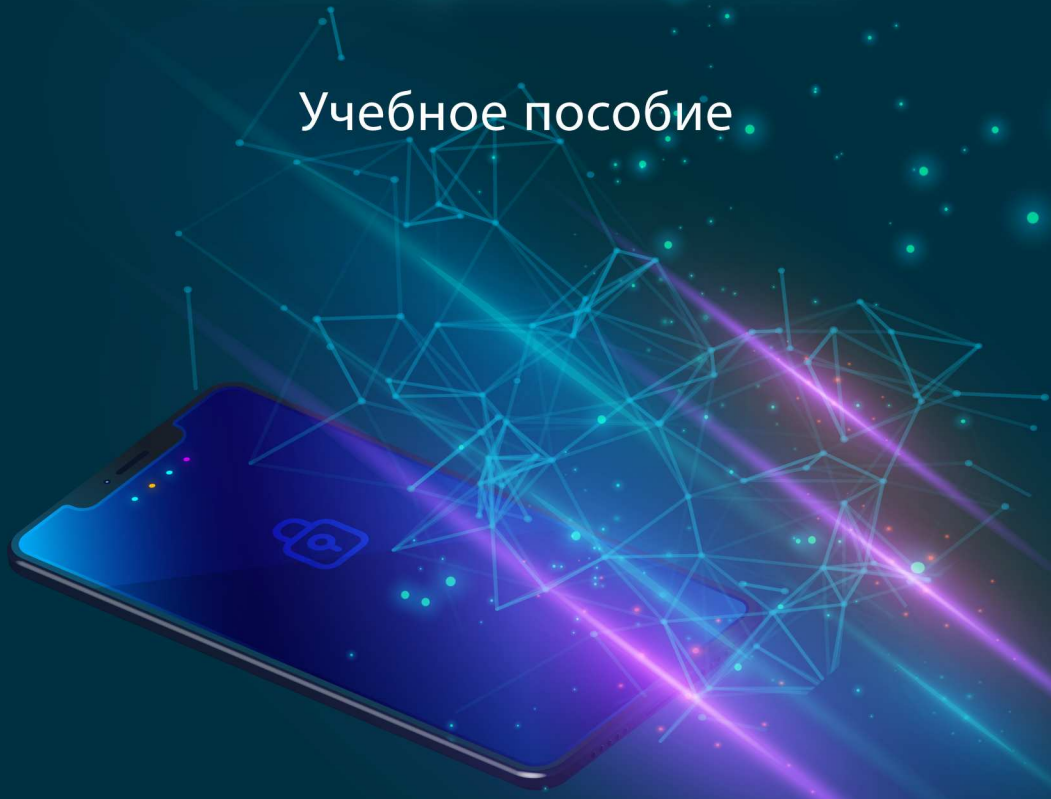
А. Л. Марухленко,
Л. О. Марухленко, М. А. Ефремов

Разработка защищённых интерфейсов Web-приложений

Учебное пособие



DirectMEDIA

А. Л. Марухленко,
Л. О. Марухленко, М. А. Ефремов

Разработка защищённых интерфейсов Web-приложений

Учебное пособие

DirectMEDIA
Москва
Берлин
2021

УДК 004.451.83(075)
ББК 16.353.12я73
М29

Рецензенты:
Довбня В. Г. — , профессор, доктор технических наук, главный научный сотрудник НИЦ (г. Курск) ФГУП «18 ЦНИИ» МО РФ
Крыжевич Л. С. — кандидат технических наук, заведующий кафедрой информационной безопасности, Курского государственного университета

Марухленко А. Л.
М29 Разработка защищённых интерфейсов Web-приложений : учебное пособие / А. Л. Марухленко, Л. О. Марухленко, М. А. Ефремов. — Москва ; Берлин : Директ-Медиа, 2021. — 174 с.

ISBN 978-5-4499-1676-1

Учебное пособие предназначено для студентов по направлению подготовки 02.03.03, 09.03.01, 09.03.02, 09.03.03, 09.03.04, 10.03.01, 10.05.02, 38.03.01, 38.03.03, 38.03.05, 38.05.01, 40.05.01, 43.03.02, 43.03.03, 45.03.03, также может быть полезно широкому кругу специалистов, повышающим свою квалификацию в области информационной безопасности и разработки защищенных информационных систем.

Рассматриваются основы информационной безопасности, особенности защиты авторских прав, технические и криптографические методы защиты, показан способ интеграции механизмов защиты ПО с использованием современных программных и аппаратных средств.

Текст приводится в авторской редакции.

УДК 004.451.83(075)
ББК 16.353.12я73

ISBN 978-5-4499-1676-1

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ

AJAX - Asynchronous Javascript And Xml – технология обращения к серверу без перезагрузки страницы

API - Application programming interface - интерфейс прикладного программирования

CSS - Cascading Style Sheets — каскадные таблицы стилей

DOM - Document Object Model — объектная модель документа

HTML - HyperText Markup Language — язык гипертекстовой разметки

HTTP - HyperText Transfer Protocol — протокол передачи гипертекста, протокол прикладного уровня передачи данных

JSON - JavaScript Object Notation – текстовый формат обмена данными

MVC - Model-View-Controller - Модель-Представление-Контроллер

SQL - structured query language — язык структурированных запросов

URL - Uniform Resource Locator - единообразный указатель ресурсов

XML - eXtensible Markup Language — расширяемый язык разметки

XSRF - The Cross-Site Request Forgery - межсайтовая подделка запроса

XSS - Cross-Site Scripting — межсайтовый скриптинг

БД – база данных;

СУБД – система управления базами данных;

ПРЕДИСЛОВИЕ

Целью данного учебного пособия является изучение принципов разработки веб-приложений на базе свободно распространяемого программного обеспечения. Важной особенностью создаваемых приложений будет отсутствие привязки к типу операционной системы.

В рамках учебного пособия вы познакомитесь с современными технологиями создания веб-ресурсов с применением фреймворков. Данный курс включает изучение основ HTML, CSS, JavaScript, PHP, MVC, SQL.

В ходе обучения, вы получите сведения о функционировании и настройке Web-серверов, научитесь работать с формами, фреймами, файлами, СУБД, регулярными выражениями, графикой и разработаете защищенную систему на базе клиент-серверной технологии, отражающей состояние определенной предметной области. В работе последовательно изложены актуальные вопросы применения средств разработки, приведены примеры выполнения типовых задач, разработаны индивидуальные задания для закрепления полученных знаний.

Учебное пособие предназначено для студентов направлений подготовки 02.03.03 «Математическое обеспечение и администрирование информационных систем», 10.05.02 «Информационная безопасность телекоммуникационных систем», 10.03.01 «Информационная безопасность автоматизированных систем», также может быть полезно широкому кругу специалистов в области информационных технологий и разработки программных средств.

ВВЕДЕНИЕ

Для успешной работы отдельных компаний и корпораций на сегодняшний день требуется все больше информационных систем, работающих в масштабе реального времени. Повсеместное распространение Интернет и создание клиент-серверных решений на базе web-технологий позволяет обеспечить доступ отдельным группам пользователей без привязки к типу операционной системы. Для простоты подобные кроссплатформенные системы, работающие с реляционными базами данных, в рамках пособия будем называть «сайтами».

Сложность и функциональность современных сайтов также возрастает и написание «с нуля» является нецелесообразной задачей. Оптимальным решением является использование фреймворков – программных платформ, определяющих структуру программной системы, программного обеспечения, облегчающего разработку и объединение разных компонентов большого программного проекта.

Вы получите сведения о функционировании Web-серверов, конфигурировании сервера Apache, научитесь работать с формами, файлами, СУБД, регулярными выражениями, графикой и подключаемыми шаблонными формами. В целях автоматизации ряда вычислений и работы с формами рассмотрен AngularJS, который позволяет создавать мощные, структурированные и простые в обслуживании приложения. Вы изучите основы php-фреймворка Laravel, научитесь настраивать окружение, разрабатывать свои базы данных. Узнаете, зачем нужны модели, контроллеры и маршруты, как их разработать. Изучите основные положения по проведению аттестации рабочих мест, включающей создание модели угроз и техническое задание для построения систем защиты.

1. ОСНОВЫ РАЗРАБОТКИ ПРЕДСТАВЛЕНИЙ И ПРОСТЕЙШИЕ ВЫЧИСЛЕНИЯ

Актуальность создания сайтов очевидна т.к. с появлением глобальной сети человек получил интерактивный инструмент, позволяющий сообщить миру о научных достижениях, услугах и товарах, привлечь единомышленников и покупателей. Как правило, расходы на содержание сайта сводятся к платежам за раскрутку, поддержание работоспособности и защиту от несанкционированного доступа.

1.1. Язык гипертекстовой разметки

HTML-документ – это обычный текстовый документ, может быть создан как в обычном текстовом редакторе (Блокнот), так и в специализированном, с подсветкой кода (Notepad++, SublimeText, PhpStorm). HTML-документ имеет расширение *.html.

Язык HTML – язык тегов. Теги описывают структуру HTML-документа. Теги оформляются угловыми скобками <имя тега>, между которыми прописывается имя тега.

Теги располагаются в HTML-документе в соответствии с правилами разметки (порядок следования, правило вложенности тегов), создавая разделы будущей веб-страницы. Кроме тегов, HTML-документы могут содержать специальные символы.

Браузер просматривает (интерпретирует) HTML-документ, выстраивая его структуру (DOM) и отображая ее в соответствии с инструкциями, включенными в этот файл (таблицы стилей, скрипты). Если разметка правильная, то в окне браузера будет отображена HTML-страница, содержащая HTML-элементы – заголовки, таблицы, изображения и т.д.

Процесс интерпретации (парсинг) начинается прежде, чем веб-страница полностью загружена в браузер. Браузеры обрабатывают HTML-документы последовательно, с самого начала, при этом обрабатывая CSS и соотнося таблицы стилей с элементами страницы.

Практически все теги имеют атрибуты (глобальные, применяемые для всех html-элементов, и собственные), указываемые в формате параметр="значение". Атрибуты позволяют изменять

свойства элемента, для которого они заданы. Атрибуты прописываются в начальном теге элемента.

Атрибуты class и id применимы ко всем HTML-элементам, за исключением элементов, содержащих техническую информацию – <html>, <head>, <meta>, <title>, <style> и <script>. Каждому конкретному элементу можно присвоить несколько значений class и только одно значение id.

Множественные значения class записываются через пробел, . Значения class и id должны состоять только из букв, цифр, дефисов и нижних подчеркиваний и должны начинаться только с букв или цифр.

Большинство тегов – парные, они состоят из начального и закрывающего тегов. Начальный тег показывает, где начинается элемент, закрывающийся – где заканчивается. Закрывающий тег образуется путем добавления слэша/ перед именем тега: <имя тега>…</имя тега>. Между начальным и закрывающим тегами находится содержимое тега – контент.

Одиночные теги не могут хранить в себе содержимого напрямую, оно прописывается как значение атрибута, например, тег <input type="button" value="Кнопка"> создаст кнопку с текстом Кнопка внутри.

Теги могут вкладываться друг в друга, например, Текст . При вложении следует соблюдать порядок их закрытия (принцип «матрёшки»), например, следующая запись будет неверной: Текст .

HTML-документ состоит из двух разделов – заголовка (между тегами <head>…</head>) и содержательной части (между тегами <body>…</body>).

Элементы, находящиеся внутри тега <html>, образуют дерево документа, так называемую объектную модель документа, DOM (document object model). При этом элемент <html> является корневым элементом (рисунок 1).

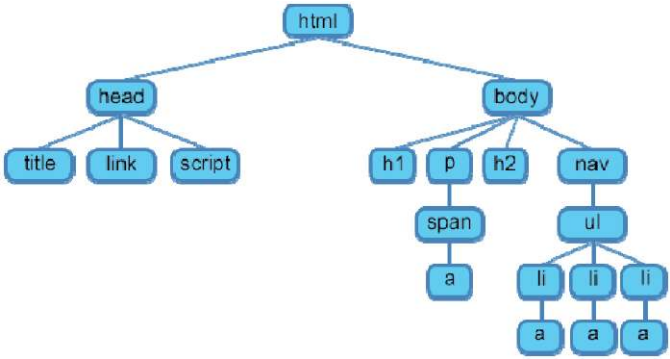


Рисунок 1 – Структура веб-страницы

Чтобы разобраться во взаимодействии элементов веб-страницы, необходимо рассмотреть так называемые «родственные отношения» между элементами. Отношения между множественными вложенными элементами подразделяются на родительские, дочерние и сестринские.

Предок – элемент, который заключает в себе другие элементы. На рисунке 1 предком для всех элементов является <html>. В то же время элемент <body> является предком для всех содержащихся в нем тегов: <h1>, <p>, , <nav> и т.д.

Потомок – элемент, расположенный внутри одного или более типов элементов. Например, <body> является потомком <html>, а элемент <p> является потомком одновременно для <body> и <html>.

Родительский элемент – элемент, связанный с другими элементами более низкого уровня, и находящийся на дереве выше их. На рисунке 1 <html> является родительским только для <head> и <body>. Тег <p> является родительским только для .

Дочерний элемент – элемент, непосредственно подчиненный другому элементу более высокого уровня. На рисунке 1 только элементы <h1>, <h2>, <p> и <nav> являются дочерними по отношению к <body>.

Сестринский элемент – элемент, имеющий общий родительский элемент с рассматриваемым, так называемые элементы одного уровня. На рисунке 1 <head> и <body> – элементы одного уров-

ня, так же как и элементы <h1>, <h2> и <p> являются между собой сестринскими.

HTML теги являются основой языка HTML. Каждый тег несет в себе определенную информацию, может описывать документ в общем или способ форматирования текста. Все содержимое веб-страницы задается с помощью тегов.

Тег помещается в угловые скобки <тег>. Чаще всего для тега задается парный закрывающий тег, но в некоторых случаях он отсутствует. Информация, заключенная между открывающим и закрывающим тегом, называется его контейнером [1].

HTML атрибуты сообщают браузеру, каким образом должен отображаться тот или иной элемент страницы. Атрибуты позволяют сделать более разнообразными внешний вид информации, добавляемой с помощью одинаковых тегов. Значение атрибута всегда заключается в двойные кавычки "". Названия и значения атрибутов не чувствительны к регистру, но, тем не менее, рекомендуется набирать их в нижнем регистре.

HTML таблицы состоят из ячеек, образующихся при пересечении строк и столбцов. Ячейки таблиц могут содержать любые HTML-элементы, такие как заголовки, списки, текст, изображения, элементы форм, а также другие таблицы. Таблицы в HTML-документах используются не только для группировки связанной информации, но и для точного позиционирования фрагментов текста и изображений относительно друг друга. С помощью таблиц можно выровнять фрагменты страниц относительно друг друга, разместить рядом изображение и текст, управлять цветовым оформлением, разбить текст на столбцы.

Создание сайта всегда начинается с создания файла index.html. Index – это общепринятое название главной страницы сайта. Общий вид HTML-документа представлен ниже.

