каталога товаров.

Листинг 3.3 – HTML код catalog.php

```
<div class="catalog">
      <!-- Ссылка на страницу каталога товаров-->
      <div class="title">
            <a          href="/products.php">Каталог
товаров</a>
      </div>
      <!-- Скрытый блок категорий товаров-->
      <div class="hidden_box">
            <div class="catalog_list">
                <!--Здесь будет выведен список ка-
тегорий, когда подключится БД-->
            </div>
      </div>
</div>
```

Результат разработанной главной страницы представлен на рис. 3.11.

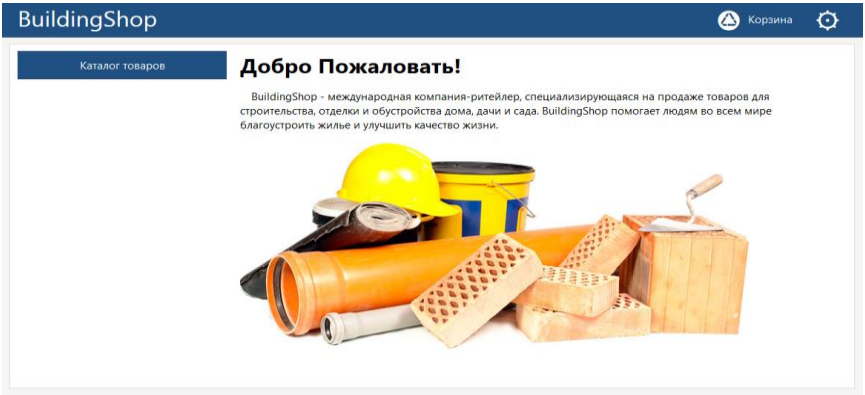

Рис. 3.11. Вид главной страницы

Разработка страницы авторизации login.php

Дизайн страницы авторизации пользователя представлен на рис. 3.12. На этой странице не будет ссылки на каталог товаров и на категории товаров, а это значит, что файл **catalog.php** не будет включен в **HTML** код этой страницы.

Для авторизации необходимы два поля для ввода логина (в данном случае электронная почта) и пароля, а также кнопка для подтверждения данных и отправки на обработку серверу методом **POST**.

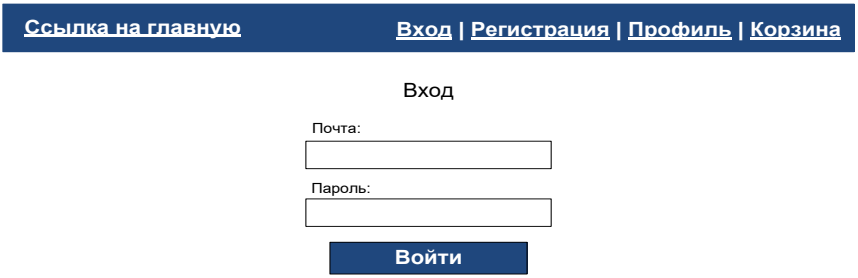

Рис. 3.12. Макет страницы авторизации

HTML код для **login.php** представлен в листинге 3.4.

Листинг 3.4 – HTML код login.php

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <link       rel="stylesheet"       type="text/css"
href="style/style.css" />
  <script                          type="text/javascript"
src="/script/jquery-2.2.0.min.js"></script>
  <script                          type="text/javascript"
src="/script/script.js"></script>

  <title>Вход</title>
</head>
<body>
  <!-- Подключим главное меню-->
  <?php require("include/header.php"); ?>
  <div class="wrapper">
```

```
    <!-- Форма для ввода данных-->
    <div class="modal_page_box">
      <form action="" method="POST">
        <h1>Вход</h1>
        <!-- Ввод почты-->
        <div class="input_box">
        <label for="eMail">Почта:</label><br />
          <input type="text" id="eMail"
placeholder="example@example.com" name="eMail" />
        </div>
        <!-- Ввод пароля-->
        <div class="input_box">
<label for="password">Пароль: </label><br />
   <input type="password" id="password" placehold-
er="&bull;&bull;&bull;&bull;&bull;&bull;&bull;&bull
;" name="password" />
        </div>
 <!-- Кнопка входа-->
        <div class="button_box">
          <button type="submit">Войти</button>
        </div>
      </form>
    </div>
  </div>
</body>
</html>
```

Результат интерпретации данного кода представлен на рис. 3.13

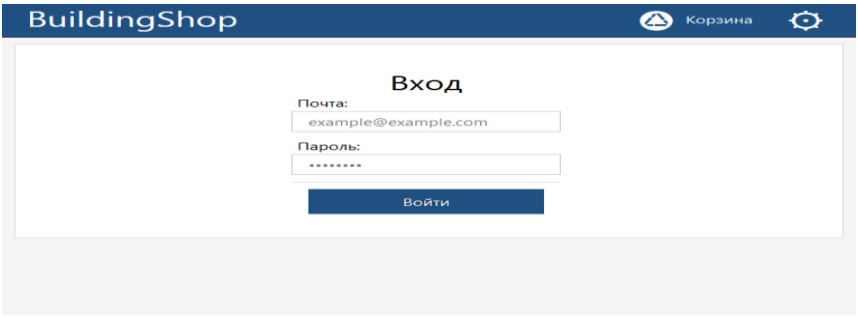

Рис. 3.13. Окно страницы авторизации пользователя

Разработка страницы регистрации пользователя register.php

Дизайн страницы регистрации пользователя представлен на рис.3.14. На этой странице не будет блока с ссылкой на каталог товаров и списка категорий. На ней необходимо поместить поля, которые отвечают за ввод данных о клиенте (пользователя) web – ресурса: логин (электронную почту), пароль, фамилию, имя, телефон и адрес, а так же кнопку подтверждения информации (методом отправки данных POST). HTML код страницы регистрации представлен в листинге 3.5. Код отдельного блока с формой для ввода данных представлен в листинге 3.6.

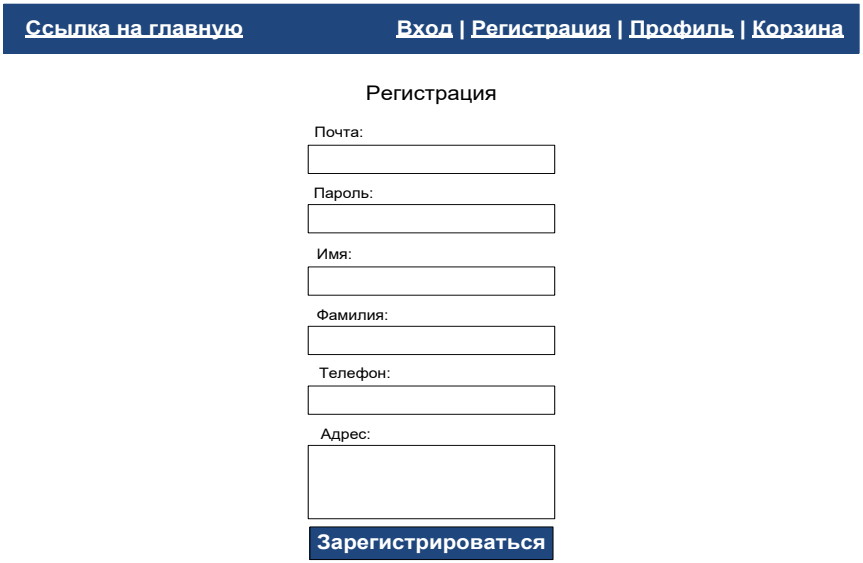

Рис. 3.14. Макет страницы регистрации

Листинг 3.5 - HTML код страницы register.php

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
```

```
  <link      rel="stylesheet"      type="text/css"
href="style/style.css" />
  <script                      type="text/javascript"
src="/script/jquery-2.2.0.min.js"></script>
  <script                      type="text/javascript"
src="/script/script.js"></script>
      <title>Регистрация</title>
</head>
<body>
      <!-- Подключаем главное меню-->
      <?php require("include/header.php"); ?>
      <!-- Форма для ввода данных-->
      <div class="wrapper">
            <!-- Позже мы сделаем при помощи PHP
динамику этого блока-->
            <?php
                  include
("/include/blocks/register_form_block.php");
            ?>
      </div>
</body>
</html>
```