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Шаблон-пример</title>
   <script src="*.js"></script>
```

```
    <link href="*.css" rel="stylesheet">
    </head>
<body>
    <p>Привет, мир!</p>
</body>
</html>
```

<!DOCTYPE> – предназначен для указания типа текущего документа. Это необходимо, чтобы браузер понимал, как следует интерпретировать текущую веб-страницу, поскольку HTML существует в нескольких версиях. Чаще всего используется значение html как в нашем примере.

Теги <html>, <head>, <body> были рассмотрены ранее. Атрибут lang="ru" относящийся к тегу <html> используется для правильного отображения некоторых национальных символов

Рассмотрим по подробнее наполнение <head>:

– <meta charset="utf-8"> используется для определения кодировки документа;

– <title>Шаблон-пример</title> – это заголовок страницы, который будет отображаться в браузере;

– <script src="*.js"></script> используется для подключение внешних скриптов. Вместо * указывается путь и имя подключаемого файла;

– <link href="*.css" rel="stylesheet"> используется для подключение внешних каскадных таблиц стилей. Вместо * указывается путь и имя подключаемого файла. Атрибут rel определяет отношения между текущим документом и файлом, на который делается ссылка. Это необходимо, чтобы браузер знал, как использовать подключаемый документ. Значение stylesheet определяет, что подключаемый файл хранит таблицу стилей (CSS).

1.2. Каскадные таблицы стилей

CSS (Cascading Style Sheets), или каскадные таблицы стилей, описывают правила форматирования отдельного элемента веб-страницы. Создав стиль один раз, его можно применять к любым элементам страницы сколько угодно раз.

Определение стиля состоит из двух основных частей: самого элемента веб-страницы – селектора, и команды форматирования – блока объявления.

Селектор сообщает браузеру, какой именно элемент форматировать, в блоке объявления перечисляются форматирующие команды.

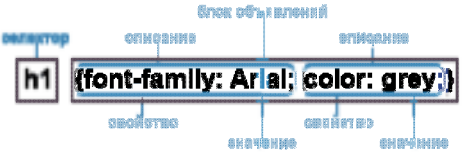


Рисунок 2 – Структура объявления стиля элемента в CSS.

Встраиваемые стили представляют собой набор стилей, являющихся частью кода веб-страницы, заключенных между тегами <style>...</style> и расположенных внутри элемента <head>. Встраиваемые стили действуют только на странице, на которой они содержатся. Встраиваемые стили имеют приоритет перед глобальными, но уступают внутритекстовым стилям. На одной странице можно размещать произвольное количество встраиваемых стилей.

Внутритекстовые стили не пользуются селекторами, их применение заключается в присваивании стиля непосредственно к html-элементу через атрибут style:

<p style="font-family: "Times New Roman", Georgia, Serif; color: #70d7700;">
Обратите внимание на этот текст.</p>

Недостаток этого способа заключается в отсутствии возможности автоматического использования данного стиля для другого элемента.

Внешняя таблица стилей представляет собой текстовый файл, в котором находится весь набор стилей CSS. Он не содержит HTML-код, поэтому его не нужно заключать внутрь тегов <style>...</style>. Название этого файла всегда должно заканчиваться расширением *.css. Заданные таким образом стили будут работать для всех страниц веб-сайта.

К веб-странице можно присоединить несколько таблиц стилей, добавляя последовательно несколько тегов <link>.

Стилизовать элементы html кода, используя внешние каскадные таблицы стилей, можно несколькими способами.

```
#primer1{
   background-color: #ffcacc;
}
.primer2{
   background-color: #9bff99;
}
p{
   font-size: 20px;
}
```

Первый способ – это использование селектора id. Селектор id (идентификатор) предназначен для применения стиля к уникальным элементам на веб-странице, уникальность подразумевает то, что данный элемент используется на странице один раз.

Для использования селектора id, нужно создать идентификатор (id), придумав ему уникальное название, и прописать его в атрибуте id элемента, к которому будет применяться стиль. В описании стиля селектор id начинается с символа # сразу после которого идет название идентификатора.

Каждый идентификатор стиля может встречаться на странице только один раз, т.е. определенный id может быть использован на странице только с тегом, для которого он предназначен, если один и тот же идентификатор будет применен более, чем к одному эле-

менту, во-первых html-код не пройдет валидацию, во-вторых это может вызвать некорректную обработку кода браузером и вы увидите не тот результат, которого ожидали.

Второй способ – это использование селектора class. Селектор класс позволяет так же как и селектор id стилизовать конкретный элемент страницы, но в отличие от id, селектор class позволяет применить свой стиль к нескольким элементам на веб-странице, а не только к одному.

Для использования селектора class, нужно указать, к какому элементу на странице вы хотите его применить, для этого надо всего лишь добавить атрибут class к HTML-тегу, который нужно стилизовать, и указать в качестве значения нужное имя класса с описанным стилем.

Правила для имен классов:

– все названия селекторов классов должны начинаться с точки, с ее помощью браузеры находят селектор класса в таблице стилей CSS;

– в имени класса разрешается использовать только буквы, числа, дефис и знак подчеркивания;

– имя класса после точки всегда должно начинаться с буквы;

– имена классов чувствительны к регистру, например .Menu и .menu будут рассматриваться в CSS, как два разных класса.

Третий способ – это применение стилей для конкретного тега. В этом случае в файле со стилями пишется название тега без <>. При использовании этого способа во всём документе к указанному тегу будет применяться заданный стиль [2].

К одному и тому же элементу можно применять сразу несколько способов стилизации. Пример кода представлен ниже.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <title>Шаблон-пример</title>
  <link href="css/style.css" rel="stylesheet">
  </head>
<body>
```

```
    <p class="primer2">Привет, мир!</p>
</body>
</html>
```

В данном примере подключены внешние таблицы стилей, код которых описан выше. То есть к тегу <p> применены второй и третий способ стилизации. В браузере будет отображаться текст с размером шрифта в 20 px и с закрашенным фоном.

Так же можно к одному элементу присваивать несколько различных классов. Главная особенность заключается в том, что если в стилях различных классов встречаются одинаковые свойства, то к html-элементу будет применено то свойство, которое в таблице стилей находится ниже.

Все свойства и способы их применения для стилизации можно посмотреть на различных интернет ресурсах, например, http://htmlbook.ru/.

1.3. Прототипно-ориентированный сценарный язык программирования

JavaScript – язык сценариев, с его помощью можно создавать интерактивные html-документы, производить вычисления, выполнять проверку допустимости данных без обращения к серверу.

Сценарий JavaScript не компилируется в бинарный код наподобие .exe, а интерпретируется и обрабатывается интерпретатором, встроенным в браузер. Браузер начинает выполнять код JavaScript сразу же, как только обнаружит его на странице. Каждая строка кода выполняется последовательно, переход к следующей строке производится только после выполнения предыдущей. Если в документе содержится несколько сценариев, то они будут исполняться в порядке их расположения в документе.

Сценарии JavaScript бывают встроенные, т.е. их содержимое является частью документа, и внешние, хранящиеся в отдельном файле с расширением *.js. Сценарии можно внедрить в html-документ следующими способами:

– в виде гиперссылки. Для этого нужно разместить код в отдельном файле и включить ссылку на файл в заголовок. Этот спо-

соб обычно применяется для сценариев большого размера или сценариев, многократно используемых на разных веб-страницах;

– в виде обработчика события. Каждый html-элемент имеет JavaScript-события, которые срабатывают в определенный момент. Нужно добавить необходимое событие в html-элемент как атрибут, а в качестве значения этого атрибута указать требуемую функцию. Функция, вызываемая в ответ на срабатывание события, является обработчиком события. В результате срабатывания события исполнится связанный с ним код. Этот способ применяется в основном для коротких сценариев, например, можно установить смену цвета фона при нажатии на кнопку;

– внутрь элемента <script>. Элемент <script> может вставляться в любое место документа. Внутри тега располагается код, который выполняется сразу после прочтения браузером, или содержит описание функции, которая выполняется в момент ее вызова. Описание функции можно располагать в любом месте, главное, чтобы к моменту ее вызова код функции уже был загружен.

Обычно код JavaScript размещается в заголовке документа (элемент <head>) или после открывающего тега <body>. Если скрипт используется после загрузки страницы, например, код счетчика, то его лучше разместить в конце документа.

1.4. Типы данных и переменные

Компьютеры обрабатывают информацию – данные. Данные могут быть представлены в различных формах или типах. Большая часть функциональности JavaScript реализуется за счет простого набора объектов и типов данных. Функциональные возможности, связанные со строками, числами и логикой, базируются на строковых, числовых и логических типах данных. Другая функциональная возможность, включающая регулярные выражения, даты и математические операции, осуществляется с помощью объектов RegExp, Date и Math.

Литералы в JavaScript представляют собой особый класс типа данных, фиксированные значения одного из трех типов данных – строкового, числового или логического.

Данные, обрабатываемые сценарием JavaScript, являются переменными. Переменные представляют собой именованные контейнеры, хранящие данные (значения) в памяти компьютера, которые могут изменяться в процессе выполнения программы. Переменные имеют имя, тип и значение.

Имя переменной, или идентификатор, может включать только буквы a-z, A-Z, цифры 0-9 (цифра не может быть первой в имени переменной), символ $ (может быть только первым символом в имени переменной или функции) и символ подчеркивания _, наличие пробелов не допускается. Длина имени переменной не ограничена. Можно, но не рекомендуется записывать имена переменных буквами русского алфавита, для этого они должны быть записаны в Unicode.

В качестве имени переменной нельзя использовать ключевые слова JavaScript. Имена переменных в JavaScript чувствительные к регистру, что означает, что переменная varmessage; и varMessage: – разные переменные.

Переменная создается (объявляется) с помощью ключевого слова var, за которым следует имя переменной, например, varmessage;. Объявлять переменную необходимо перед ее использованием.

Переменная инициализируется значением с помощью операции присваивания =, например, varmessage="Hellow";, т.е. создается переменная message и в ней сохраняется ее первоначальное значение "Hellow". Переменную можно объявлять без значения, в этом случае ей присваивается значение по умолчанию undefined. Значение переменной может изменяться во время исполнения скрипта. Разные переменные можно объявлять в одной строке, разделив их запятой: varmessage="Hellow", number_msg = 6, time_msg = 50;

1.5 Арифметические операторы

Арифметические операторы (таблица 1) предназначены для выполнения математических операций, они работают с числовыми операндами (или переменными, хранящими числовые значения), возвращая в качестве результата числовое значение.

Если один из операндов является строкой, интерпретатор JavaScript попытается преобразовать его в числовой тип, а после выполнить соответствующую операцию. Если преобразование типов окажется невозможным, будет получен результат NaN (не число).

Таблица 1 – Операторы в JavaScript.

Оператор/ операция	Описание	Приоритет
+ Сложение	Складывает числовые операнды. Если один из операндов – строка, то результатом выражения будет строка.	12
- Вычитание	Выполняет вычитание второго операнда из первого.	12
- Унарный минус	Преобразует положительное число в отрицательное, и наоборот.	14
* Умножение	Умножает два операнда.	13
/ Деление	Делит первый операнд на второй. Результатом деления может являться как целое, так и число с плавающей точкой.	13
% Деление по модулю (остаток от деления)	Вычисляет остаток, получаемый при целочисленном делении первого операнда на второй. Применяется как к целым числам, так и числам с плавающей точкой.	13
== Равенство	Проверяет две величины на совпадение, допуская преобразование типов. Возвращает true, если операнды совпадают, и false, если они различны.	9
!= Неравенство	Возвращает true, если операнды не равны	9

Продолжение таблицы 1

=== Идентичность	Проверяет два операнда на «идентичность», руководствуясь строгим определением совпадения. Возвращает true, если операнды равны без преобразования типов.	9
!== Неидентичность	Выполняет проверку идентичности. Возвращает true, если операнды не равны без преобразования типов.	9
> Больше	Возвращает true, если первый операнд больше второго, в противном случае возвращает false.	10
>= Больше или равно	Возвращает true, если первый операнд не меньше второго, в противном случае возвращает false.	10
< Меньше	Возвращает true, если первый операнд меньше второго, в противном случае возвращает false.	10
<= Меньше или равно	Возвращает true, если первый операнд не больше второго, в противном случае возвращает false.	10

1.6. Библиотека JQUERY

jQuery – библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX. Сейчас разработка jQuery ведётся командой jQuery во главе с Джоном Резигом.

Возможности:

– движокккроссбраузерных CSS-селекторов Sizzle, выделившийся в отдельный проект;

– переход по дереву DOM, включая поддержку XPath как плагина;

– события;

– визуальные эффекты;
– AJAX-дополнения;
– JavaScript-плагины.

Точно так же, как CSS отделяет визуализацию от структуры HTML, JQuery отделяет поведение от структуры HTML. Например, вместо прямого указания на обработчик события нажатия кнопки управление передаётся JQuery, которая идентифицирует кнопки и затем преобразует его в обработчик события клика. Такое разделение поведения и структуры также называется принципом ненавязчивого JavaScript.

Библиотека jQuery содержит функциональность, полезную для максимально широкого круга задач. Тем не менее, разработчиками библиотеки не ставилась задача совмещения в jQuery функций, которые подошли бы всюду, поскольку это привело бы к большому коду, бóльшая часть которого не востребована. Поэтому была реализована архитектура компактного универсального ядра библиотеки и плагинов. Это позволяет собрать для ресурса именно ту JavaScript-функциональность, которая на нём была бы востребована.

28 сентября 2008 года на официальном блоге jQuery сообщили о том, что компании Microsoft и Nokia собираются сотрудничать с группой разработчиков. Компания Microsoft собирается интегрировать в свой продукт ASP.NET листинги кода и примеры jQuery, а компания Nokia собирается интегрировать jQuery для своих мобильных виджетов.

Библиотека jQuery производит манипуляции с html-элементами, управляя их поведением и используя DOM для изменения структуры веб-страницы. При этом исходные файлы HTML и CSS не меняются, изменения вносятся лишь в отображение страницы для пользователя.

Для выбора элементов используются селекторы CSS. Выбор осуществляется с помощью функции $(). При вызове функция $() возвращает новый экземпляр объекта JQuery, который оборачивает ноль или более элементов DOM и позволяет взаимодействовать с ними различными способами.

Выполнение различных сценариев возможно только после окончания загрузки структуры документа document, когда браузер

преобразует html-код страницы в дерево DOM. Управление процессом загрузки обеспечивает конструкция.

Сначала производится обертывание экземпляра document в функцию jQuery(), после применяется метод ready(), которому передается функция function() {...}, исполняемая после загрузки документа.

На практике обычно используется сокращенная форма такой записи jQuery(function() {...});, или $(function() {...});.

Для хранения информации при работе с библиотекой jQuery используются переменные JavaScript. В переменных могут храниться элементы. Имена переменных, предназначенных для хранения возвращаемых элементов, начинаются со знака $. Для хранения нескольких элементов используются массивы JavaScript:

Библиотека jQuery упрощает процесс отбора элементов HTML-страниц. С помощью методов jQuery производятся манипуляции с объектной моделью документа DOM. Чтобы отобрать группу элементов, нужно передать селектор функции jQuery. В качестве селектора элемента может выступать сам элемент, его идентификатор или класс, а также комбинация селекторов:

Функция $() возвращает объект jQuery, содержащий массив элементов DOM — так называемый обернутый набор, соответствующих указанному селектору. Большинство методов возвращает по завершении действий первоначальный набор элементов.

Селекторы jQuery выбирают элементы веб-страницы, а методы выполняют операции с этими элементами.

Чтобы выбрать элементы, нужно передать селектор функции $(), например, $("img:odd"). Данный селектор будет применен ко всему дереву DOM, и чтобы ограничить процедуру отбора элементов, можно указать определенный фрагмент дерева DOM – $("img:odd", "div#slideshow"). Таким образом будут выбраны все четные картинки из блока с id=slideshow.

Для более точного выбора элементов селекторы можно комбинировать, например, следующая запись позволит выбрать все флажки полей формы, которые были выделены пользователем – $("input[type=checkbox][checked]").

А с помощью этой комбинации селекторов $("input:checkbox:checked:enabled") можно выбрать только активные и отмеченные флажки полей формы.

Также, допускается объединять несколько селекторов в одно выражение, разделяя селекторы запятой – $("p,span"), что позволит отобрать все элементы <p> и .

Таблица 2 – Селекторы jQuery

Селектор	Описание, пример
Элемента	Выбирает все элементы данного типа на странице, например, $("div").
Элемента 1 элемента 2	Выбирает все элементы 2, вложенные непосредственно в элемент 1, например, $("p img").
Класса	Выбирает все элементы заданного класса, например, $(".sidebar").
Идентификатора	Выбирает элемент с указанным идентификатором, например, $("#main").
Элемента класса	Выбирает из элементов данного типа только те элементы, которым назначен указанный класс, например, $("p.first").
Потомка	Выбирает все указанные элементы выбранного селектора, например, $(".sidebar a").
Дочерние	Выбирает элементы, соответствующие второму селектору, которые содержатся непосредственно внутри первого селектора, являющиеся дочерними по отношению к нему, например, $("body> p").
Сестринские	Выбирает элементы, соответствующие второму селектору, идущие непосредственно за первым элементом, являющимся для него сестринским, например, $("h2 + p").
Атрибута	Выбирает элементы, соответствующие второму селектору, являющиеся сестринскими по отношению к первому элементу и расположенные после него, например, $("h2 ~ p").

Продолжение таблицы 2

Селектор	Описание, пример
	Выбирает все элементы, которые содержат данный атрибут или указанно значение атрибута, например, $("img[alt]"), $("a[href]"), $("input[type="text"]").
	Выбирает все элементы, начинающиеся с определенного значения, например, $("a[href^="http://"]").
	Выбирает все элементы, заканчивающиеся на определенное значение, например, $("a[href$=".pdf"]").
	Выбирает все элементы, содержащие в любом месте определенное значение, например, $("a[target*="blank"]").
:even	Выбирает элементы по четным значениям индекса 0, 2, 4…, т.е. выбирает 1, 3, 5… элементы, например, $("li:even").
:odd	Выбирает элементы по нечетным значениям индекса, т.е. выбирает 0, 2, 4… элементы.
:first	Выбирает только один элемент, первый из подходящих, например, $("p:first").
:last	Выбирает только один элемент, последний из подходящих.
:first-child	Выбирает элементы, которые являются первыми дочерними элементами своих родителей.
:last-child	Выбирает элементы, которые являются последними дочерними элементами своих родителей.
:only-child	Выбирает элементы, являющиеся единственными дочерними элементами своих родителей.
:nth-child(n)	Выбирает элементы, которые являются n-дочерними элементами своих родителей.
:nth-child(Xn+Y)	Выбирает n-элемент, порядковый номер которого рассчитывается по формуле в круглых скобках.

Продолжение таблицы 2

Селектор	Описание, пример
:nth-of-type(n)	Выбирает элементы, являющиеся n-ми дочерними элементами данного типа для своих родителей.
:parent	Выбирает непустые элементы, которые имеют вложенные (дочерние) элементы, а также содержащие текст.
:eq(n)	Выбирает элементы с индексом n, при этом индексы отсчитываются от нуля.
:gt(n)	Выбирает все элементы, индекс которых больше n, индексы отсчитываются от нуля.
:lt(n)	Выбирает все элементы, расположенные перед n-элементом, не включая n-элемент.
:not(селектор)	Позволяет выбрать элемент, не соответствующий данному типу селектора, например, $("a:not(.link)"), $("a:not([href$=".pdf"])").
:has(селектор)	Выбирает элементы, которые содержат внутри себя указанный селектор, например, элементы списка, содержащие внутри себя ссылки: $("li:has(a)").
:contains(текст)	Выбирает элементы, которые содержат указанный в скобках текст, например, $("a:contains(Скачать)").
:hidden	Выбирает скрытые элементы, для которых установлено значение display: none;, а также элементы форм со значением type="hidden" например, $("ul:hidden").show() — делает скрытые элементы видимыми.
:visible	Выбирает видимые элементы, к видимым элементам относятся элементы, размеры которых больше нуля, а также элементы со значением visibility: hidden и opacity: 0.
:active	Выбирает элемент, который активизирован пользователем, например, с помощью щелчка мыши.
:checked	Выбирает только отмеченные флажки или радиокнопки.

Продолжение таблицы 2

Селектор	Описание, пример
:focus	Выбирает элемент, находящийся в фокусе.
:hover	Выбирает элемент, на который наведен указатель мыши.
:disabled	Выбирает неактивные элементы (форм).
:enabled	Выбирает активные элементы (форм).
:empty	Выбирает элементы, не содержащие дочерних элементов.
:target	Выбирает элементы, на которые ссылается идентификатор фрагмента в url-адресе.
:animated	Выбирает все анимируемые в данный момент элементы.
:button	Выбираетвсекнопки input[type=submit], input[type=reset], input[type=button], button.
:checkbox	Выбирает элементы-флажки input[type=checkbox].
:file	Выбирает элементы типа file, input[type=file].
:header	Выбирает элементы-заголовки от h1 до h6.
:image	Выбирает изображения в элементах форм input[type=image].
:input	Выбираетэлементыформ input, select, textarea, button.
:password	Выбирает элементы ввода пароля input[type=password].
:radio	Выбирает радиокнопки input[type=radio].
:reset	Выбираеткнопкисброса input[type=reset], button[type=reset].
:selected	Выбирает выделенные элементы option.
:submit	Выбирает кнопки отправки формы input[type=submit], button[type=submit].
:text	Выбирает элементы ввода текстаinput[type=text].

1.7. Текстовый формат обмена данными JSON

JSON (англ. JavaScriptObjectNotation) – текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми. Формат JSON был разработан Дугласом Крокфордом.

Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

За счёт своей лаконичности по сравнению с XML, формат JSON может быть более подходящим для сериализации сложных структур. Если говорить о веб-приложениях, в таком ключе он уместен в задачах обмена данными как между браузером и сервером (AJAX), так и между самими серверами (программные HTTP-интерфейсы).

Поскольку формат JSON является подмножеством синтаксиса языка JavaScript, то он может быть быстро десериализован встроенной функцией eval(). Кроме того, возможна вставка вполне работоспособных JavaScript-функций. В языке PHP, начиная с версии 5.2.0, поддержка JSON включена в ядро в виде функций json_decode() и json_encode(), которые сами преобразуют типы данных JSON в соответствующие типы PHP и наоборот.

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

– набор пар ключ: значение. В различных языках это реализовано как объект, запись, структура, словарь, хэш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка (регистрозависимая: имена с буквами в разных регистрах считаются разными), значением – любая форма;

– упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

Это универсальные структуры данных: как правило, любой современный язык программирования поддерживает их в той или иной форме. Они легли в основу JSON, так как он используется для обмена данными между различными языками программирования.

В качестве значений в JSON могут быть использованы:

– объект – это неупорядоченное множество пар ключ: значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми;

– массив (одномерный) – это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми;

– число;

– литералы true, false и null;

– строка – это упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\» (поддерживаются варианты \", \\, \/, \t, \n, \r, \f и \b), или записаны шестнадцатеричным кодом в кодировке UTF-8 в виде \uFFFF.

1.8. Индивидуальные задания

Таблица 3

№	Дано	Найти
1.	Две стороны треугольника и угол между ними	Длину третьей стороны и площадь этого треугольника
2.	Четырёхзначное число	Произведение всех цифр числа
3.	Катеты прямоугольного треугольника	Периметр и площадь треугольника
4.	Два действительных числа	Среднее арифметическое и среднее геометрическое этих чисел
5.	Два действительных числа	Среднее арифметическое этих чисел и среднее геометрическое их модулей
6.	Гипотенуза и катет прямоугольного треугольника	Второй катет и радиус вписанной окружности
7.	Внешний радиус кольца равен числу r (r>20)	Площадь кольца, внутренний радиус которого равен 20
8.	Длина окружности	Площадь круга
9.	Основания и высота равнобедренной трапеции	Её периметр и площадь
10.	Три положительных числа	Дробную часть среднего арифметического трех заданных положительных чисел
11.	Сторона равностороннего треугольника	Площадь и периметр треугольника
12.	Сторона квадрата. В квадрат вписана окружность	Площадь квадрата и окружности, вписанной в этот квадрат
13.	Радиус окружности. В окружность вписан квадрат	Найти площади окружности и квадрата
14.	Пятизначное число	Сумму всех цифр
15.	Четырёхзначное число	Записать цифры числа в обратной последовательности

1.9. Пример выполнения

Разберём выполнение работы на примере реализации калькулятора.

Первым делом нужно придумать макет, показывающий веб-интерфейс. На рисунке 3 представлено, как будет выглядеть калькулятор.

Рисунок 3 – Макет веб-интерфейса

Так же перед написанием кода нужно продумать логику работы программы. В два верхних поля будут вводиться числа, над которыми будут совершаться операции при нажатии одной из кнопок.

Хоть калькулятор на 4 действия и является простейшей задачей, но и тут есть нюансы, а именно необходимо перед проведением действий над числами нужно проверить, не пустые ли поля для ввода чисел и проверить то, чтобы в поля были введены именно числа, а не буквы или другие символы.

Так же для действия деление нужно проверять, чтобы во втором поле не был введён ноль.

Теперь можно приступать к написанию кода, но предварительно нужно создать необходимые папки и файлы.

Например, создаём папку с названием «Калькулятор», в которой будут находиться папки и файлы, указанные на рисунке 4.

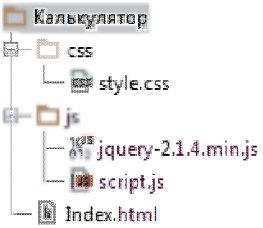


Рисунок 4 – Набор необходимых файлов

Теперь можно приступать к написанию кода.

В файле index.html используются 4 кнопки и три формы ввода данных. Текст кода содержащегося в index.html и некоторые комментарии представлены ниже:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Калькулятор</title>
  <link href="css/style.css" rel="stylesheet">
  <script src="js/script.js"></script>
  <script src="js/jquery-2.1.4.min.js"></script>
</head>
<body>
  <div class="center">
    <h4>Калькулятор</h4>
    <input id="dataA" class="center" type="text" maxlength="10" placeholder="Введите число А"> <!-- поле для первого числа -->
    <input id="dataB" class="center" type="text" maxlength="10" placeholder="Введите число Б"> <!-- поле для второго числа -->
    <br> <!-- перевод на следующую строку -->
    <br>
    <button type="submit" class="slojenie" onclick="slojenie()" >Сложение</button> <!-- кнопка сложения -->
```

```
        <button type="submit" class="vychitanie"
onclick="vychitanie()" >Вычнтание</button>  <!-- кнопка вычита-
ния -->
        <br>
        <br>
        <button type="submit" class="umnijenie"
onclick="umnijenie()" >Умножение</button>  <!-- кнопка умноже-
ния-->
        <button type="submit" class="delenie" onclick="delenie()"
>Деление</button>  <!-- кнопка деления -->
        <br>
        <br>
        <input id="result" class="center" type="text" value=""
readonly placeholder="Результат"> <!-- поле для вывода результата -
->
    </div>
</body>
</html>
```

Стили из файла style.css применяемые к html элементам представлены ниже:

```
.center {
    text-align: center;    /* отцентровать объект */
}
h4 {
    font-size: 30px;       /* размер шрифта */
    color: blue;           /* цвет */
}
#dataA {
    width: 150px;          /* ширина поля */
}
#dataB {
    width: 150px;
}
.slojenie {
    width: 100px;          /* ширина кнопки */
```

```
}
.vychitanie {
  width: 100px;
}
.umnijenie {
  width: 100px;
}
.delenie {
  width: 100px;
}
```

Остаётся только реализовать логику программы.

Чтобы при нажатии на кнопку происходило действие, нужно в тег button добавить onclick="slojenie()", где slojenie – это имя вызываемой функции.

В самом файле script.js прописывается function slojenie() { }, где в фигурных скобках прописываются необходимые действия.

Для начала нужно считать переменную из input. Для этого можно воспользоваться двумя способами:

– var a=document.getElementById('dataA').value; – считывание переменной с помощью JavaScript;

– var a=$("#dataA").val(); – считывание переменной с помощью библиотеки jQuery.

var – используется для объявления переменной.

Далее используем функцию parseFloat(), которая принимает строку в качестве аргумента и возвращает десятичное число. Эта функция, если будет введены какие-то символы отличные от цифр, вернёт NaN. Это поможет сделать простую проверку на правильность ввода символов. А именно применим такой код:

```
    if (isNaN(c) || isNaN(d)) {
    alert("Введите числа!");
}
```

После можно будет делать нужные операции над переменными.

Для того чтобы вывести результат, есть два варианта:

– document.getElementById("result").value = result; – с помощью JavaScript;

– $("#result").val(result); – с помощью библиотеки jQuery.

Полный текст скрипта представлен ниже:

```
function slojenie() {
   var a=document.getElementById('dataA').value;
   var b=document.getElementById('dataB').value;
   c=parseFloat(a,10);
   d=parseFloat(b,10);

   if (isNaN(c) || isNaN(d)) {
      alert("Введите числа!");
   }
   else  {
      result=c+d;
      document.getElementById("result").value = result;
   }
}
function vychitanie() {
   var a=document.getElementById('dataA').value;
   var b=document.getElementById('dataB').value;
      c=parseFloat(a,10);
   d=parseFloat(b,10);

   if (isNaN(c) || isNaN(d)) {
      alert("Введите числа!");
   }
   else  {
      result=c-d;
      document.getElementById("result").value = result;
   }
}
function umnijenie() {
```