Листинг 3.6 – Блок формы регистрации register_form_block.php

```
<div class="modal_page_box">
  <form action="" method="POST">
    <h1>Регистрация</h1>
    <div class="input_box">
      <label for="eMail">Почта:</label><br />
      <input   type="text"   id="eMail"   placehold-
er="example@example.com"
                  name="eMail"                    <?php
if(isset($_POST['eMail']))
            echo    'value="'.$_POST['eMail'].'"';?>
/>
    </div>
    <div class="input_box">
   <label for="password">Пароль:</label><br />
      <input  type="password"  id="password"  place-
hold-
er="&bull;&bull;&bull;&bull;&bull;&bull;&bull;&bull
;" name="password" <?php
```

```
if(isset($_POST['password']))
                                    echo
'value="'.$_POST['password'].'"';?> />
    </div>
    <div class="input_box">
      <label for="name">Имя:</label><br />
      <input          type="text"          id="name"
placeholder="Иван" name="name"
        <?php     if(isset($_POST['name']))     echo
'value="'.$_POST['name'].'"';?> />
    </div>
    <div class="input_box">
      <label for="lastname">Фамилия:</label><br />
      <input  type="text"  id="lastname"  placehold-
er="Иванов" name="lastname"
                              <?php
if(isset($_POST['lastname']))
                                    echo
'value="'.$_POST['lastname'].'"';?> />
    </div>
    <div class="input_box">
     <label for="phone">Телефон:</label><br />
      <input type="text" id="phone" placeholder="+7
XXX-XX-XX" name="phone"
        <?php    if(isset($_POST['phone']))    echo
'value="'.$_POST['phone'].'"';?>/>
    </div>
    <div class="input_box">
     <label for="address">Адрес:</label><br />
      <textarea name="address" id="address"
        placeholder= "Воронежская обл. г. Воронеж
ул. Выдуманная д. XX кв. XX">
        <?php   if(isset($_POST['address']))   echo
$_POST['address'];?></textarea>
    </div>
    <div class="button_box">
      <button
type="submit">Зарегистрироваться</button>
    </div>
  </form>
</div>
```

! Необходимо обратить внимание на подчёркнутый код в конструкциях типа:

```
    <input type="text" id="name" placeholder="Иван" name="name"
    <?php if(isset($_POST['name'])) echo 'value="'.$_POST['name'].'"';?> /></div>
```

Это **PHP** вставки, позволяющие при нажатии кнопки «Зарегистрироваться» оставить введенные в поля с такими же вставками данные. Обычно, при нажатии кнопки, выполняющей запрос на сервер методом **POST** или **GET** web - страница обновляется, то и в данном случае, после нажатия на кнопку «Зарегистрироваться», данные из всех полей должны быть переписаны на данные по умолчанию. При нажатии на реализованную кнопку «Зарегистрироваться» формируется массив **_POST**, состоящий из данных заполненных полей. Однако, в данном случае **PHP** код проверяет, присутствуют ли в массиве **_POST** данные соответствующего поля, в представленном отрывке кода это – **name**, то **PHP** запишет значения этих данные в нужные поля.

Результат интерпретации кода представлена на рис. 3.15.

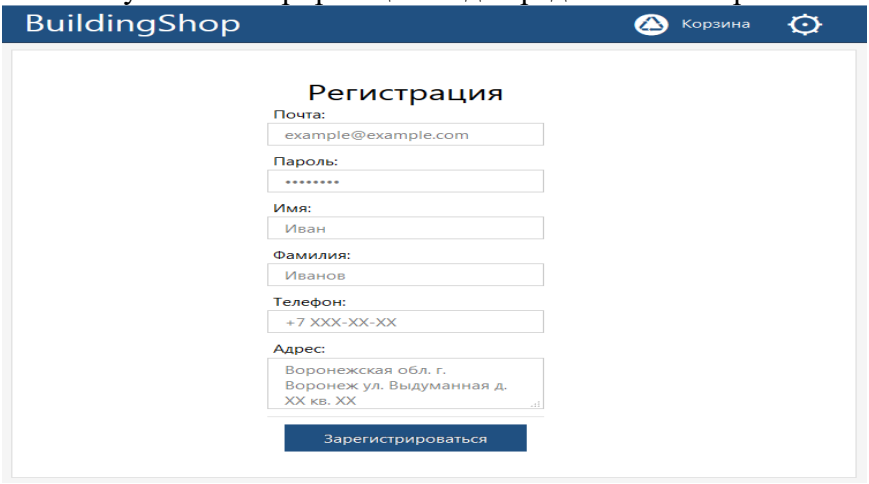

Рис. 3.15. Заполненная форма страницы регистрации

Разработка страницы списка товаров products.php

Дизайн страницы списка товаров представлен на рис. 3.16. Для страницы списка товаров помимо блока основного меню header.php необходимо включить ещё и список категорий товара catalog.php. Однако, данная страница будет иметь раскрывающийся список категорий товаров при наведении курсора мыши на ссылку перехода к странице списка товаров products.php. На данном этапе реализуется дизайн к такой функции, а сама функция будет разработана позже, после подключения базы данных. Итак, на этой странице необходимо поместить список товаров из базы данных, поле для ввода текста поиска и кнопку поиска товара.

HTML код страницы products.php представлен в листинге 3.7.

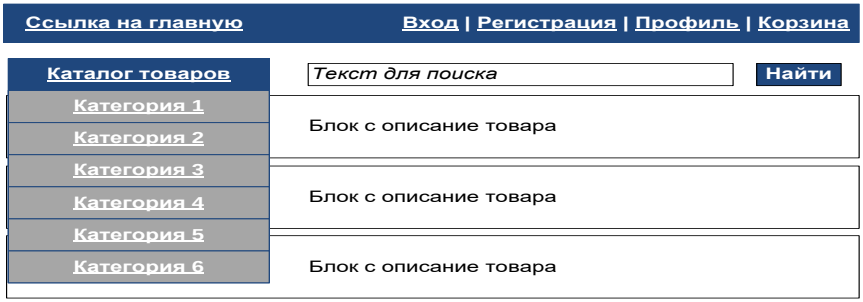

Рис. 3.16. Макет страницы каталога товаров

Листинг 3.7 – HTML код страницы products.php

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <link       rel="stylesheet"       type="text/css"
href="style/style.css" />
  <script                      type="text/javascript"
src="/script/jquery-2.2.0.min.js"></script>
  <script                      type="text/javascript"
src="/script/script.js"></script>
  <title>Каталог</title>
```

```
</head>
<body>
  <!-- Подключаем главное меню-->
  <?php require("include/header.php"); ?>
  <div class="wrapper">
    <div class="top_content_bar">
      <div class="left_bar">
      <!-- Подключаем меню каталога товаров-->
      <?php require("include/catalog.php"); ?>
      </div>
      <!-- Строка поиска-->
      <div class="search_block">
        <div class="serch_form">
          <input                    class="input_text"
placeholder="Введите текст для поиска"
      type="text" name="text" />
          <!—Значок поиска-->
          <button
class="serch_button">&#128269;</button>
        </div>
      </div>
    </div>
    <div class="content">
<!-- Здесь при помощи PHP будет выведен список то-
варов-->
    </div>
  </div>
</body>
</html>
```

В качестве текста для кнопки поиска применен код специального символа 🔍 (&#128269). Результат интерпретации кода представлена на рис. 3.17.

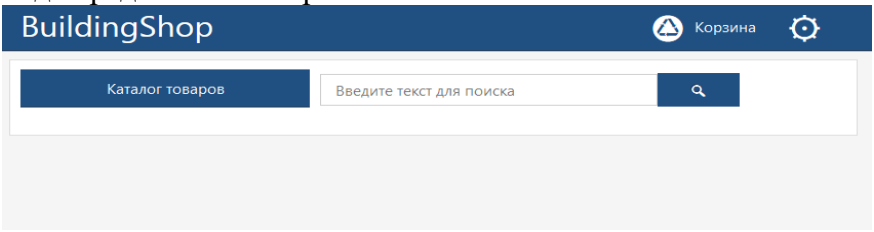

Рис. 3.17. Окно страницы каталога товаров

3.4.2 Процесс подключения базы данных к web – страницам

Следующим этапом разработки проекта является подключение базы данных к web - страницам.

Необходимо создать в папке **building-shop.local\include** дополнительный подключаемый файл **base.php**, содержащий код для работы с базой данных, и который необходимо включить в каждую страницу, разрабатываемого проекта.

В листинге 3.8 приведен код файла **base.php** и приведены пояснения по подключению базы данных.

Листинг 3.8– HTML код файла base.php

```
<?php
  //Посылаем заголовок с тек кодировкой
  header('Content-Type: text/html; charset=utf-8');
  //=== Настройки подключение к БД
  define("DB_HOST", "localhost");
  define("DB_USER", "root");
  define("DB_PASSWORD", "");
  define("DB_DATABASE_NAME", "building_shop");
  //=== Красивое сообщение об ошибке
  function errorMassage($message)
  {
    echo '<div style="background-color: #fdd; pad-
ding: 20px; margin: 50px 20%; font-family: sans-
serif;">';
    echo $message;
    echo '</div>';
  }
  //=== Соединение с БД
  //Соединяемся с БД
  $db = @mysqli_connect(DB_HOST, DB_USER,
DB_PASSWORD, DB_DATABASE_NAME); //@ - говорит о
том, чтобы не выводились предупреждения
  //Проверяем удалось ли соединиться
  if (!$db)
  {
    errorMassage("MySQL сервер недоступен!" .
mysqli_connect_error());
    exit();
```

```
  }
  //=== Возвращает массив строк по запросу
  function GetData($queryString)
  {
    global $db; // Берем переменную из глобальной
облсати видимости

    //--- Запрос к БД
    $q = mysqli_query($db, $queryString);
    //Проверка запроса на ошибки
    if($q == false)
    {
      errorMassage('<b>Произошла ошибка во время
запроса:</b> <br />'.mysqli_error());
      exit(); //Завершаем выполнение скрипта
    }
    //--- Обработка полученных строк
    //Массив будет хранить строки полученные из БД
    $rows = Array();
    //Получаем первую строку
    $row = mysqli_fetch_assoc($q);
    //Если запрос не вернул строк
    if($row == null || mysqli_num_rows($q) == 0)
    {
      //То возвращаем ложь
      return false;
    }
    //Получаем все остальные строки
    do
    {
      //Добавляем строку в массив строк
      $rows[] = $row;

      //Получаем след строку из запроса
      $row = mysqli_fetch_assoc($q);
    }
    while ($row != false);
    //Возвращаем массив строк
    return $rows;
  }
  //=== Функция проверяющая есть ли пользователь в
системе
  function isUser()
  {
```

```
    //Если у пользователя существуют куки почты и
пароля
    if(isset($_COOKIE["mail"]) && is-
set($_COOKIE["p40"]))
    {
      //Обрабатываем данные пришедшие от пользова-
теля
      $eMail = htmlspecialchars($_COOKIE["mail"]);
      $password = sha1($_COOKIE["p40"]);
      //Делаем запрос БД на вход тек пользователя
      $user = GetData("SELECT `id`, `e-mail`,
`name`, `lastname`, `phone`, `address` FROM users
WHERE `e-mail` = '$eMail' AND `password` =
'$password';");
      //Если не нашли пользователя, то выходим
      if($user === false)
      {
        //Удаляем cookie
        setcookie("mail", "", time() - 3600);
        setcookie("p40", "", time() - 3600);

        //Говорим что пользователь не найден или не
авторизован
        return false;
      }
      //Возвращаем данные о пользователе
      return $user[0];
    }
    //Говорим что пользователь не найден или не ав-
торизован
    return false;
  }
  //Получаем статус пользователя (в системе или
нет)
  $user = isUser();
  //Авторизован ли пользователь
  $isUser = ($user === false) ? false : true;

  //=== Функция выводящая список ошибок
  function PrintErrorMessage($errors)
  {
    //Если список ошибок не пуст
    if(count($errors) > 0)
    {
```

```
      echo '<div class="message_error_box">';
      foreach($errors as $message)
      {
        echo '<span>❯ '.$message."</span><br />";
      }
      echo '</div>';
    }
  }
  //=== Функция для запроса добавления, изменения и
удаления
  function ChangeData($q, $message, $exit = false)
  {
    global $db;

    try
    {
      // Делаем запрос
      $is = mysqli_query($db, $q);

      // Если что то пошло не так
      if($is == false)
      {
        //Выбрасываем исключение
        throw new Exception();
      }
    }
    //Если при произошла непридвиденная ошибка
    catch(Exception $e)
    {
      echo $message;
      if ($exit == true)
      {
        exit();
      }
    }
  }
?>
```

После подключения базы данных необходимо настроить функциональность web–ресурса через взаимодействие с ней.

3.4.3 Процесс реализации серверной логики на web-страницах

Для страницы главного меню **header.php** необходимо реализовать функцию, определяющую статус пользователя в системе: если пользователя в системе нет, в меню должны выводиться ссылки на страницы авторизации и регистрации пользователя, если же пользователь уже авторизован, то должна появляться ссылка на профиль пользователя и кнопка выхода.

Для этого в папке **building-shop.local/include/controls** создается файл **header_controller.php** (листинг 3.9).

Листинг 3.9 – Код файла header_controller.php

```
<?php
//Подключается базовый код соединения с БД
  require_once("include/base.php");
  //Функция,выводящая блок пользователя
  function UserBlock($isUser, $user)
  {
    //Если пользователь вошел
    if($isUser == true)
    {
      //подключается код блока
      include ("include/blocks/user_block.php");
    }
    //Если пользователь не вошел, выводятся ссылки
на регистрацию и авторизацию.
    else
    {
      echo '<a href="/login.php">Вход</a> | ';
      echo                                    '<a
href="/register.php">Регистрация</a>';
    }
  }
?>
```

В данном коде применяется переменная $isUser, созданная в файле base.php. Если пользователь в сети, то подключа-

ется дополнительный файл **user_block.php** с **HTML** и **PHP** кодом, выводящим данные о пользователе (листинг 3.10).

Листинг 3.10 – Код файла user_block.php

```
<?php
//Проверяется, подключения пользователя напрямую к
данному файлу, как к самостоятельной странице.
  if (isset($user) == false)
  {
    exit();
  }
?>
<!-- Пишется имя пользователя в названии блока в
главном меню -->
<div class="user_block_name" id="userBlock">
  <b><?=$user['name']?></b>
</div>
<!-- Создается скрытый блок с данными пользователя
-->
<div class="hidden_user_block">
  <div class="user_name">
    <?php            echo            $user["name"]."
".$user["lastname"];?>
  </div>
  <div class="user_dop_info">
    <span>Почта: </span>
    <?php echo $user["e-mail"];?>
  </div>
  <div class="user_dop_info">
    <span>Тел: </span>
    <?php echo $user["phone"];?>
  </div>
  <div class="user_dop_info">
    <span>Адрес: </span>
    <?php echo $user["address"];?>
  </div>
  <hr />
<!-- Добавляется кнопка для выхода пользователя-->
  <a                    href="/login.php?exit=true"
class="button">Выйти</a>
</div>
```

В коде листинга 3.10 имя пользователя получается из строки данных пользователя **$user** по ключу **"name"**. Кнопка выхода создается в виде ссылки. Адресом ссылки является страница авторизации /**login.php** ("/" подразумевает относительные ссылки). При переадресации на эту страницу передаётся переменной **exit** значении истины методом **GET** (через адресную строку). Позже реализуется выход пользователя с использованием этой переменной. Если пользователь в системе не авторизован, выводятся ссылки на авторизацию и регистрацию.

Блок **hidden_user_block** с данными пользователя прописывается «скрытым» при помощи **CSS** кода, а отображается с применением **JavaScript кода**.

Теперь внутри блока:

```
<div class="user_block_content">
</div>
```

файла **header.php** необходимо поместить следующий код:

```
<?php UserBlock($isUser, $user); ?>
```

Кроме того, неообходимо включить **header_controller.php** в начало файла **header.php**:

```
<?php require_once("include/controller/ header-
er_controller.php"); ?>
```

Задача файла **catalog.php** заключается в необходимости выбрать из базы данных все категории товаров и вывести их в раскрывающемся блоке в единое меню под ссылкой на страницу каталога товаров **products.php**. Для этого в папке **building-shop/include/controls** создается файл **catalog_controller.php** (листинг 3.11).

Листинг 3.11 – Код файла catalog_controller.php

```
<?php
  //Подключается базовый код соединения с БД
  require_once("include/base.php");
  //Если существует параметр открытия, то добавляется дополнительный класс
  function CategoriesOpenFunc($parametr)
  {
    //Если был создан параметр открытия и он равен истине
    if(isset($parametr) && $parametr == true)
    {
      //То добавляем класс open, чтобы меню отображалось всегда
      echo " open";
    }
  }
  //Выводит список категорий
  function PrintListCategories()
  {
    //Получаем список категорий
    $categories = GetData("SELECT id, title FROM product_categories;");
    //Если список категорий не пуст
    if ($categories != false)
    {
      //То он выводится
      foreach($categories as $cat)
      {
//Если методом GET была передана строка поиска
        if (isset($_GET["text"]) && $_GET["text"] != "")
        {
//То добавляется к ссылке значение искомой строки
          echo
'<a
href="/products.php?cat='.$cat['id'].'&text='.$_GET["text"]
.'">'.$cat['title'].'</a>';
        }
        //Иначе
        else
        {
```

```
//Выводится ссылка с номером категории
          echo                                    '<a
href="/products.php?cat='.$cat['id']
.'">'.$cat['title'].'</a>';
        }
      }
    }
  }
?>
```

В листинге 3.11 функция **CategoriesOpenFunc()** отвечает за то, будет ли меню категорий показываться всегда, как это планировалось для главной страницы, либо будет раскрываться при наведении курсора мыши на заголовок данного меню, как на странице товаров.

Рассмотрим части **CSS** кода из **style.css.**

```
.catalog:hover .catalog_list,
.catalog .catalog_list.open{
  opacity: 1;
  visibility: visible;
}
```

Здесь, для класса **catalog** прописывается псевдокласс **hover**, который «говорит», что нижеследующий код, относящийся к этому псевдоклассу, применяется только в том случае, если указатель мыши наведён на блок с классом **catalog**. Кроме того, **CSS** код позволяет определять блок стилей для разных классов, просто перечислив их через запятую. Для этого определяем его как стиль по умолчанию для класса **catalog_list.open**. То есть, если в файл **catalog.php** передаётся переменная **isCategoriesOpen** со значением **true**, то класс блока меню категорий товаров именуется как **catalog_list.open**, а значит, будет открыт в любом случае. Если же **isCategoriesOpen = false**, то класс блока будет называться **catalog,** и отображение его будет зависеть от наведения курсора мыши (**hover**). Переменная **isCategoriesOpen** передаётся

в качестве параметра функции **CategoriesOpenFunc($parametr)**. Вместо кода:

```
<div class="catalog_list">
    <!--Здесь будет выведен список категорий, когда подключим БД-->
</div>
```

файла **catalog.php** пишется следующий код:

```
<div    class="catalog_list<?php    CategoriesOpen-Func($isCategoriesOpen); ?>">
    <?php PrintListCategories(); ?>
</div>
```

Также в самом начале данного файла подключается **catalog_controller.php** строкой кода:

```
<?php                                                    require_once("include/controller/catalog_controller.php"); ?>
```

Необходимо включить файл base.php в index.php, т.к. функции из этого файла будут применяться на главной странице. Для подключения **base.php** в файл **index.php** необходимо прописать в начало, следующий код:

```
<?php require_once("include/base.php"); ?>
```

Продемонстрировать страницу, где в главном меню сайта будет выпадающий блок данных пользователя и кнопка выхода, на данный момент не получиться, т.к. страница авторизации пользователя ещё не связана с базой данных, а значит, и авторизоваться пользователь не может (рис. 3.18).

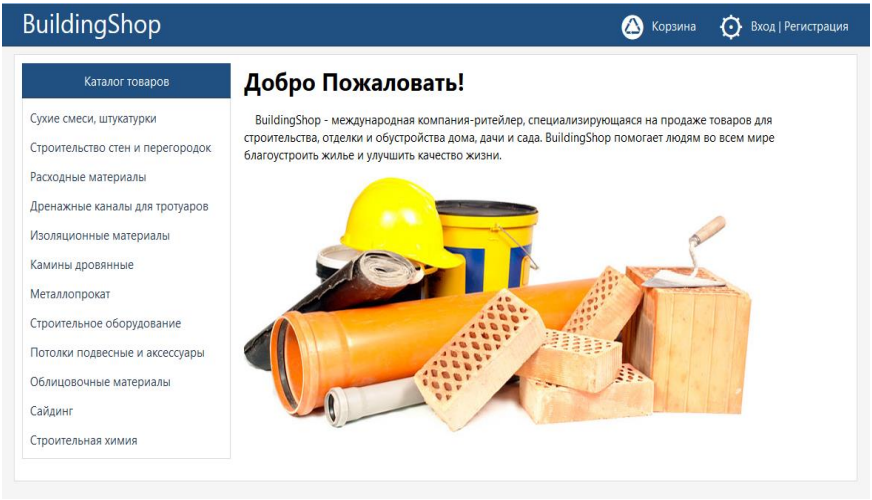

Рис. 3.18. Вид главной страницы

На страницу авторизации пользователя **login.php** необходимо добавить подключение базы данных, реализованное во включаемом файле **base.php**. Кроме того, надо реализовать следующие функции:

1. Если пользователь попал на страницу авторизации, то его необходимо перенаправить на страницу профиля пользователя, даже если пользователь уже в системе, так как данная страница авторизации **login.php** не будет доступна в главном меню. Помимо этого пользователь может ухитриться набрать адрес страницы вручную, либо перейти во вкладку браузера, где уже открыта эта страница, и обновить её. Так же это позволит избежать авторизации несколько пользователей в системе, либо авторизации одного пользователя, но несколько раз.

2. Проверку введённых данных в поля для логина и пароля пользователя методом **POST** и, если такой пользователь найден в базе данных, авторизовать его и отправить на страницу пользователя.

3. Проверку, на наличия ошибок во время выполнения **PHP** кода, и, если они были, вывести их в браузер.

4. Проверку входных параметров адресной строки, на наличие параметра exit (GET параметры), который передается в случае, если пользователь решил выйти из системы и соответственно с отправкой его на главную страницу.

Для этого создается в папке **building-shop.local/include/controls** файл **login_controller.php** (листинг 3.12).

Листинг 3.12 – Код страницы login_controller.php

```
<?php
//Если передан параметр выхода, то выходим из си-
стемы
if(isset($_GET["exit"]) && $_GET["exit"] == "true")
  {
    //Удаляем cookie
    setcookie("mail", "", time() - 3600);
    setcookie("p40",  "", time() - 3600);
//Пересылаем на страницу входа (на текущую страни-
цу)
    header("Location: /");
    exit();
  }
  //Подключаем базовый код соединения с БД
  require_once("include/base.php");
  //Если пользователь уже есть в системе
  if ($isUser == true)
  {
    //то пересылаем его на главную страницу
    header("Location: /index.php");
    exit();
  }
  //Список ошибок
  $errorMessage = Array();
//Проверяем, может пользователь уже пытался войти,
или, были ли переданы какие-то данные методом POST
  if(isset($_POST['eMail'])          &&          is-
set($_POST['password']))
  {
    //Считываем и обрабатываем данные
```

```
    $eMail                          =      htmlspecial-
chars(trim($_POST['eMail']));
    $password             =             htmlspecial-
chars(trim($_POST['password']));
//Т.к. данные пользователя, хранящиеся в БД закеши-
рованы, причём дважды, то и передаваемый из тексто-
вого поля пароль тоже закешируем дважды в том же
порядке.
    $passwordMD5        = md5($password);
    $passwordMD5AndSHA1 = sha1($passwordMD5);
    //Ищем пользователя в БД
    $login = GetData("SELECT `id` FROM users WHERE
`e-mail` = '$eMail' "
."AND `password` = '$passwordMD5AndSHA1';");
    //Проверяем есть ли пользователь
    if($login != false)
    {
//Если пользователь с такими данными найден в БД,
добавим его в КУКИ на час и...
      setcookie("mail", $eMail, time() + 3600);
      setcookie("p40",   $passwordMD5,   time()   +
3600);
//Перешлём на главную страницу
      header("Location: /index.php");
      exit();
    }
//Если пользователь не найден то добавляем ошибку
    else
    {
      $errorMessage[] = "Неправильные имя пользова-
теля или пароль";
    }
  }?>
```

Код для проверки на получение страницей данных о выходе пользователя размещается в самое начало, до включения файла base.php. Это связано с тем, что в случае выхода пользователя он не будет оставаться на странице авторизации, а перейдет на главную страницу, соответственно, выполнять подключение и идентификацию пользователя не требуется.

Затем добавляется в файл **login.php** код для вывода списка ошибок сразу после блока ** **:

```
<div class="wrapper">
    <?php PrintErrorMessage($errorMessage); ?>
```

А также подключается файл **login_controller.php** в начало страницы **login.php**:

```
<?php require_once("/include/controller/login_controller.php"); ?>
```

Результат работы выше рассмотренного кода представлен на рис. 3.19.

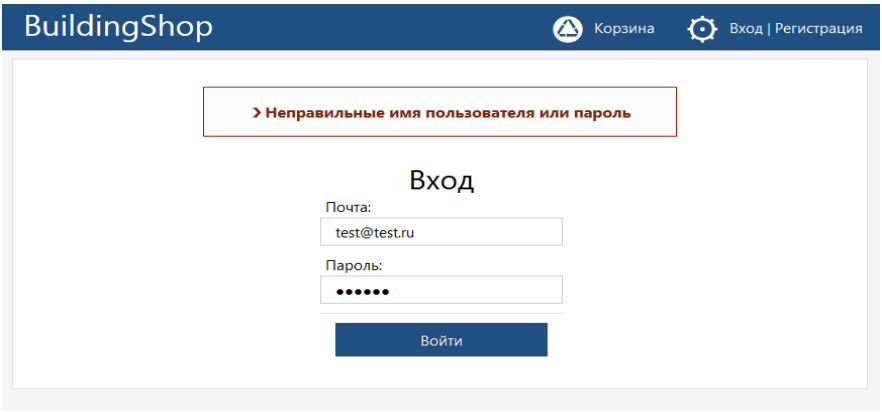

Рис. 3.19. Окно страницы авторизации пользователя с ошибкой входа

Однако прежде чем пользователь будет иметь возможность авторизоваться, ему необходимо зарегистрироваться.

Изменения на странице регистрации **register.php** будут аналогичны изменениям страницы авторизации. Данные, необходимые для регистрации будут передаваться методом **POST**. Обозначим функциональные требования, которые необходимо реализовать на странице регистрации:

- Переадресация пользователя на главную страницу **index.php**, если пользователь уже в сети и попытался перейти на страницу регистрации.
- Реализация блока, который позволит выводить сообщения об ошибках.
- Реализация блока для вывода сообщения об успешной регистрации (можно все сообщения выводить в один блок выводить, но нецелесообразно т.к. планируется использовать для них разные стили **CSS**).
- Реализация вывода сообщения об успешной регистрации, и в случае успешной регистрации убирается форма для заполнения регистрационных данных.
- Проверка введённой электронной почты на соответствие вида электронной почты.
- Проверка введенного пароля на заданные критерии (в данном случае он должен иметь длину не менее шести символов).
- Проверка введенных данных о пользователе: имени, фамилии (должно иметь длину более одного символа); адреса (не менее двадцати символов); телефона (должен иметь форму телефонного номера).
- Пароль необходимо хешировать дважды (при помощи **md5** и **sha1** функций).

Для реализации выше перечисленных требований в папке building-shop.local/include/controls создается файл register_controller.php (листинг 3.13).

Листинг 3.13 – Код файла register_controller.php

```
<?php
  //Подключаем базовый код соединения с БД
  require_once("include/base.php");
  //Если пользователь уже есть в системе
  if ($isUser)
  {
    //то пересылаем его на главную страницу
    header("Location: /index.php");
    exit();
```