```
        var a=$("#dataA").val();
        var b=$("#dataB").val();
        c=parseFloat(a,10);
    d=parseFloat(b,10);

        if (isNaN(c) || isNaN(d)) {
            alert("Введите числа!");
        }
        else  {
            result=c*d;
            $("#result").val(result);
        }
    }
    function delenie() {
        var a=$("#dataA").val();
        var b=$("#dataB").val();
        c=parseFloat(a,10);
        d=parseFloat(b,10);

        if (isNaN(c) || isNaN(d)) {
            alert("Введите числа!");
        }
        else if (d==0) {
            alert("Мы не в комплексной области! Здесь делить на 0
нельзя!")
        }
        else if (d!==0) {
            result=c/d;
            $("#result").val(result);
        }
    }
```

1.10. Контрольные вопросы

1. Что такое HTML?
2. Как подключить внешние каскадные таблицы стилей?
3. Есть ли разница между class и id?
4. Что такое родительский элемент?
5. Для чего нужны каскадные таблицы стилей?
6. Для чего используется JavaScript?
7. Зачем нужен <!DOCTYPE>?
8. Основные блоки html-документа?
9. Для чего используется библиотека jQuery?
10. Что такое DOM?

2. АДАПТИВНАЯ ВЕРСТКА

2.1. Основные понятия

Как уже упоминалось, фреймворком называют стандартизованный набор концепций, практик и критериев для работы с общим типом проблем, который может быть использован как руководство по помощи и решения новых проблем схожей природы.

Большинство вебсайтов имеют схожую (а иногда и одинаковую) структуры. Целью фреймворка является обеспечение общей структуры, чтобы разработчикам не приходилось делать всё сначала и можно было использовать разработанные куски кода. Таким образом фреймворки позволяют уменьшить объём работы и затрачиваемое время.

Практически любая веб-страница содержит множество похожих компонентов, которые встречаются и на других сайтах. Это меню, навигация, элементы форм, заголовки и др. Не говоря уже про многоколоночную вёрстку, без которой вообще сложно превратить картинку макета в готовую веб-страницу. Чтобы меньше писать кода и по максимуму задействовать уже готовые решения применяются специализированные библиотеки, их часто называют фреймворками. Одним из таких фреймворков для вёрстки является Bootstrap.

Bootstrap разработали Марк Отто и Якоб Торнтон, сотрудники Twitter, именно поэтому в названии фигурирует имя компании. Их цель понятна любому разработчику – создать единый стандартный набор инструментов для сотрудников компании, ускоряющий их работу [3].

На сегодняшний день Bootstrap давно перерос рамки одной компании, это открытый продукт применяемый веб-разработчиками для вёрстки сайтов во всём мире.

Фактически Bootstrap представляет собой конструктор, фрагменты которого вы включаете в свой проект при необходимости. Это уменьшает время разработки, потому что не требуется придумывать и писать их самостоятельно.

Bootstrap направлен на создание макета под разные устройства – ноутбуки, планшеты, смартфоны. При этом код пишется один,

а масштабирование в зависимости от ширины устройства берёт на себя фреймворк.

Компоненты библиотеки написаны и протестированы с учётом работы разных браузеров. Это гарантирует, что макет будет выглядеть одинаково независимо от выбранного браузера.

Чтобы использовать библиотеку в своей работе, требуется обладать минимальными знаниями по HTML, CSS и JavaScript. Это позволяет создавать эффектные сайты даже начинающим разработчикам.

Bootstrap не просто вставляет какие-то элементы на страницу, но сразу же устанавливает их оформление и взаимодействие с пользователем через JavaScript. Вы получаете полностью работающий набор компонент, который достаточно добавить и настроить под себя.

Недостатки, конечно же, тоже имеются и они вытекают в основном из универсальности системы.

Во-первых, файлы библиотеки, даже сжатые, занимают довольно много места и увеличивают нагрузку на сервер. С повышением опыта нужно загружать и устанавливать только требуемые компоненты Bootstrap, сокращая тем самым объём файлов [4].

Во-вторых, предлагаемые в библиотеке стили могут не подходить под дизайн разрабатываемого сайта и придётся много переделывать. В таких случаях, как известно, проще написать всё самому с нуля.

В общем, Bootstrap годится для типовых сайтов, дизайн которых ориентирован на библиотеку. А сайтов, которые предлагают темы и шаблоны, в том числе бесплатные, в последнее время родилось довольно много. Достаточно поискать по ключевым словам «BootstrapTheme».

Для начала необходимо скачать Bootstrap с сайта разработчика по следующему адресу:

http://getbootstrap.com/getting-started/#download

Это полная версия, содержащая все необходимые скрипты и стили. Внутри находится три папки: css, fonts и js.

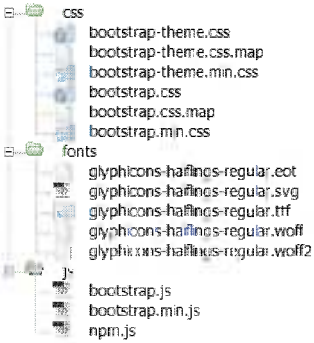


Рисунок 5 – Файлы содержащиеся в Bootstrap

В папках со стилями и скриптами приложено две версии файлов – исходная и компактная (в имени содержится min). Компактная отличается лишь размером файла и снижением читаемости кода. Лучше всего на рабочий сайт добавлять именно эту версию, так мы чуть ускорим загрузку веб-страниц.

Копируем все папки в наш проект и в корне создаём index.html. В итоге структура нашего проекта будет выглядеть следующим образом.

Рисунок 6 – Структура проекта

Содержимое index.html включает в себя ссылку на стилевой файл bootstrap.css и внизу страницы мы вызываем bootstrap.js. Больше пока ничего не нужно.

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Bootstrap</title>
<link href="css/bootstrap.css" rel="stylesheet">
```

```
</head>
<body>
<h1>Привет, мир!</h1>
<scriptsrc="js/bootstrap.js"></script>
</body>
</html>
```

Если в процессе работы потребуется переопределить стили каких-то элементов, то вы можете подключить ещё один собственный стилевой файл и в нём задать необходимые свойства

Адаптивная вёрстка в фреймворке Bootstrap начинается с подключения файлов Bootstrap к нашему index.html.

Неотъемлемой частью адаптивной вёрстки является использование стиля container для общего блока div.

```
<div class="container">…</div>
```

Container отвечает за выравнивание блока по центру с отступами слева и справа по 15 px. Контейнер будет иметь фиксированную ширину, изменение которой происходят в следствии изменения ширены окна браузера (таблица 4).

Таблица 4

Ширина контейнера	Ширина окна веб-клиента
1170px	Свыше или равна 1200px
970px	Свыше или равна 992px и меньше 1200px
750px	Свыше или равна 768px и меньше 992px
Динамическая ширина (ширина равна ширине рабочей области окна веб-клиента)	Меньше 768px

Рисунок 7 – Наглядное представление стиля container

Или же можно использовать стиль container-fluid, который не имеет фиксированной ширины и из свойств имеет только отступы с лева и права по 15px.

Рисунок 8 – Наглядное представление стиля container-fluid

В Bootstrap для адаптивной вёрстки используется разделение рабочей области на 12 колонок. Для их правильного функционирования нужно обязательно использовать стиль row.

…

Ряд принимает такую же ширину, как и контейнер, но имеет отрицательный отступ слева и справа в 15 px.

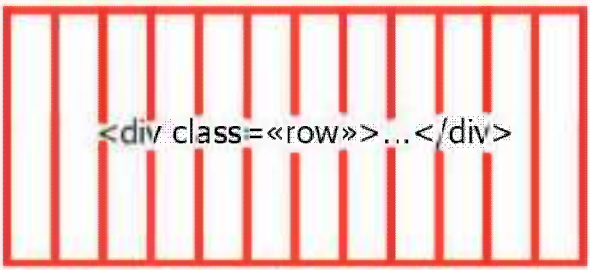


Рисунок 9 – Блоки в одном ряду

Внутри ряда будут размещаться блоки. В общем виде это выглядит так:

```
<div class="row">
    <div class="col-*-*">...</div>
    <div class="col-*-*">...</div>
    <div class="col-*-*">...</div>
</div>
```

Ширина каждого блока указывается в относительном формате посредством выбора необходимого числа колонок. То есть минимальная ширина блока равняется одной колонке, а максимальная ширина – двенадцати колонкам. Например, если нам нужно в одном ряду иметь три блока, из которых первый занимает половину всей области, а второй и третий блок равны друг другу и вместе занимают оставшуюся половину, то необходимо присвоить первому блоку ширину в шесть колонок, а второму и третьему по три колонке каждому. В сумме выходит 6+3+3=12 колонок.

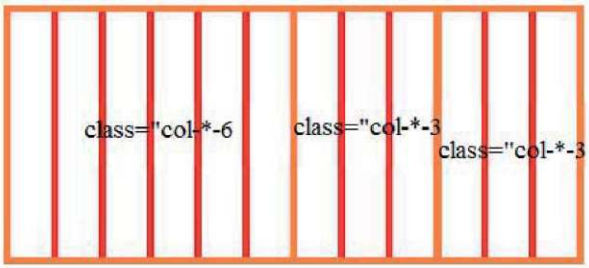


Рисунок 10 – Пример 3 блоков в одном ряду

Теперь разберёмся со звёздочками после col. Так как существует большое количество устройств с различным размером экранов, в Bootstrap они были разделены на 4 группы. Собственно вместо звёздочки нужно будет указывать необходимую группу. Рассмотрим эти группы:

– col-xs-* – применяется для создание сетки с маленькими экранами, то есть телефоны;

– col-sm-* – для устройств чуть большей ширины экрана, чем у телефона, например планшет;

– col-md-* – средних размеры экранов, например, нетбуки;

– col-lg-* – для больших экранов.

Таблица 5 – данные по этим группам

Вид системы сеток	Маленький экран(<768px)	Небольшой экран (≥768px и <992)	Средний экран (≥992px и <1200px)	Большой экран (≥1200px)
Фиксированный макет (class="container")	Соответствует 100% ширине веб-клиента	750px	970px	1170px
Резиновый макет (class="container-fluid")	Соответствует 100% ширине экрана веб-клиента			
Префикс класса	class="col-xs-*"	class="col-sm-*"	class="col-md-*"	class="col-lg-*"
Максимальная ширина колонки Bootstrap для фиксированного макет (class="container")	Ширина веб клиента делится на 12	62px (750px / 12)	81px (970px / 12)	97px (1170px / 12)
Максимальная ширина колонки для резинового макета (class="container-fluid")	Ширина веб клиента делится на 12			
Внутренние отступы	15px по краям			

Важная особенность, если использовать класс class="col-sm-*", то он будет применяться не только к небольшим экранам, но и к средним и большим экранам, но это действует в том случае, если не было указано "col-md-*" и "col-lg-*". То есть если в ряду будет один блок для всех экранов, то достаточно написать ... < /div>, а не ... < /div>.

Так же бывает необходимость убирать некоторые блоки, когда окно браузера становится меньше необходимой величины. Для этого используется класс hidden-*, где вместо звёздочки пишется префикс класса.

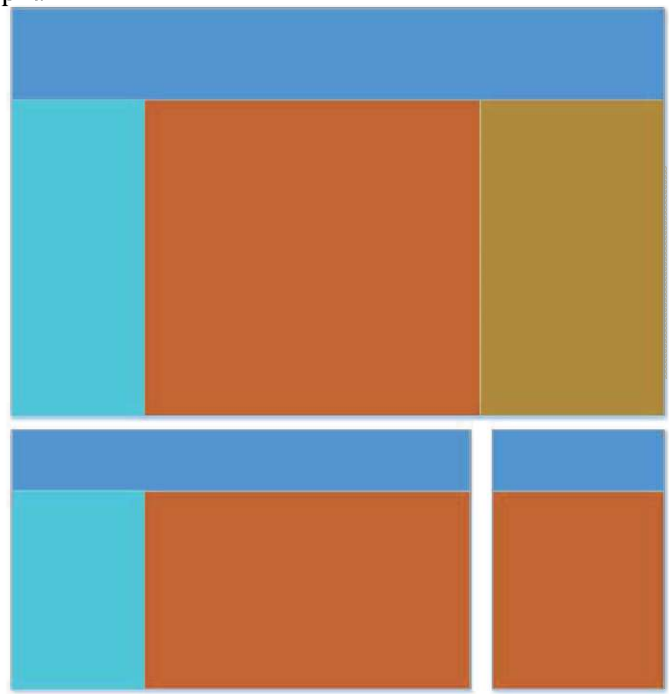
2.2. Индивидуальные задания

Сделать адаптивную вёрстку для трёх размеров экрана:

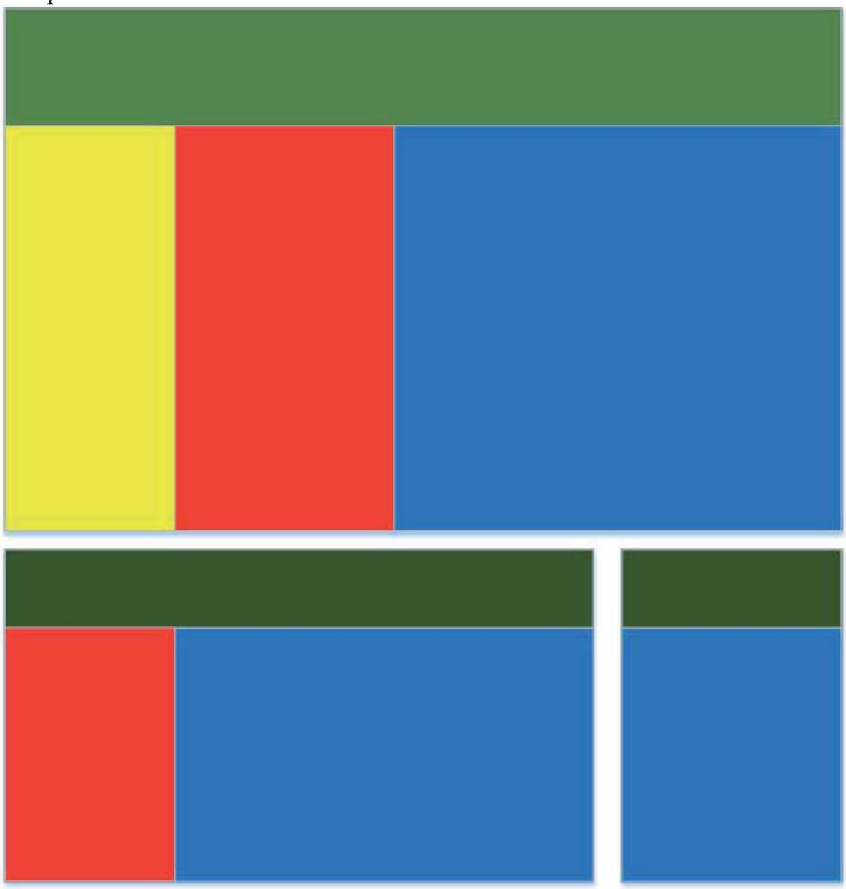
- средний и большой экраны
- небольшой экран
- маленький экран

Варианты заданий представлены ниже:

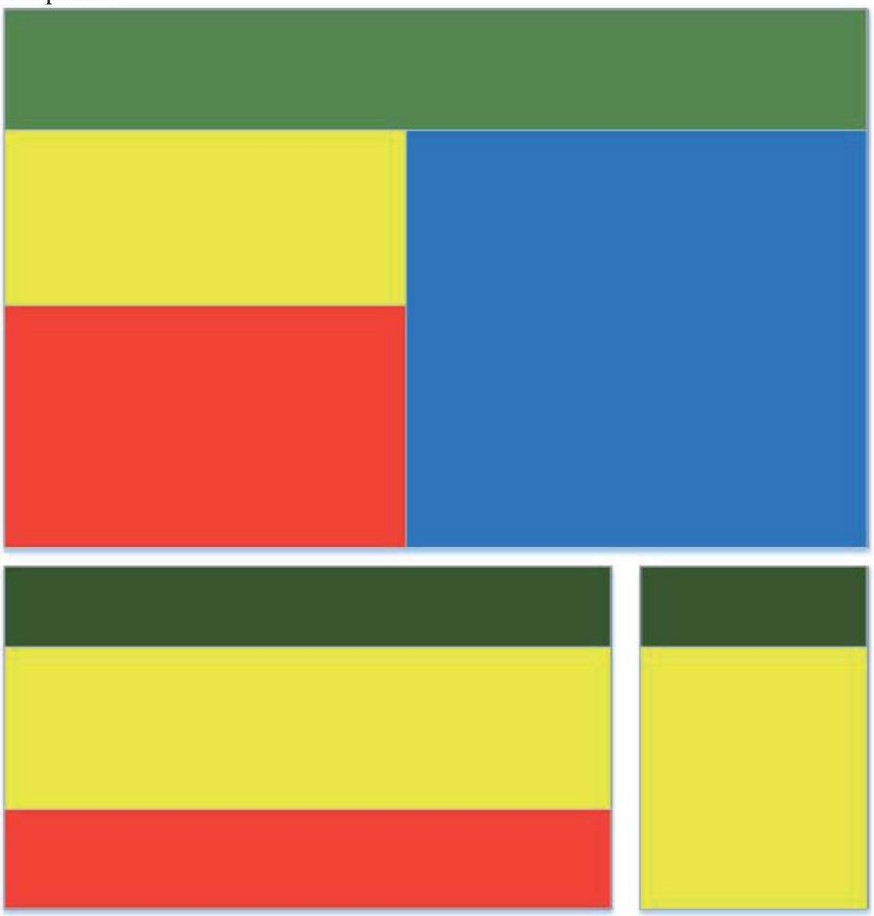
1 вариант



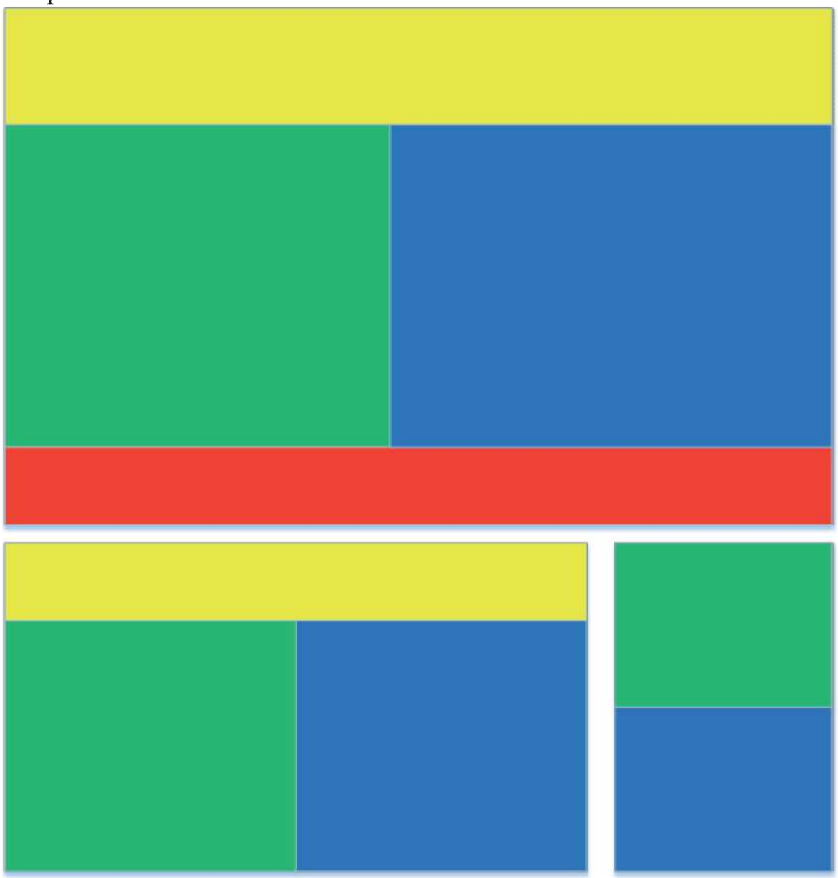
2 вариант



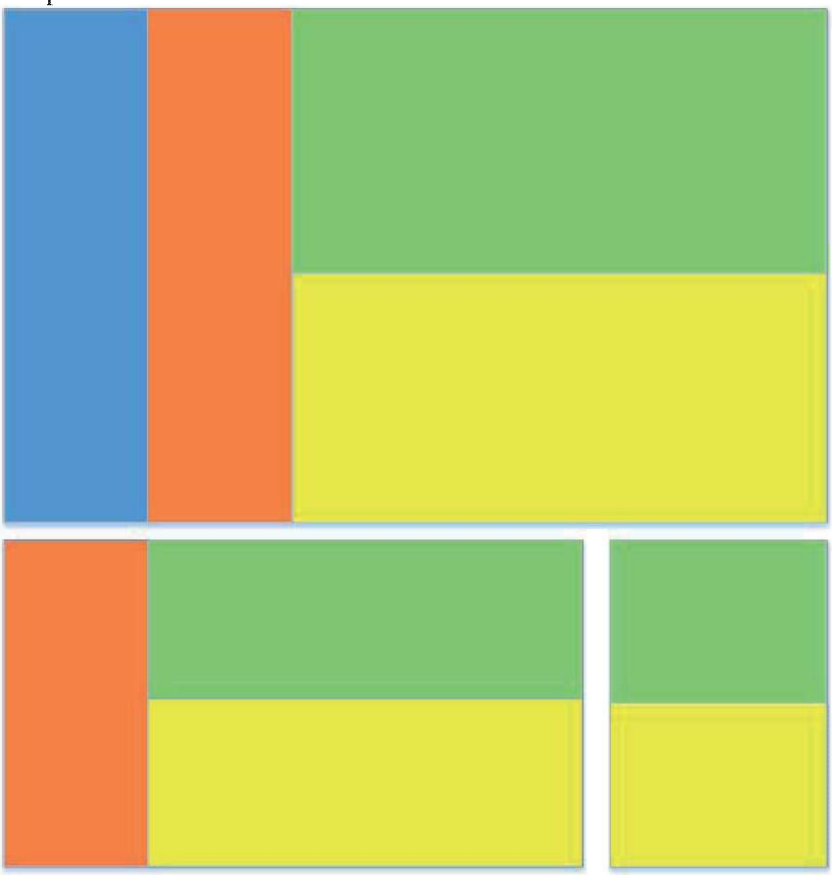
3 вариант



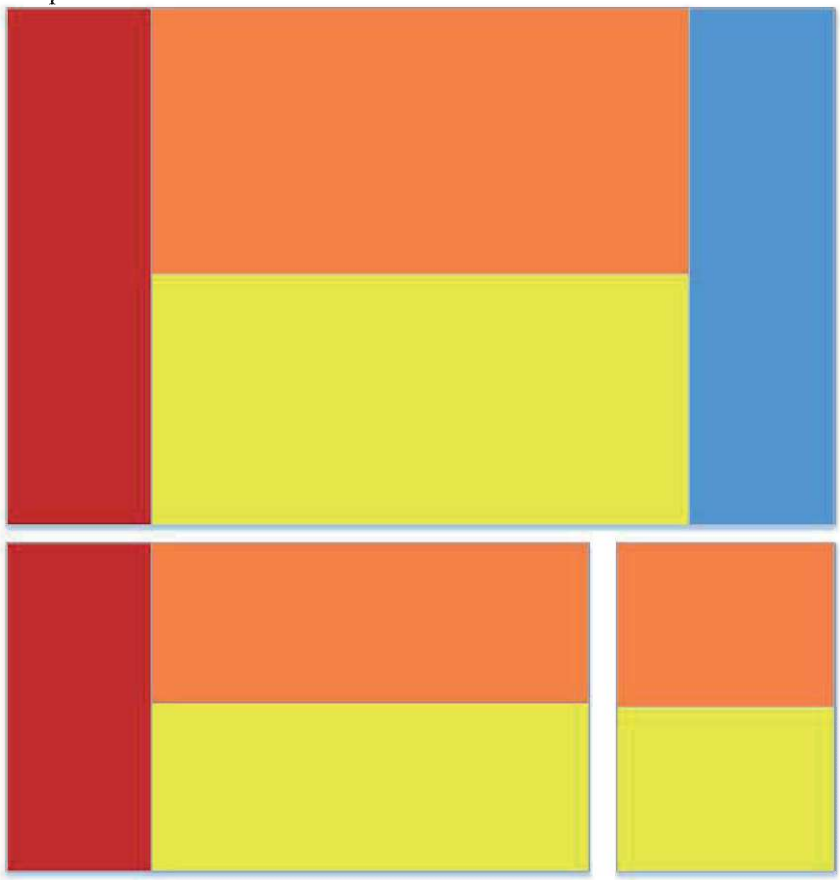
4 вариант



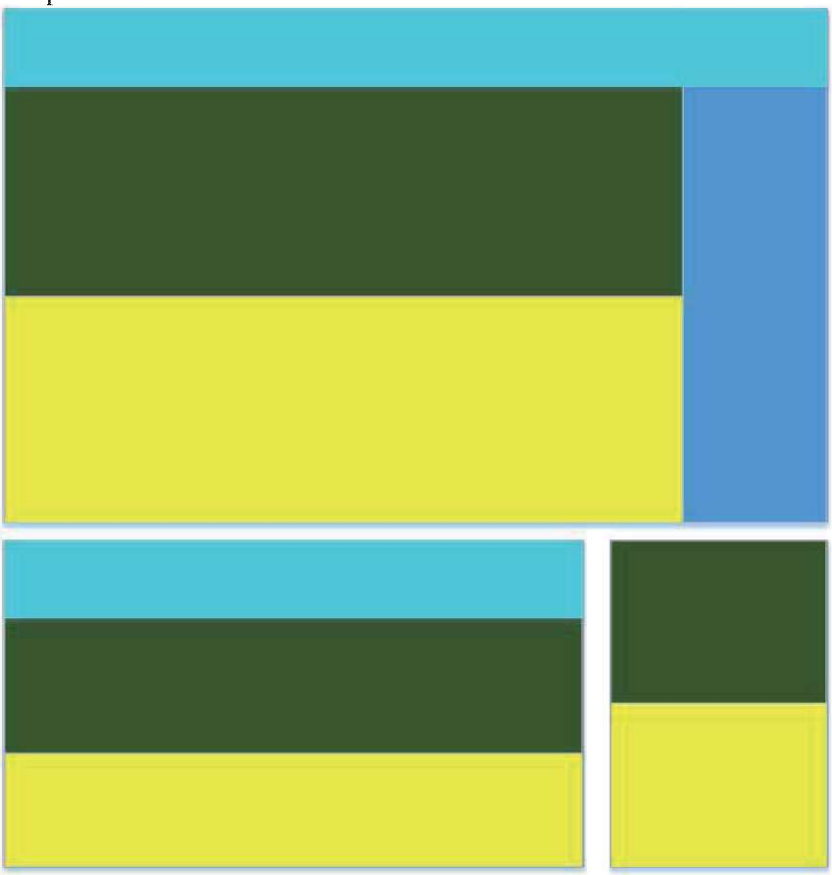
5 вариант



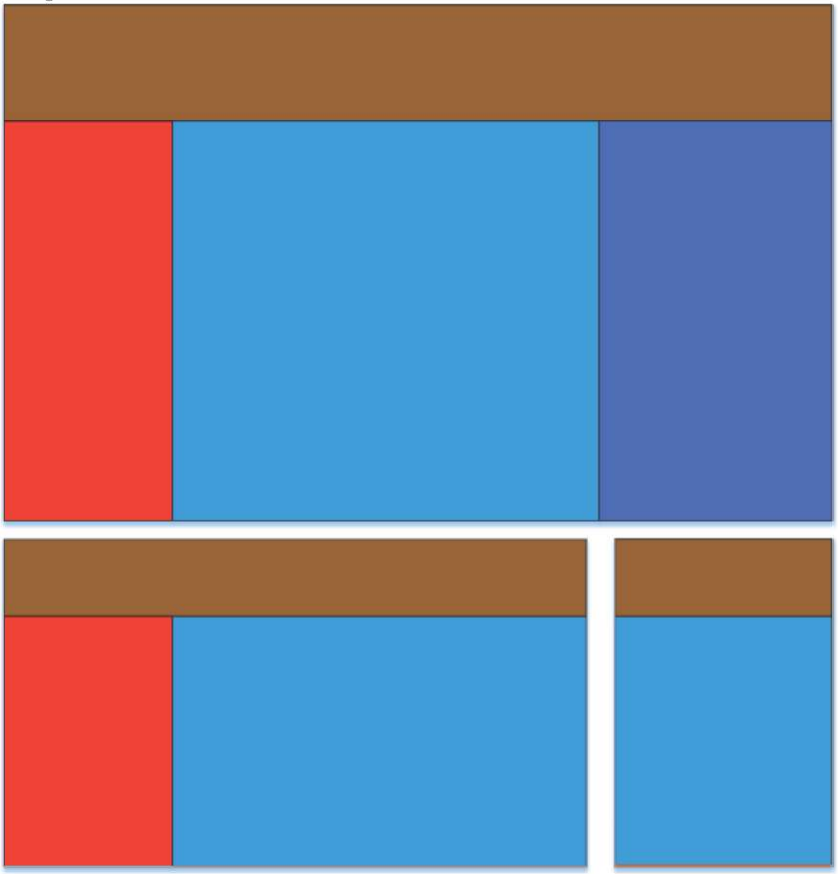
6 вариант



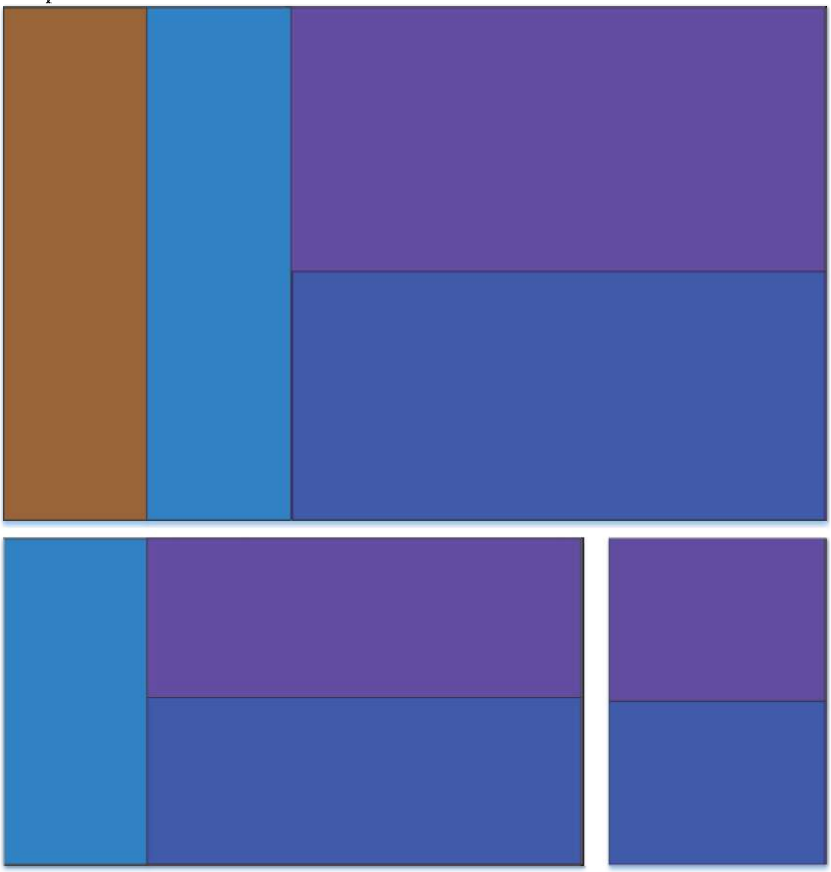
7 вариант



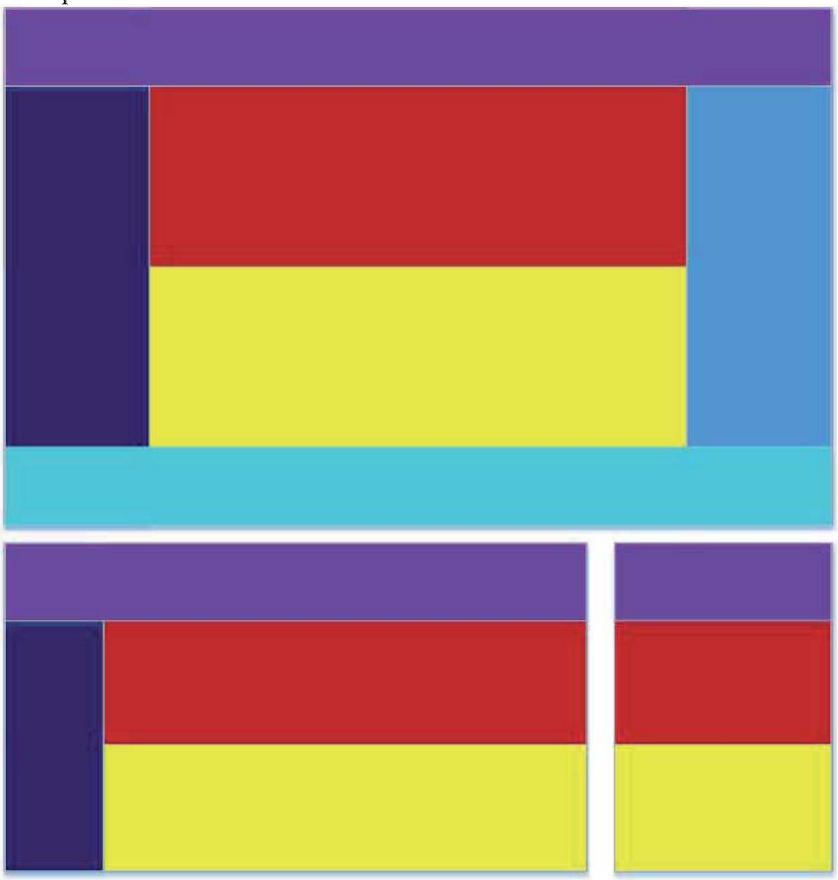
8 вариант



9 вариант



10 вариант



2.3. Пример выполнения

Рассмотрим на простом примере, как работает адаптивная вёрстка. Для этого сначала определимся, из каких блоков будет состоять сайт. Будем использовать четыре блока различной ширины. Для средних и больших экранов дизайн будет одинаковым.

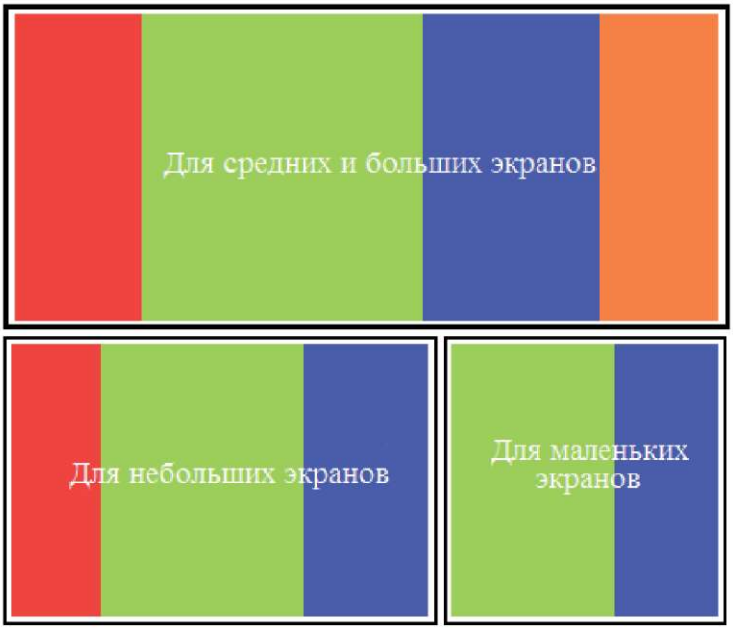


Рисунок 11 – Макет адаптивной вёрстки для разных устройств

Код выполнения данного примера для index.html представлен ниже:

```
<!doctype html>
<html>
<head>
   <meta charset="utf-8">
   <title>Bootstrap</title>
   <link href="css/bootstrap.css" rel="stylesheet">
   <link href="css/style.css" rel="stylesheet">
```

```
    <script src="js/bootstrap.js"></script>
</head>
<body>
<div class="container-fluid">
    <div class="row">
        <div class="col-md-2 col-sm-2 hidden-xs red"></div>
        <div class="col-md-5 col-sm-6 col-xs-8 green"></div>
        <div class="col-md-3 col-sm-4 col-xs-4 blue"></div>
        <div class="col-md-2 hidden-sm hidden-xs orange"></div>
    </div>
</div>
</body>
</html>
```

Код style.css

```
.red {
    background-color: red;
    height: 100vh;
}
.green {
    background-color: greenyellow;
    height: 100vh;
}
.blue {
    background-color: blue;
    height: 100vh;
}
.orange {
    background-color: orange;
    height: 100vh;
}
```

2.4. Контрольные вопросы

1. Что такое фреймворк?
2. Для чего используется адаптивная вёрстка?
3. Зачем нужен Bootstrap?
4. Что делать стиль hidden-xs?
5. В чём разница между стилями container и container-fluid?
6. Какие файлы содержит Bootstrap?
7. Зачем среди файлов Bootstrap есть файлы, в именах которых присутствует min?
8. На какое количество столбцов можно разбить рабочую область?

3. ОСНОВЫ ANGULARJS

3.1. Основные понятия

AngularJS – JavaScript-фреймврк с открытым исходным кодом. Предназначен для разработки одностраничных приложений. Его цель – расширение браузерных приложений на основе MVC шаблона, а также упрощение тестирования и разработки [5].

AngularJS является структурной основой для динамических веб-приложений. Он позволяет использовать HTML в качестве языка шаблонов и позволяет расширять синтаксис HTML, чтобы четко и лаконично выражать компоненты приложения. Увязка данных AngularJS и инъекция зависимостей устраняют большую часть кода, который в противном случае вам приходилось писать. И все это происходит в браузере, что делает его идеальным партнером любой серверной технологии.

AngularJS – это то, что было бы HTML, если бы оно предназначалось для приложений. HTML – отличный декларативный язык для статических документов. Он не содержит много способов создания приложений, и в результате создание веб-приложений - это упражнение в *том, что мне нужно сделать, чтобы обмануть браузер, делая то, что я хочу?*

3.2. Основные директивы

AngularJS преподает новый синтаксис браузера через конструкцию, которую мы называем *директивами*.

На директивах держится практически вся декларативная часть AngularJS. Именно они используются для обогащения синтаксиса HTML. В процессе компиляции DOM директивы берутся из HTML и исполняются. Директивы могут добавить какое-то новое поведение и/или модифицировать DOM. В стандартную поставку входит достаточно большое количество директив для построения веб приложений. Но ключевой особенностью является возможность разработки своих директив, за счет чего HTML может быть превращен в DSL.

Директивы именуются с помощью lowerCamelCase, например, ngBind. При использовании директиву необходимо именовать в нижнем регистре с использованием в качестве разделителя одного

из спец символов: :, -, или _. По желанию для получения валидного кода можно использовать префиксы x- или data-. Примеры: ng:bind, ng-bind, ng_bind, x-ng-bind и data-ng-bind.

Директивы могут использоваться как элемент (<my-directive></my-directive>), атрибут (), в классе () или в комментарии (). Это зависит от того, как конкретная директива была разработана:

ng-app – объявляет элемент корневым для приложения.

ng-bind – автоматически заменяет текст HTML-элемента на значение переданного выражения.

ng-model – то же что и ng-bind, только обеспечивает двустороннее связывание данных. Изменится содержимое элемента, ангулар изменит и значение модели. Изменится значение модели, ангулар изменит текст внутри элемента.

ng-class – определяет классы для динамической загрузки.

ng-controller – определяет JavaScript-контроллер для вычисления HTML-выражений.

ng-repeat – создает экземпляр для каждого элемента из коллекции.

ng-show и ng-hide – показывает или скрывает элемент в зависимости от значения логического выражения.

ng-switch – создает экземпляр шаблона из множества вариантов, в зависимости от значения выражения.

ng-view – базовая директива, отвечает за обработку маршрутов, которые принимают JSON перед отображением шаблонов, управляемых указанными контроллерами.

$scope – это объект, имеющий отношение к модели в приложении. Он является контекстом выполнения для выражений. Scope-ы выстраиваются в иерархическую структуру, похожую на DOM. При этом они наследуют свойства от своих родительских $scope.

$scope являются как бы «клеем» между контроллером и представлением. В процессе выполнения фазы связывания шаблона директивы устанавливают наблюдение ($watch) за выражениями в рамках scope. $watch дает директивам возможность реагировать на изменения для отображения обновленного значения или каких-либо других действий. И контроллеры, и директивы имеют ссылку на $scope, но не имеют ссылок друг на друга. Так контроллеры изо-

лируются от директив и от DOM-а. За счет этого возрастают возможности по тестированию приложения.

3.3. Безопасность приложений

В любом веб-приложении необходимо предусматривать меры за-щиты конфиденциальной информации от несанкционированного доступа. Единственным, по-настоящему безопасным местом в таких приложениях является сервер. За его пределами необходимо учитывать возможность взлома программного кода и вследствие этого предусматривать проверки в точке, где данные поступают или покидают сервер. Необходимые меры безопасности, которые следует предпринять и на стороне сервера, и на стороне клиента, перечисленные ниже:

– предотвращение на стороне сервера несанкционированного доступа к данным и разметке HTML;

– шифрование трафика с целью исключить перехват пакетов;

– предотвращение атак типа «межсайтовый скриптинг» (Cross-Site scripting, XSS) и «подделка межсайтовых запросов» (Cross-Site Request Forgery, XSRF);

– блокирование попыток внедрения кода JSON.

На стороне сервера меры по обеспечению безопасности должны осуществляться всегда, но это не значит, что они не нужны на стороне клиента – мы обязаны предусматривать подобные меры и в пользовательском интерфейсе приложения, чтобы явно показать, что пользователь может иметь доступ только к определенному кругу функциональных возможностей, соответствующих его привилегиям. Кроме того, мы обязаны реализовать ясную процедуру аутентификации, не мешающую пользователю взаимодействовать с приложением. Важными являются вопросы обеспечения безопасности с применением средств фреймворка AngularJS, в том числе:

– различия в обеспечении безопасности полнофункциональных клиентских приложений и более традиционных приложений, опирающихся на хранение информации на стороне сервера;

– обработка ошибок авторизации на сервере посредством перехвата HTTP-ответов;

– ограничение доступа к разделам путем обеспечения безопас-ности маршрутов;

Предотвращение перехвата cookie («атака через посредника»).

Всякий раз, когда между клиентом и сервером осуществляется передача данных по протоколу HTTP, существует вероятность вмешательства третьей стороны с целью прочитать конфиденциальную информацию или, хуже того, cookies авторизации, чтобы перехватить управление сеансом и получить возможность обращаться к серверу от вашего имени. Нападения такого вида часто называют «атака через посредника» («man-in-the-middle»). Самый простой способ предотвращения таких нападений – использование протокола HTTPS вместо HTTP [6].

Шифруя соединение посредством протокола HTTPS , мы исключаем возможность чтения конфиденциальных данных посторонними лицами, оказавшимися между сервером и клиентом, а также не позволяем неавторизованным пользователям получать блоки cookies аутентификации, с помощью которых они могли бы перехватить управление сеансом.

Нападения вида «межсайтовый скриптинг» (Cross-Site Scripting, XSS) заключаются во внедрении злонамеренного сценария в веб-страницу при просмотре ее другим пользователем. Наибольший вред от таких нападений может быть нанесен, если внедренный сценарий сможет обращаться к серверу, потому что сервер будет считать эти обращения аутентифицированными и выполнять их.

Существует большое разнообразие нападений XSS. Чаще всего используется разновидность, основанная на отображении получаемого из сети содержимого без надлежащего экранирования, предотвращающего интерпретацию специально подготовленной разметки HTML.

3.4. Предотвращение подделки межсайтовых запросов

В любом приложении, где сервер оказывает доверие зарегистрированному пользователю и позволяет выполнять некоторые операции на сервере, полагаясь на это доверие, существует вероятность, что другие сайты смогут получить доступ к этим операциям и выполнять их от вашего имени. Эта разновидность нападений называется «подделка межсайтовых запросов» (Cross-Site Request Forgery, XSRF). Если пользователь посетит злонамеренный сайт, когда он уже прошел процедуру аутентификации на защищенном сайте, веб-страница со злонамеренного сайта сможет послать защищенному сайту запрос, поскольку в данный момент вы считаетесь аутентифицированным пользователем.

Такие нападения часто реализуются в виде мошеннического атрибута src в теге <img>, который пользователь может загрузить по неосторожности при переходе на злонамеренную страницу, не закрывая сеанс работы с защищенным сайтом. Когда браузер попытается загрузить изображение, он фактически выполнит запрос к защищенному сайту.

<imgsrc="http://my.securesite.com/createAdminUser?username=badguy">;

Чтобы устранить эту проблему, сервер может передавать браузеру секретный ключ, доступный только сценарию на JavaScript, выполняющемуся в браузере, и недоступный в атрибуте src. При обращении к серверу этот ключ должен включаться в заголовки запросов, чтобы подтвердить аутентичность пользователя.

Служба $http уже реализует это решение для защиты от нападений подобного рода. Чтобы его активизировать, необходимо на стороне сервера обеспечить передачу в сеансовом блоке cookie параметра с именем XSRF-TOKEN, в ответ на первый GET-запрос приложения. Значение этого параметра должно быть уникальным для данного сеанса.

На стороне клиента служба $http будет извлекать этот ключ из cookie и добавлять его в каждый HTTP-запрос в виде заголовка X-XSRF-TOKEN. Сервер должен проверять ключ в каждом запросе и блокировать доступ, если он окажется недействительным. Разработчики AngularJS рекомендуют использовать этот ключ в допол-

нение к cookie аутентификации, чтобы повысить защищенность приложения.

Служба security – это созданный нами компонент, реализующий основной прикладной интерфейс для управления открытием и закрытием сеанса, и для получения информации о текущем пользователе.

Эту службу можно внедрять в контроллеры и директивы, которые в свою очередь могут добавлять перечисленные ниже свойства и методы в объект контекста, чтобы обеспечить доступ к ним из шаблонов.

currentUser: свойство с информацией о текущем, аутентифицированном пользователе;

getLoginReason(): этот метод возвращает локализованное сообщение, объясняющее необходимость аутентификации, например: «Текущий пользователь не авторизован».

showLogin(): этот метод отображает форму аутентификации.

Вызывается в ответ на щелчок на кнопке Login(Войти), а также при получении HTTP-ответа HTTP 401 Unauthorized;

login(email, password): этот метод отправляет указанные параметры на сервер для аутентификации. Вызывается, когда пользователь отправляет форму аутентификации. В случае успеха форма закрывается и повторяется попытка выполнить запрос, вызвавший появление этой формы.

logout(redirectTo): этот метод закрывает текущий сеанс работы с пользователем и выполняет переход по указанному маршруту. Вызывается в ответ на щелчок на кнопке Log out (Выйти).

cancelLogin(redirectTo): этот метод прерывает попытку открыть сеанс, аннулирует все неавторизованные запросы, приведшие к появлению формы аутентификации, и выполняет переход по указанному маршруту. Вызывается, когда пользователь закрывает форму аутентификации или щелкает на кнопке Cancel(Отмена).