```
  }
//Показывать ли нам форму для ввода  регистрацион-
ных данных
  $isShowForm = true;
  //Список с ошибками
  $errorMessage = Array();
//Если были переданы регистрационные данные методом
POST
  if(
    isset($_POST['eMail']) &&
    isset($_POST['password']) &&
    isset($_POST['name']) &&
    isset($_POST['lastname']) &&
    isset($_POST['phone']) &&
    isset($_POST['address'])
  ){
    //Извлечём их из массива _POST
    $eMail                          =    htmlspecial-
chars(trim($_POST['eMail']));
    $password                      =      htmlspecial-
chars(trim($_POST['password']));
    $name                           =    htmlspecial-
chars(trim($_POST['name']));
    $lastname                      =      htmlspecial-
chars(trim($_POST['lastname']));
    $phone                          =    htmlspecial-
chars(trim($_POST['phone']));
    $address                       =     htmlspecial-
chars(trim($_POST['address']));
//Проверяем почту на соответствие формы почты
if(!preg_match("/\A[^@]+@([^@\.]+\.)+[^@\.]+\z/",
$eMail)){
      $errorMessage[]  =  "Некорректно  ввдена  поч-
та.";
    }
    //Пароль не меньше шести символов!
    if(strlen($password) < 6){
      $errorMessage[]  =  "Пароль  должен  содержать
минимум 6 символов!.";
    }
    //Имя не менее двух.
    if(strlen($name) < 2){
      $errorMessage[]   =   "Некорректно     введено
имя.";
```

```
    }
    //Фамилия не менее двух.
    if(strlen($lastname) < 2){
      $errorMessage[] = "Некорректно  введена фами-
лия.";
    }
//Соответствие телефона форме телефонного номера.
if(!preg_match("/^(\+?\d+)?\s*(\(\d+\))?[\s-]*([\d-
]*)$/", $phone)){
      $errorMessage[] = "Некорректно  введен теле-
фон.";
    }//Адрес не менее двадцати символов.
    if(strlen($address) < 20){
      $errorMessage[] = 'Вы действительно тут живе-
те? - "'.$address.'".';
    }//Если ошибок нет
    if(count($errorMessage) == 0)
    {//Ищем пользователя в базе с такой же почтой
      $isIssetUserEMail = GetData("SELECT `id` FROM
users WHERE `e-mail` "
."= '$eMail';");
//Если нет пользователя с таким же e-mail
      if($isIssetUserEMail === false)
      {
        global $db;
        //Хешируем пароль
        $passwordMD5        = md5($password);
        $passwordMD5AndSHA1 = sha1($passwordMD5);
//Формируем запрос на добавление данных нового
пользователя
        $q = "INSERT INTO `users` (`e-mail`, `pass-
word`, `name`, `lastname`, "
        ."`phone`,  `address`)  VALUES  ('$eMail',
'$passwordMD5AndSHA1', "
            ."'$name',    '$lastname',    '$phone',
'$address');";
//Форму заполнения регистрационных данных не пока-
зывать
        $isShowForm = false;
        //Тут может возникнуть исключение
        try
        {//Добавляем данные в БД
          $isRegister = mysqli_query($db, $q);
          //Если данные не добавились
```

```
          if($isRegister == false)
          {//Выбрасываем исключение
            throw new Exception();
          }
//Если пользователь зарегистрировался он автомати-
чески входит в систему
          setcookie("mail", $eMail,        time() +
3600);
          setcookie("p40",   $passwordMD5, time() +
3600);
        }
//Если  при  регистрации  произошла  непредвиденная
ошибка
        catch(Exception $e){
          //Показываем форму
          $isShowForm = true;
          //Сообщаем пользователю об ошибке
          $errorMessage[] = "Во время регистрации
произошла ошибка.";
        }
      }//Если был найден пользователь с такой поч-
той
      else
      {
        $errorMessage[]  =  "Пользователь  с  такой
почтой - " . $eMail
                  ."., уже зарегистрирован!";
      }
    }
  }//Показывает форму (если нужно)
  function WriteRegisterForm($isShow)
  {//Проверяем, нужно ли показывать форму
    if($isShow == true)
    {
  //Включим блок с формой
      include
("/include/blocks/register_form_block.php");
    }
//Если форму показывать не нужно, то пользователь
успешно зарегистрирован
    else
    {
      echo '<div class="message_info_box">Вы успеш-
но зарегистрированы!</div>';
```

```
    }
  }
?>
```

В коде листинга 3.13 переменная **$isShowForm** отвечает за отображение формы с управляющими элементами, необходимыми для заполнения регистрационных полей.

В файле **register.php** сразу после блока ** ** идет следующий код:

```
<div class="wrapper">
  <?php
  PrintErrorMessage($errorMessage);
  WriteRegisterForm($isShowForm);
  ?>
</div>
```

В выше приведенном коде функция **PrintErrorMessage()** осуществляет отображение ошибок, произошедших в случае некорректной регистрации, функция **WriteRegisterForm()** выводить форму, если это необходимо.

В начало файла **register.php** надо добавить код:

```
<?php require_once("include/controller/register_controller.
php"); ?>
```

Результат работы выше приведенного кода представлен на рис. 3.20 – 3.21.

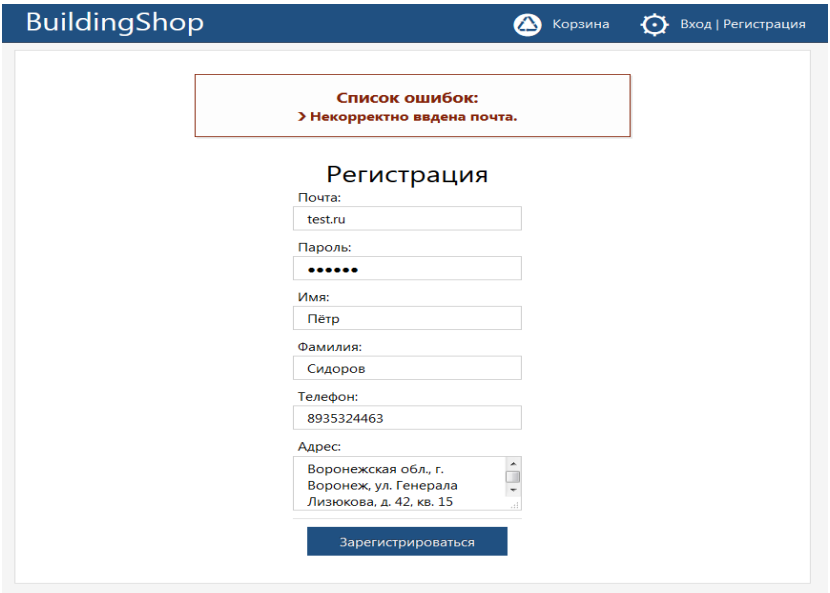

Рис. 3.20. Окно экрана некорректной регистрации пользователя

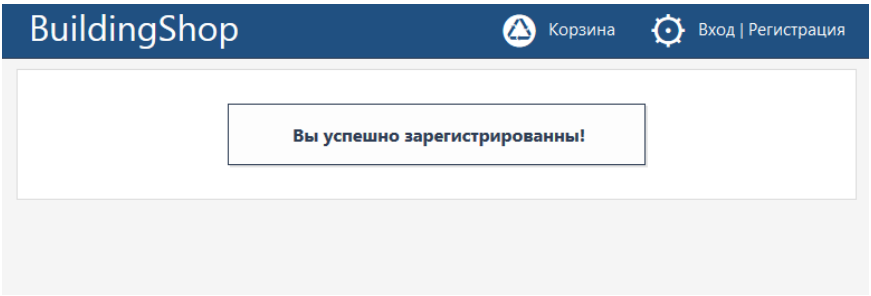

Рис. 3.21. Окно экрана успешной регистрации пользователя

После успешной регистрации пользователя вид главной страницы с авторизованным пользователем представлен на рис. 3.22.

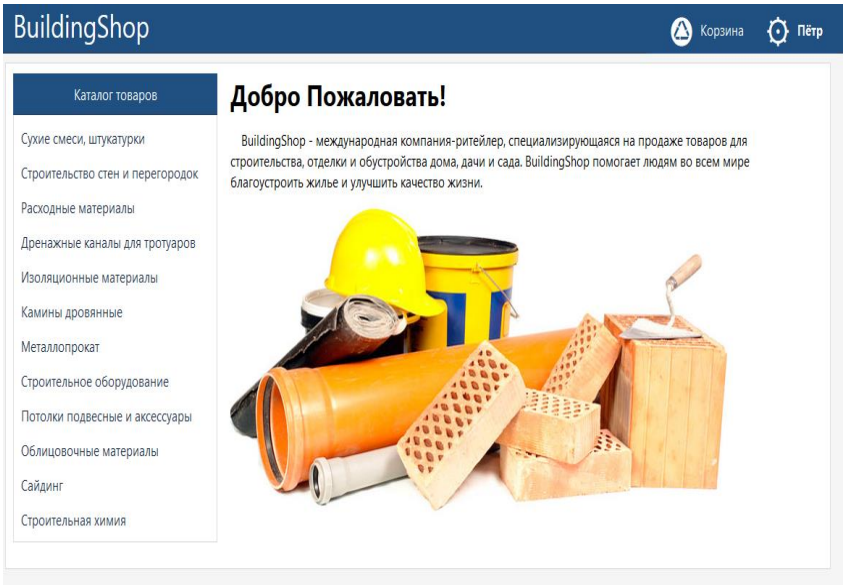

Рис. 3.22 Окно главной страницы с авторизованным пользователем

Теперь действительно видно, что когда пользователь авторизован, то в главном меню web-ресурса вместо ссылки на страницы авторизации и регистрации появляется имя пользователя, однако выпадающие блоки пока не работают.

Для реализации функциональности каталога товаров **product.php** необходимо:

- Соединение с базой данных.
- Раскрытие меню категорий при наведении курсора мыши.
- Осуществить выборку товаров по категории, если был передан соответствующий параметр методом **GET**.
- Для каждого товара создать **HTML** блок и разместить в нём следующую информацию:
  - Наименование товара.
  - Описание.
  - Категории, к которым относится товар.
  - Артикул.

- Цену.
- Количество товара на складе.
- Отметку о наличии товара на складе, исходя из предыдущего пункта.
- Кнопку добавления товара в корзину.

- Реализовать возможность добавления товара в корзину.

Для реализации выше перечисленного создается в папке **building-shop.local/include/controls** файл **product_controller.php** (листинг 3.14).

Листинг 3.14 – Код файла product_controller.php

```
<?php
  //Подключение базы данных
      require_once("include/base.php");
//Формирование запроса на выборку товаров из БД
  $q     =     "SELECT     products.id,     prod-
uct_categories.title AS categor, "
      ."products.title, products.description, prod-
ucts.price, "
      ."products.count  FROM  products  INNER  JOIN
product_categories ON "
      ."(products.category_id        =        prod-
uct_categories.id)";
  //Если методом GET передана категория
  if (isset($_GET['cat']))
  {
//То добавляем в запрос дополнительное условие для
выборки по условию принадлежности к категории
    $q = $q . " WHERE products.category_id = " .
((int)($_GET['cat']));
  }
//Если передан запрос на поиск методом GET
  if (isset($_GET['text']))
  {
//Обрабатываем строку поиска специальными функциями
(для защиты)
    global $db;
    $text = trim($_GET['text']);
    $text = mysqli_real_escape_string($text);
```

```
//Если была передана категория
    if (isset($_GET['cat']))
    {
//То одно дополнительное условие уже существует,
значит, нужно добавить еще одно
      $q = $q . " && ";
    }
//Если категория не была передана
    else
    {
//То добавляем дополнительное условие для поиска
      $q = $q . " WHERE ";
    }
//Формируем дополнительное условие поиска
    $q   =   $q   ."(product_categories.title   LIKE
'%".$text
            ."%' OR products.title LIKE '%".$text
      ."%'     OR     products.description     LIKE
'%".$text."%')";
  }
//Отправляем запрос, получаем данные
  $products = GetData($q);
  //Выводит список продуктов
  function  PrintListProducts($productsList,  $isUs-
er)
  {
  //Если список товаров не пуст
    if($productsList != false)
    {
    //То выводим их в цикле
      foreach($productsList as $product)
      {
//Используя включаемый блок товара
        include                                    ("in-
clude/blocks/product_block.php");
      }
    }

//Если список товаров пуст
    else
    {
      echo '<div class="message_info_box">Ничего не
найдено</div>';
    }
```

```
  }
?>
```

При приходе на данную страницу, категория товара будет передаваться в виде GET параметра через адресную строку

В случае если нет товаров, удовлетворяющих выборки, будет выводиться сообщение об отсутствии товара.

Список товаров выводится при помощи дополнительного включаемого блока кода с данными товара **product_block.php** в папке **building-shop.local\include\blocks** (листинг 3.15).

Листинг 3.15 – Код файла product_block.php

```
<?php
//если переменной $product или $isUser не существует
  if (isset($product) == false || isset($isUser) == false)
  {
    exit();
  }
?>
<div class="tovar_block clear" id="tovar_id_<?php echo $product['id'];?>">
  <div class="tovar_right_panel">
    <span class="price">
      <?php echo $product['price']." ₽"; ?>
    </span>
    <?php
    //Получаем кол-во товара на складе
      $count = (int)$product['count'];
      //Если товар есть на складе
      if($count > 0)
      {
        echo '<span class="count_message true">✔ Есть в наличии</span>';
        echo '<span class="count">Количество: '.$count." шт.</span>";
        //Если пользователь авторизован
        if($isUser == true)
```

```
        {
          //Добавляем активную кнопку
          echo                                    '<button
class="add_basket_tovar_button"><input
type="hidden" '
      .'value="'.$product['id'].'"                />В
корзину</button>';
        }
      }
//Если товара нет, то сообщаем пользователю
      else
      {
        echo '<span class="count_message false">✘
Нет в наличии</span>';
      }
    ?>
  </div>
  <div class="tovar_title">
    <?php echo $product['title'];?>
  </div>
  <div class="tovar_description">
    <?php echo nl2br($product['description']);?>
  </div>
  <div class="tovar_footer">
    Категория:
    <span class="gray_bold">
      <?php echo $product['categor'];?>
    </span>
    <br />
    Артикул:
    <span class="gray_bold">
      <?php echo $product['id'];?>
    </span>
  </div> </div>
```

В коде листинга 3.15 в каждый блок товара добавлено: дополнительный блок, в котором выводится информация о наличии товара, количестве, цене; кнопка, позволяющая добавить товар в корзину, где данные передаются методом **POST**.

Раскрытие списка категорий товаров при наведении указателя мыши на заголовок меню категорий реализовано при помощи **CSS**.

Кнопка добавления товара в корзину пока не функционирует, это будет реализовано позже, с применением **JavaScript** и **AJAX**.

Продемонстрируем результат на рисунках 3.23 – 3. 24.

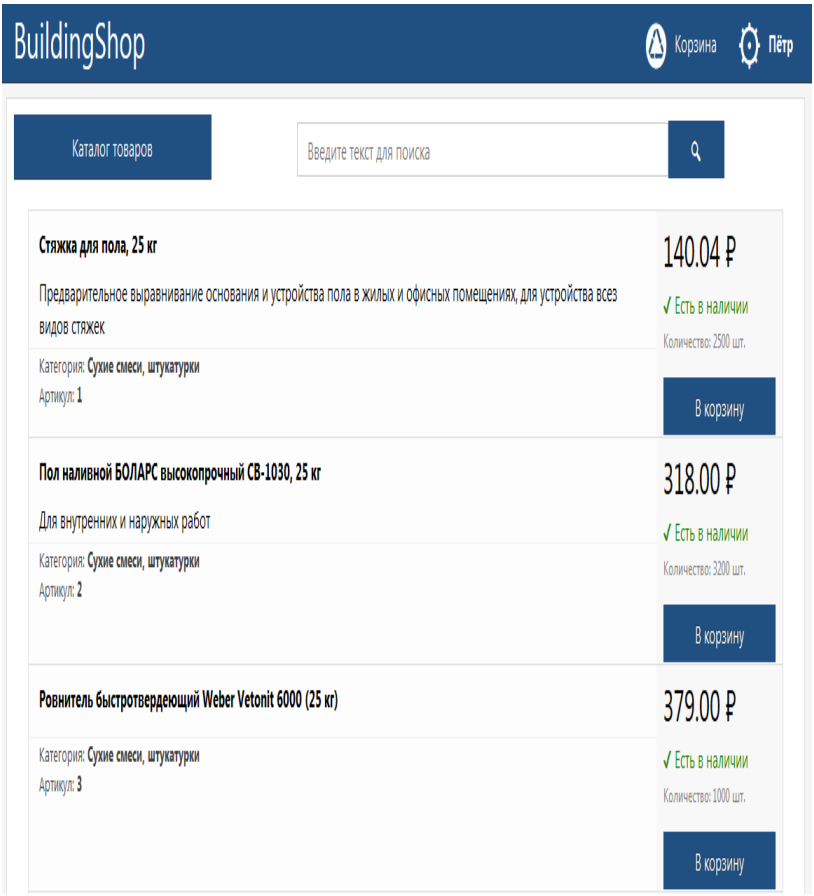

Рис. 3.23. Форма экрана заполненного списка товаров

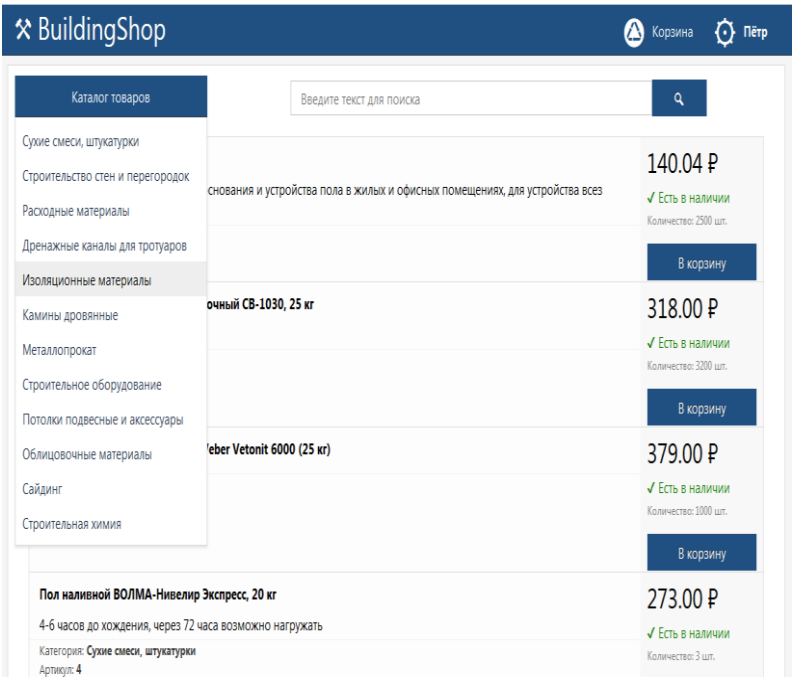

Рис. 3.24. Окно меню категорий

Для расширения функциональности раскрывающимся блокам пользователя и корзины необходимо:

- Обеспечить «выпадение» блоков из главного меню при наведении курсора мыши на значок корзины или имя пользователя.
- В блоке пользователя вывести информацию о пользователе, кнопку выхода и обеспечить её обработку.
- В блоке корзины вывести список товаров (наименование, цена, кнопка удаления товара из корзины), обеспечить обработку кнопок удаления товара, вывести итоговую стоимость всех товаров корзины и их количество.

Информация о пользователе и кнопка выхода уже разработана в **user_block.php**, кнопку выхода обрабатывать не нужно, она представлена в виде ссылке и передачи параметра

для выхода (**href="/login.php?exit=true"**), необходимо обеспечить выпадение данного блока, и вывести товары в блоке корзины.

Для начала создается файл **basket_ajax.php** в папке **building-shop.local**, в нём будут базовые функции для работы с корзиной (листинг 3.16).

Листинг 3.16 – Код файла basket_ajax.php