3.5. Предотвращение переходов по защищенным маршрутам

Предотвращение доступа к защищенным маршрутам с использованием кода на стороне клиента не обеспечивает необходимый уровень безопасности. Единственный надежный способ, гарантирующий невозможность попадания неавторизованных пользователей в защищенные разделы приложения, требует перезагрузки страницы, чтобы дать серверу возможность отказать в доступе. Но перезагрузка страницы не является идеальным вариантом, потому что сводит на нет все достоинства полнофункциональных клиентских приложений. На деле, так как мы можем защитить данные, отображаемые перед пользователем, не так важно блокировать доступ к маршрутам с переадресацией на сервер. Вместо этого можно просто запретить переход неавторизованного пользователя к защищенному маршруту на стороне клиента, в момент изменения маршрута [7].

AngularJS можно скачать на официальном сайте angularjs.org на главной странице, нажав на кнопку download. Скачанную библиотеку помещаем в папку libraries. Создадим файл Helloworld.html. Нам необходимо подключить библиотеку angular с помощью элемента Script:

<script src="./js/angular.js"></script>

Далее размещаем свой элемент Script, в котором будет происходить настройка и подготовка нашего кода для работы.

Создадим переменную:

var *model* = "Hello Word";

Ниже создадим модуль. Библиотека angular.js представлена одним глобальным объектом angular. Если мы хотим получить что-то от библиотеки angular.js, нам необходимо обратиться к этому объекту и вызвать на нем специальный метод. AngularJS приложения строятся из строительных блоков, которые называются модулями. Когда мы вызываем метод module, мы должны передать ему несколько параметров: первый – имя модуля, второй – зависимости модуля. Если не указать второй параметр, то это означает, что мы хотим найти в JavaScript приложении модуль с названием, который мы указали, как первый параметр.

В нашем случает мы создали модуль:

var *HelloWorldApp* = angular.*module*("helloworldApp", []);

В этот модуль потом помещаются все строительные блоки AngularJS приложения: контроллеры, директивы, фильтры, сервисы и так далее. Для удобства модуль сохранили в переменную. Для того чтобы наше приложение начало работать именно с этого модуля, в элементе html мы добавляем директиву **ng-app:**

<html ng-app="HelloWordApp">.

Это означает, что наше приложение, которое будет работать на это html страничке , оно представлено модулем HelloWordApp.

Теперь нам нужно создать полезную функциональность приложения. Контроллеры создаются следующим образом:

HelloWorldApp.controller("HellowWorldCtrl", function($scope){
$scope.message = model;
});

Мы вызываем метод controller на уже существующем модуле. Первым аргументом мы передаем имя контроллера,а вторым аргументом функцию. Контроллер у нас называется HellowWorldCtrl, после запятой — поведение(функция), которое определяет наш контроллер. HellowWorldCtrl будет использоваться в разметке для того, чтобы определить какая часть пользовательского интерфейса контролируется этим контроллером. Далее в элементе body должна присутствовать еще одна директива **ng-controller**, в которой указывается имя ранее созданного контроллера. Благодаря этому имени AngularJS приложение будет понимать, что те действия и те данные, которые будут находится в элементе body, адресовываются HellowWorld контроллеру.

Таким образом мы получили следующий код:

```
<html ng-app="HelloWordApp">
<head>
  <meta charset="utf-8">
  <title>HelloWord</title>

  <script src="./js/angular.js"></script>
  <script>
    //Модель
    var model = "Hello Word";
```

```
        var helloworldApp = angular.module("helloworldApp", []);
        //Контроллер
        helloworldApp.controller("HellowWorldCtrl", function($scope) {
            $scope.message = model;
        });
    </script>
</head>

<body ng-controller="HellowWorldCtrl">
    <h1> {{message}}</h1>
</body>
</html>
```

На странице у нас изображается:

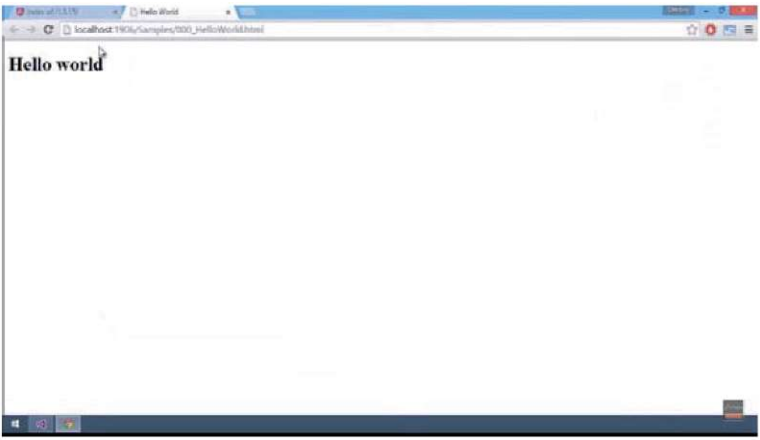


Рисунок 12 – Это пример простейшего angular js приложения

3.6. Индивидуальные задания

Разработать простейшее приложение при помощи AngularJS, которое будет выполнять следующие действия:

Таблица 6

№	Дано	Найти
0.	Число	Простое оно или составное
1.	Две стороны треугольника и угол между ними	Длину третьей стороны и площадь этого треугольника
2.	Четырёхзначное число	Произведение всех цифр числа
3.	Катеты прямоугольного треугольника	Периметр и площать треугольника
4.	Два действительных числа	Среднее арифметическое и среднее геометрическое этих чисел
5.	Два действительных числа	Среднее арифметическое этих чисел и среднее геометрическое их модулей
6.	Гипотенуза и катет прямоугольного треугольника	Второй катет и радиус вписанной окружности
7.	Внешний радиус кольца равен числу r (r>20)	Площадь кольца, внутренний радиус которого равен 20
8.	Длина окружности	Площадь круга
9.	Основания и высота равнобедренной трапеции	Её периметр и площадь
10.	Три положительных числа	Дробную часть среднего арифметического трех заданных положительных чисел
11.	Сторона равностороннего треугольника	Площадь и периметр треугольника
12.	Сторона квадрата. В квадрат вписана окружность	Площадь квадрата и окружности, вписанной в этот квадрат
13.	Радиус окружности. В окружность вписан квадрат	Найти площади окружности и квадрата
14.	Пятизначное число	Сумму всех цифр
15.	Четырёхзначное число	Записать цифры числа в обратной последовательности

3.7. Пример выполнения задания

Требуется создать страницу, которая должна включать в себя возможность проверки вводимого числа на простоту

Для наглядности возможностей Angular.js сначала выполним это задание без использования этой библиотеки.

Напишем простой код для визуальной оболочки:

Код Index.html:

```
<html>
<head>
  <meta charset="utf-8">
  <title>Проверка числа</title>
  <link href="css/style.css" rel="stylesheet">
  <script src="js/script.js"></script>
  <script src="js/jquery-2.1.4.min.js"></script>
</head>
<body>
  <div class="center">
    <h4>Проверка числа</h4>
    <input id="data" class="center" type="text" maxlength="10"
placeholder="Введите число">  <!-- поле для ввода числа -->
    <button type="submit" class="proverka" onclick="proverka()"
>Проверка</button>          <!-- кнопка проверки -->
    <br>
    <br>
    <input id="result" class="center" type="text" value="" readonly
placeholder="Здесь будет результат"> <!-- поле для вывода результата -->
  </div>
</body>
</html>
```

Код, выполняющий проверку числа на простоту:

```
    function proverka() {
  var number=$("#data").val();
  var root = Math.sqrt(number);
```

```
var root_int = Math.round(root);
var leftover;
var is_simple = 1;
   for (var i = 2; i <= root_int; i++) {
      leftover = number % i;
      if (leftover == 0) {
         is_simple = 0;
         break;
      }
   }
   if (is_simple == 1) {
      result = "Число " + number + " простое";
   }
   else {
      result = "Число " + number + " составное";
   }
   $("#result").val(result);
}
```

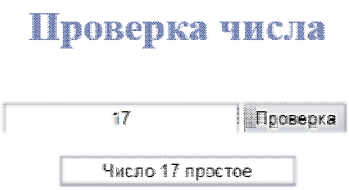


Рисунок 13 – Результат выполнения кода программы

Используя библиотеку Angular.js, предполагается, что пользователю предлагается вести число, и сразу же появляется информация является ли число простым или составным.

Первом делом, на странице lab.html требуется объявить контроллер lab, который создан в файле app.js Это достигается добавлением к первому использованному тегу директивы ng-controller, которая определяет JavaScript-контроллер для вычисления HTML-выражений, в данном случае контроллер lab.

```
    <body ng-controller="lab">
  <form role = "form">
    <div class = "form-group">
      <label for = "number" id = "number1">Введите число</label>
      <input ng-model="number" type = "number" class = "form-
control" id = "number" placeholder = "Введите число для проверки">
    </div>
  </form>
  <span ng-bind="Func()"></span>
  <div>
    {{result}}
  </div>
</div>
    </body>
```

С помощью фигурных скобок {{}} можно вызывать значение переменных на страницу. Тег <input> является одним из разносторонних элементов формы и позволяет создавать разные элементы интерфейса и обеспечить взаимодействие с пользователем. Главным образом <input> предназначен для создания текстовых полей, различных кнопок, переключателей и флажков. С помощью директивы ng-model и ее значения, заключенных в надстрочные запятые, мы показываем браузеру, что в поле ввода input вводится переменная number, которая принимается на вход контроллера. Атрибут type сообщает браузеру, к какому типу относится элемент формы. Атрибут placeholder своим значением отвечает за то, что будет написано в поле ввода, пока пользователь не внес свои данные. Директива ng-bind - автоматически заменяет текст HTML-элемента на значение переданного выражения, а с помощью {{result}} выводится значение переменной result нашего контроллера:

```
    app.controller("lab", ['$scope',function($scope){
  $scope.title = 'Введите число';
  $scope.number = '';
  $scope.Func=function () {
    var number = $scope.number;
    var root = Math.sqrt(number);
```

```
    var root_int = Math.round(root);
    var leftover;
    var is_simple = 1;
    for (var i = 2; i <= root_int; i++) {
      leftover = number%i;
      if (leftover == 0) {
        is_simple = 0;
        break;
      }
      }
    if (is_simple == 1) {
      $scope.result="Число " + number + " простое";
    }
    else {
      $scope.result="Число " + number + " составное";
    }
  }
}]);
```

3.8. Контрольные вопросы

1. Назначение директивы ng-app.
2. Что определяет второй аргумент контроллера?
3. Что такое scope?
4. Какие меры безопасности необходимо предпринимать на стороне сервера и на стороне клиента?
5. Самый простой способ предотвращения «атак через посредника»?
6. В чем заключаются нападения вида «межсайтовый скриптинг» (Cross-Site Scripting, XSS)?
7. Как решается проблема подделки межсайтовых запросов на стороне клиента?
8. Как решается проблема подделки межсайтовых запросов на стороне сервера?
9. Что реализует служба security?
10. Как предотвратить переход по защищенным маршрутам?

4. ВСТРАИВАНИЕ ВНЕШНИХ ЭЛЕМЕНТОВ

4.1. Основные понятия

При создании сайтов возникает необходимость встраивания различных элементов.

Так для отправки текста, содержащего абзацы, нужно добавлять текстовый редактор, иногда возникает необходимость указания схемы расположения объекта или учета статистики посещаемости ресурса.

4.2. Текстовый редактор

Добавление текстового редактора делается в три этапа:

- объявление подключения скрипта тегом <script>;
- создание текстовой области;
- вызов самого скрипта.

Поле <textarea> представляет собой элемент формы для создания области, в которую можно вводить несколько строк текста. В отличие от тега <input> в текстовом поле допустимо делать переносы строк, они сохраняются при отправке данных на сервер. Атрибут name обозначает имя поля, предназначенное для того, чтобы обработчик формы мог его идентифицировать. Атрибут id задает стилевой идентификатор – уникальное имя элемента, которое используется для изменения его стиля и обращения к нему через скрипты. Так, имя элемента TextBody используется в скрипте CKEditor. Атрибут class задает стилевой класс, который позволяет связать определенный тег со стилевым оформлением. Атрибут rows обозначает высоту поля в строках текста. Код представлены ниже:

```
<script src="js/ckeditor/ckeditor.js"></script>
<form>
  <textarea name="textBody" id="textBody" class="form-
control" rows="10" required></textarea>
  <script>
    CKEDITOR.replace('textBody');
  </script>
```

```
</form>
<button class="btn-info" onclick="alert('Кнопка нажата')">ОПУБЛИКОВАТЬ</button>
```

CKEditor – это текстовый редактор с открытым кодом для использования на веб-страницах. Он сильно упрощает добавление и редактирование информации, содержащейся на сайте, так как не нужно вручную вводить теги. Скачать CKEditor можно на сайте https://ckeditor.com/ckeditor-4/download/.

Так выглядит редактор без использования CKEditor (рисунок 14).

Рисунок 14 – Текстовый редактор без использования CKEditor

А так будет выглядеть редактор, когда используется CKEditor (рисунок 15).

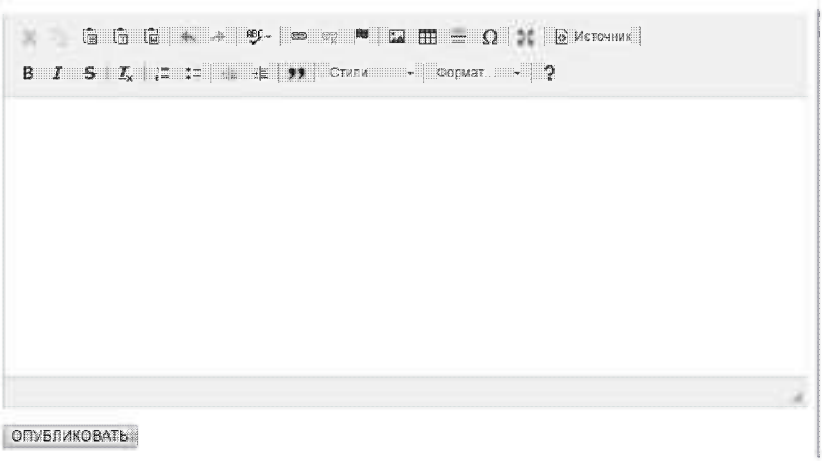


Рисунок 15 – Текстовый редактор с использованием CKEditor

4.3. Привязка к местности

Очень полезной вещью является встраивание карт на страницу сайта, так как это позволяет указать местоположение различных мест таких как офис, склад или магазин. Рассмотрим это на примере использования Яндекс карт.

Первым делом нужно зайти в конструктор карт Яндекса https://yandex.ru/map-constructor/ и выбрать местоположения необходимого объекта, например корпус на Челюскинцев (рисунок 16).