```
<?php
  //Подключение базовых функций
  require_once("include/base.php");
  //Если пользователь на авторизован
  if ($isUser == false)
  {
    echo '<div class="message_info">Для добавления
товаров в корзину, '
.'вы должны авторизоваться.</div>';
    exit();
  }
  //Обработка какого-либо действия, переданного ме-
тодом POST
  if (isset($_POST['act']) == true)
  {
//Извлечём информацию о переданном действии
    $act = htmlspecialchars($_POST['act']);
    //Если это запрос на добавление в корзину
    if ($act == "add")
    {
//Сформулируем текст ошибки, если она произойдёт
      $message = '<div class="message_info">'
                                  .'Ошибка  добавления
товара в корзину.</div>';
//Если ID добавляемого товара не было передано
      if (isset($_POST['tovarId']) == false)
      {
        //Выведем ошибку и завершим работу.
        echo $message;
        exit();
      }
  //Извлечём ID товара
      $tovarId = (int)$_POST['tovarId'];
  //Сформируем запрос к БД на добавление
```

```
      $qAdd = "INSERT INTO `basket` (`user_id`,
`product_id`) VALUES ('"
.$user["id"] . "', ' " . $tovarId . " ')";
//Отправим заброс, включая и текст ошибки, если та-
ковая произойдёт при попытки внести изменения в ба-
зу данных
      ChangeData($qAdd, $message, true);
    }
    //Если это запрос на удаление из корзины
    else if ($act == "del")
    {
  //Текст ошибки
      $message = '<div class="message_info">'
.'Ошибка удаления товара из корзины.</div>';
  //Если не был передан ID корзины
      if (isset($_POST['basketId']) == false)
      {
        echo $message;
        exit();
      }
  //Извлечём ID корзины
      $basketId = (int)$_POST['basketId'];
  //Сформируем запрос
      $qDel = "DELETE FROM `basket` WHERE `user_id`
= '" . $user["id"]
  ."' AND `id` = '" . $basketId . "'";
      //Отправим заброс к БД
      ChangeData($qDel, $message, true);
    }
  }
//Сформируем заброс на выборку товаров из корзины
  $q = "SELECT basket.id AS b_id, products.id,
products.title, products.price "
      ."FROM basket INNER JOIN products ON (prod-
ucts.id = basket.product_id) "
      ."WHERE basket.user_id = '" . $user["id"] .
"'";
  //Получим данные по запросу
  $data = GetData($q);
//Если ничего не получили - значит корзина пуста
  if ($data === false)
  {
    echo '<div class="message_info">В корзине нет
товаров.</div>';
```

```
    exit();
  }
  $sum   = 0;  //Посчитаем итоговую сумму всех то-
варом
  $count = 0;  //И их количество
  //Переберём все товары корзины
  foreach ($data as $tovar)
  {
    //Блок товара
    echo '<div class="user_basket_tovar">';
    //Кнопка удаления товара из корзины
    echo  '<div  class="user_basket_delete"><input
type="hidden" value="'
  .$tovar['b_id'] .'" />✖</div>';
    //Наименование товара
    echo     '<span     class="user_basket_title">'
.$tovar['title'].'</span>';
    //Цена товара
    echo  '<span  class="user_basket_price">  -  '
.$tovar['price'].' ₽</span>';
    echo '</div>';
//Посчитаем сумму цен и количество товаров.
    $sum = $sum + $tovar['price'];
    $count++;
  }
  //Добавим итоговую черту
  echo '<hr />';
//Напечатаем сумму и количество всех товаров корзи-
ны
  echo  '<div  class="user_basket_itog"><span>Кол-во
товаров:</span> ' . $count
                    .' шт.</div>';
  echo                                          '<div
class="user_basket_itog"><span>Сумма:</span>  '  .
$sum.  ' ₽</div>';
?>
```

Теперь необходимо обеспечить функциональность кнопок и выпадение списков. Для этого будет применен **JavaScript**, технология **AJAX** и библиотека **/script/jquery-2.2.0.min.js**, которая уже подключена к каждой странице web-ресурса. Файл **/script/script.js** тоже подключен к станицам ре-

сурса и создается в папке **building-shop.local\script** (листинг 3.17).

Листинг 3.17 – Код файла script.js

```
$(function(){
  var content = $('.content');
  //--- Поиск ---------------------------
  // Выбираем элементы управления поиском
  var     searchInputText     =     $('.search_block
.input_text');
  var      searchButton      =      $('.search_block
.serch_button');
  // Нажатие на кнопку поиска
  searchButton.click(function(){
    $.get("/products_ajax.php",
    { text: searchInputText.val()},
    function(data){
      content.html(data);
    });
  });
  //---------------------------------------
  //--- Выпадающие блоки сверху ----------
  $('.user_top_block').hover(function(){

$(this).find('.hidden_user_block').slideDown(120);
  },
  function(){
  $(this).find('.hidden_user_block').slideUp(50);
  });
  //---------------------------------------
  //--- Корзина --------------------------
  var userBasket = $('#userBasket');

  function ActiveDeleteButton()
  {
    // Нажатие по кнопке удалить
  $('.user_basket_delete').click(function(){
      var id = $(this).find('input').val();
      $(this).remove();
      $.post(
        "/basket_ajax.php",
        {act: "del", basketId: id},
        function(data){
```

```
            userBasket.html(data);
  ActiveDeleteButton();
          }
        );
    });
  }
  // Загрузка
  function LoadBasket()
  {
    $.post("/basket_ajax.php", function(data){
      userBasket.html(data);
      ActiveDeleteButton();
    });
  }
  LoadBasket();
  //Нажатие на кнопку "в корзину"
  $('.add_basket_tovar_button').click(function(){
    var basketButton = $(this);
    var inputData = basketButton.find('input');
    if (basketButton.attr('newAdd') == 0)
      return;
    basketButton.attr('newAdd', "0");
    var id = inputData.val();
    var backVal = basketButton.html();
    basketButton.html("Товар добавлен");
    setTimeout(function(){
      basketButton.html(backVal);
      basketButton.attr('newAdd', "1");
    }, 2000);
    $.post(
      "/basket_ajax.php",
      {act: "add", tovarId: id},
      function(data){
        userBasket.html(data);
        ActiveDeleteButton();
      }
    );
  });
});
```

В коде листинга 3.17 добавлено: обработка поиска, выпадение блоков корзины, информация о пользователе, заполнение блока корзины товарами, удаление и добавление товара

в корзину. При добавлении товара на некоторое время меняется название кнопки «В корзину» на «Товар добавлен».

В коде **JavaScript** используется дополнительный файл **products_ajax.php**, который создается в папке **building-shop.local** (листинг 3.18).

Листинг 3.18 – Код файла products_ajax.php

```
<?php
require_once("/include/controller/products_controller.php");
PrintListProducts($products, $isUser);
?>
```

Результат работы представлен на рис. 3.25 – 3.27.

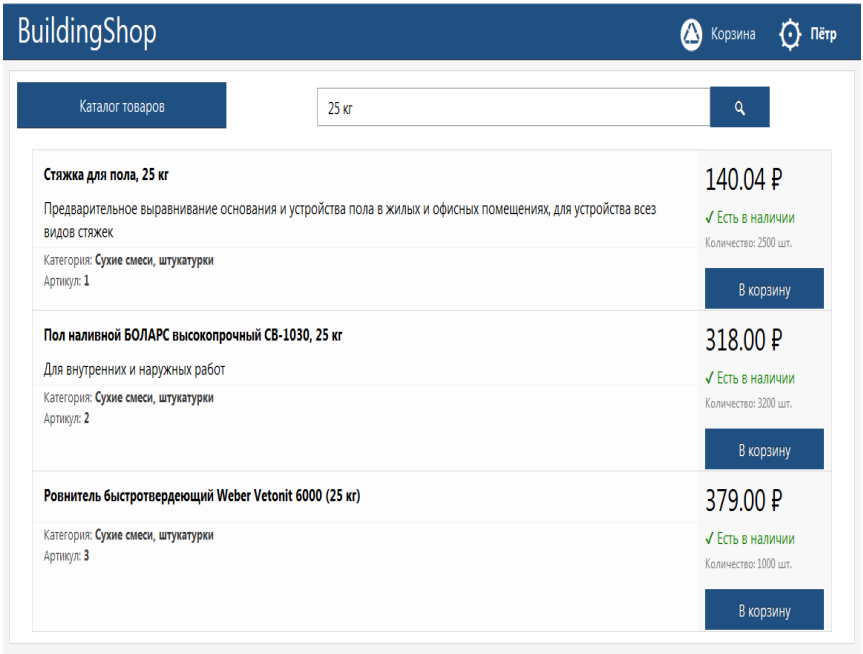

Рис. 3.25. Демонстрация поиска «25 кг»

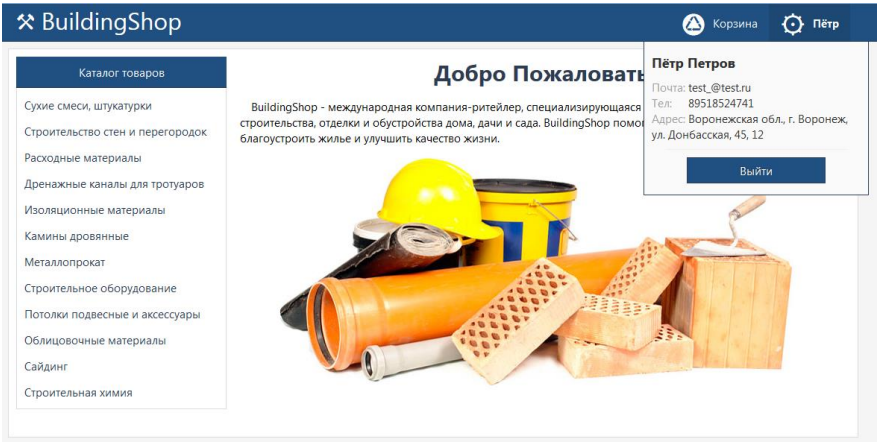

Рис. 3.26. Окно информация о пользователе

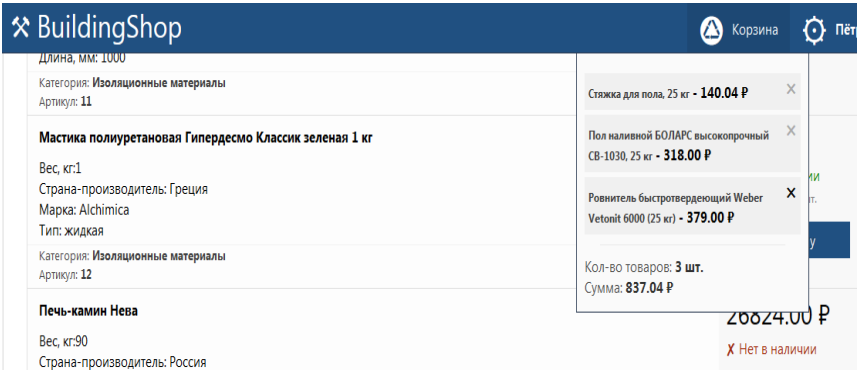

Рис. 3.27. Демонстрация роботы корзины

Итогом разработки явился полноценный web-ресурс, содержащий как клиентскую, так и серверную составляющие и реализованный с применением инструмента **phpMyAdmin**, языков **PHP**, **JavaScript**, технологией **AJAX**.

ЗАКЛЮЧЕНИЕ

Современные web-технологии – это мощный инструмент, позволяющий создавать полнофункциональные приложения, доступные как в глобальной сети Интернет, так и корпоративной Интранет -среде.

При выборе тех или иных инструментов web-разработки необходимо четко представлять функциональные возможности, предоставляемые используемой технологией и ее особенности.

Поэтому крайне важным является освоение основ разработки web-страниц и web-программирования. Данное пособие знакомит читателя с основными принципами современной web-разработки и способствует формированию системного представления об использовании современных web-технологий.

В данном практикуме даны теоретические сведенья и описаны практические действия, необходимые для разработки и создания полноценного web-ресурса.

Для реализации проекта был применен программный комплекс OpenServer в состав, которого входит: СУБД MySQL, PHP интерпретатор, web-сервер Apache, инструмент phpMyAdmin.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Быстров, П. Объектно-ориентированный анализ и проектирование систем / П. Быстров. – М. : Лори, 2012. – 264 с.

2. Веллинг, Л. Разработка web-приложений с помощью PHP и MySQL / Л. Веллинг, Л. Томсон. – М.: Издательский дом "Вильямс", 2012. – 848 с.

3. Гоше, Х.Д. HTML5. Для профессионалов: [пер. с англ.]/ Х.Д. Гоше. – СПб.: Питер, 2013. – 496 с.

4. Джилленуотер, З. Сила CSS3. Освой новейший стандарт веб-разработок / З. Джилленуотер– СПб.: Питер, 2012. – 304 с..

5. Лабберс, П. HTML 5 для профессионалов. Мощные инструменты для разработки современных веб-приложений: [пер. с англ.] / П. Лабберс, Б. Олберс, Ф. Салим. – М.: Издательский дом "Вильямс", 2011. – 272 с.

6. Маклафлин, Б. PHP и MySQL. Исчерпывающее руководство. / Б. Маклафлин. – СПб.: Питер, 2013. – 512 с.

7. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. –3-е изд. – СПб.: Питер, 2015. – 688 с.

8. Пилгрим, М. Погружение в HTML5: [пер. с англ.] / М. Пилгрим. – СПб.: БХВ-Петербург, 2011. – 304 с.

9. Пьюривал, С. Основы разработки веб-приложений. / С. Пьюривал. – СПб.: Питер, 2015. – 272 с.

10.Фрайн, Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Б. Фрайн. – СПб.: Питер, 2013. – 304 с.

11.Шварц, Б. MySQL. Оптимизация производительности: [пер. с англ.] / Б. Шварц, П. Зайцев, В. Ткаченко, Дж. Заводны, А. Ленц, Д. Бэллинг. – 2-е издание. – СПб.:Символ-Плюс, 2010. – 832 с.

12.ГОСТ на написание технического задания на создание автоматизированной системы [Электронный ресурс]. 1990. URL: http:/ /www.rugost.com

13.http://wpcreate.ru/optimizaciya-sayta/zachem-nuzhna-validatsiya-html-koda-sayta-i-kak-proverit-validatsiyu-html-koda.html

14.https://ospanel.io

15.Современный учебник JavaScript. [Электронный ресурс]: Режим доступа: http://learn.javascript.ru.

ПРИЛОЖЕНИЕ

В процессе разработки web-ресурса в контексте задания необходимо решить следующие задачи:

- проанализировать заданную предметную область в контексте задания;
- определить информационное обеспечение заданной предметной области и построить инфологическую модель данных для заданной предметной области;
- спроектировать и реализовать базу данных на основе разработанной инфологической модели с использованием инструмента phpMyAdmin;
- разработать дизайн web-ресурса для реализации решения поставленной задачи в контексте заданной предметной области;
- разработать и реализовать web-ресурса, взаимодействующий с созданной базой данных.

Индивидуальные варианты по разработке web-ресурса:

1. Разработка web-ресурса по продаже лакокрасочных материалов.
2. Разработка web-ресурса по предоставлению туристических услуг.
3. Разработка web-ресурса по продажи кровельных материалов.
4. Разработка web-ресурса по изготовлению и размещению наружной рекламы в стройиндустрии.
5. Разработка web-ресурса автосервиса.
6. Разработка web-ресурса по учету строительного оборудования и техники находящегося в аренде.
7. Разработка web-ресурса проектированию и продаже коттеджей.
8. Разработка web-ресурса магазина сантехники.
9. Разработка web-ресурса для продажи сырья по производству строительных материалов.
10. Разработка web-ресурса по учету и продажи строительного оборудования.

ОГЛАВЛЕНИЕ

Учебное издание

Курипта Оксана Валериевна
Наливайко Иван Анатольевич
Лынов Яков Васильевич

РАЗРАБОТКА WEB-РЕСУРСА

Подписано в печать _____2018.
Формат 60x84 1/16. Бумага для множительных аппаратов.
Усл. печ. л. 6,25 Тираж ____ экз. Заказ № ____.

ФГБОУ ВО «Воронежский государственный технический университет»
394026 Воронеж, Московский проспект, 14

Участок оперативной полиграфии издательства ВГТУ
394026 Воронеж, Московский проспект, 14