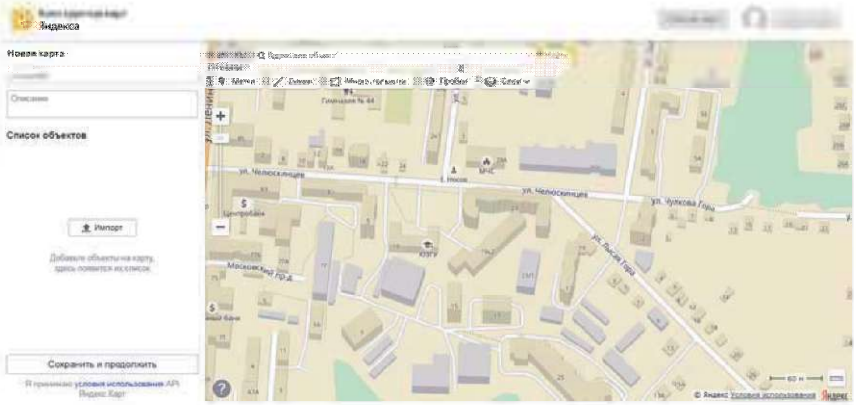


Рисунок 16 – Общий вид создания карты

Есть возможность добавления маркеров на карту, в которых будет описание (рисунок 17).

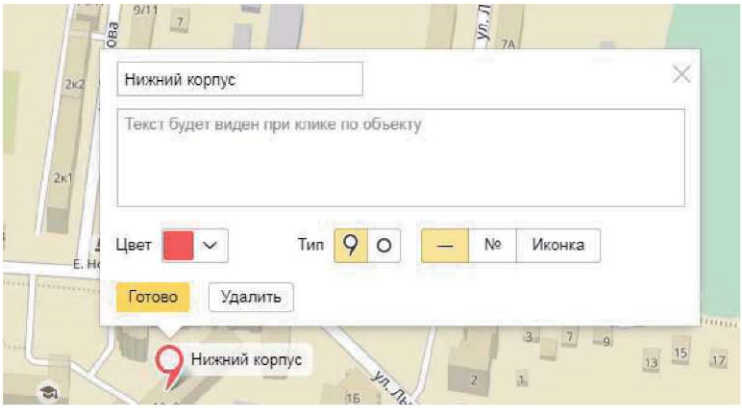


Рисунок 17 – Создание маркера

Так же можно выделять необходимые области, прокладывать маршруты, как добираться до места, выбирать слои карты (схема, спутник, гибрид).

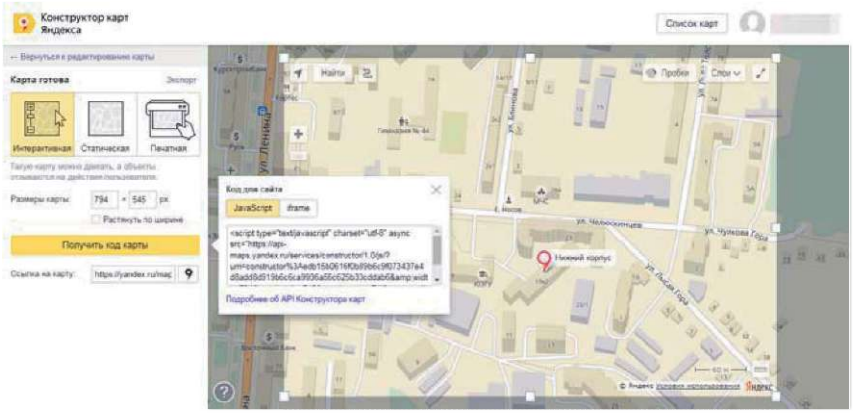


Рисунок 18 – Завершение создания карты

Получившийся результат представлен на рисунке 19.

Расположение нижнего корпуса на улице Челюскинцев

Рисунок 19 – Встроенная карта на сайт

Когда были проделаны необходимые действия и нажата кнопка «Сохранить и продолжить», остаётся выбрать тип карты (нас интересует интерактивная), размер и масштаб карты. Нажав на кнопку «Получить код карты», копируем код и вставляем его на сайт (рисунок 18).

Код представлен ниже:

```
<!doctype html>
<html>
<head>
   <meta charset="utf-8">
   <title>Добавление карты на сайт</title>
   </head>
<body>
<div>
        <h1>Расположение нижнего корпуса на улице Челюскинцев</h1>
        <script type="text/javascript" charset="utf-8" async src="https://api-
```

```
maps.yandex.ru/services/constructor/1.0/js/?um=constructor%3Aedb15
b0616f0b89b6c9f073437e4d8add8d919b6c6ca9936a56c625b33cddab6
&amp;amp;width=794&amp;amp;height=545&amp;amp;lang=ru_RU&amp;amp;scroll=tru
e"></script>
    </div>
    </body>
    </html>
```

4.4. Добавление виджетов

Часто на сайтах размещают различные полезные виджеты. Обычно алгоритм действия для добавления необходимого виджета сводится к 4 пунктам:

– поиск виджета в сети Интернет;

– настройка виджета (визуальная часть и какая информация будет отображаться);

– получение кода созданного виджета;

– добавление кода на сайт.

Результат добавления виджета, содержащего погоду представлен на рисунке 20, который расположен ниже.

Рисунок 20 – Добавленный виджет погоды

Код страницы представлен ниже:

```
<!doctype html>
<html>
<head>
   <meta charset="utf-8">
   <title>Добавление виджета погоды на сайт</title>
```

```
    </head>
    <body>
    <div style="background-color: deepskyblue; text-align: center">
        <h1>Добавление виджета погоды на сайт</h1>
      <a
href="https://clck.yandex.ru/redir/dtype=stred/pid=7/cid=1228/*https://
yandex.ru/pogoda/8" target="_blank"><img
src="https://info.weather.yandex.net/8/2.ru.png?domain=ru" border="0"
alt="Яндекс.Погода"/><img width="1" height="1"
src="https://clck.yandex.ru/click/dtype=stred/pid=7/cid=1227/*https://i
mg.yandex.ru/i/pix.gif" alt="" border="0"/></a>

    </div>
    </body>
    </html>
```

4.5. Учет статистики

Так же полезной вещью на сайте является счётчик посещаемости. Он позволяет оценивать общую посещаемость, а так же следить за детальной статистикой: какой материал на сайте наиболее интересен, какое среднее время пребывания на сайте, какие поисковые запросы привлекли пользователей и т.д.

Разберём добавление счётчика на сайт на примере бесплатного счётчика от liveinternet.ru.

Сначала нужно зайти на страницу регистрации http://www.liveinternet.ru/add и заполнить все необходимые поля (рисунок 21).

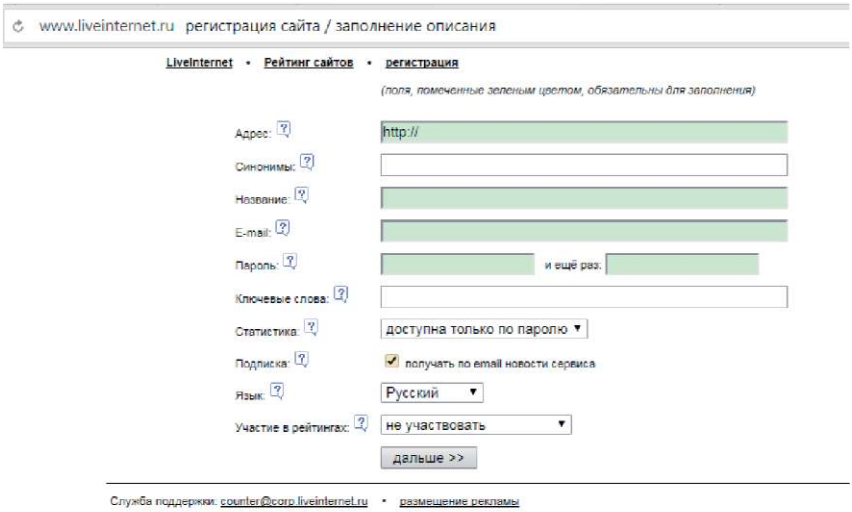


Рисунок 21 – Форма регистрации

После нажатия на кнопку «дальше», откроется страница, где нужно проверить правильность введённых данных. На следующей странице нужно нажать на кнопку «получить html-код счетчика». Откроется станица выбора счётчика (рисунок 22). Выбрав нужные параметры, можно будет скопировать код счётчика, который необходимо будет вставить на сайт.

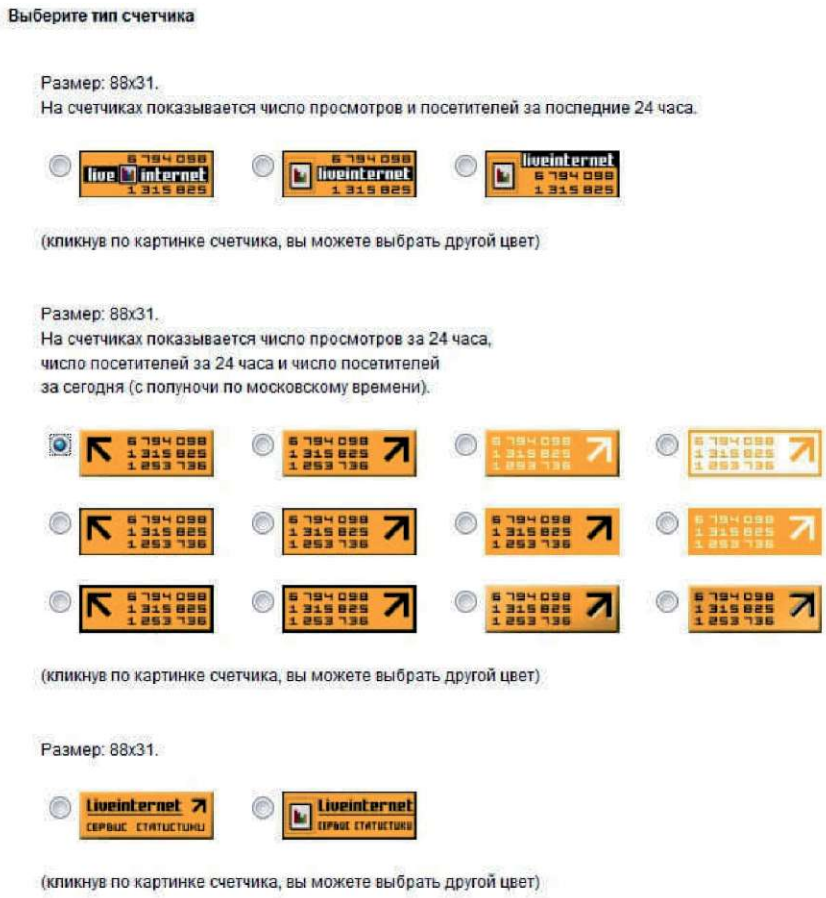


Рисунок 22 – Выбор типа счётчика

Ещё бесплатные счётчики есть у таких ресурсов:

- top100.rambler.ru;
- metrika.yandex.ru;
- top.mail.ru.

В качестве задания необходимо добавить на сайт карту с расположением объекта, счётчик посещаемости сайта и встроить текстовый редактор.

4.6. Контрольные вопросы

1. Для чего нужны встраиваемые элементы?
2. Какая польза от добавления редактора CKEditor?
3. Для чего нужны счётчики?
4. Почему лучше добавлять интерактивную карту?
5. В какую часть html-кода добавляются встраиваемые элементы?

5. ОСНОВЫ PHP-ФРЕЙМВОРКА, НАСТРОЙКА ОКРУЖЕНИЯ

5.1. Основные понятия

Laravel является мощным MVC PHP фреймворком, предназначенный для разработчиков, которые нуждаются в простом и элегантном наборе инструментов для создания полнофункциональных веб-приложений. Laravel был создан Тейлором Отуэллом. Laravel – очень особенный фреймворк с сильным брендингом, поэтому почти всё особенное в Laravel имеет уникальное название.

Composer – менеджер зависимостей для PHP. Он не предназначен конкретно для Laravel, но Laravel без него не работает.

Composer (с Packagist) содержит тысячи PHP–пакетов от сообщества, большинство из которых имеют минимум зависимостей и могут быть просто и без проблем добавлены в ваше приложение. Поэтому вам не нужно изобретать велосипед, а фреймворк не будет мешать вам использовать существующие решения.

Composer обрабатывает автозагрузку PHP-классов, используя любую конфигурацию из карты классов, файлов, PSR-0 или PSR-4.

Поскольку Composer использует версионность, вы легко можете зафиксировать версии пакетов для вашего приложения. Так вы будете уверены, что везде, где вы разворачиваете своё приложение, будет запускаться одинаковый код [8].

Начать работать с Laravel очень просто, достаточно установить Composer и выполнить:

```
composer create-project laravel/laravel
```

Laravel разработан в строгом соответствии с парадигмами ООП, и вы действительно можете использовать новейшие возможности PHP [9]. Это означает, что ваш код получит улучшенную производительность новой среды выполнения PHP.

Где вы храните секретную информацию вашего приложения, такую как пароль от БД, логин для email и другие вещи? Laravel использует хорошо известный файл .env, который также используют многие фреймворки для других языков программирования. Это файл, в котором вы определяете пары ключ-значение для любой вашей секретной информации.

Например, ваш файл .env может выглядеть так:

```
PHP
APP_ENV=production
DB_HOST=127.0.0.1
DB_DATABASE=laraveldb
DB_USERNAME=laravelapp
DB_PASSWORD=str0ngP4sw0rd
```

Вы можете создать этот файл локально и ещё один на вашем рабочем сервере и затем обновлять ваше приложение, не беспокоясь о БД и других важных секретных данных.

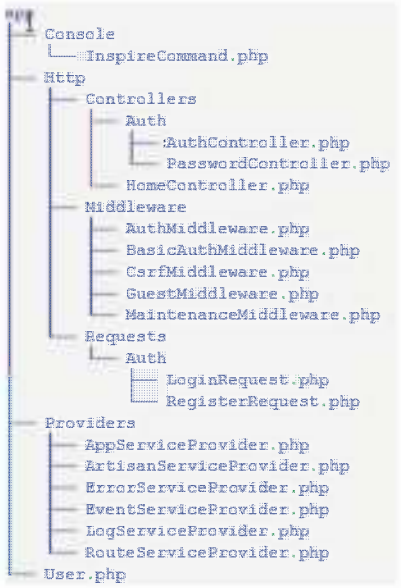


Рисунок 23 – Каталог приложения по умолчанию

Загрузка файла .env автоматически обрабатывается Laravel с помощью библиотеки Dotenv.

Laravel – первый фреймворк, поддерживающий PSR-4. Прямо из коробки Composer автоматически загрузит все классы из каталога app, используя стандарт автозагрузки PSR-4.

Это значит, что у вас может быть одно пространство имён для вашего приложения, и вы можете структурировать его, как захоти-

те и как посчитаете логичным. Laravel не требует от вас размещения определённых файлов в определённых папках.

Это каталог приложения по умолчанию с пространством имён App:

Как видите, здесь есть некоторые вещи по умолчанию, но вы можете разместить классы там, где захотите, и это делается очень просто.

Вы всегда можете изменить пространство имён приложения с помощью команды:

php artisan app:name YourNamespace

Эта команда пройдёт по всем вашим файлам и изменит в них объявление пространства имён.

В Laravel есть эта восхитительная возможность, которая называется запросами форм. Это проверка запросов для ваших контроллеров. Это не просто проверка данных, а полная обработка запроса.

Запросы форм комбинируются с функциональностью внедрения методов, чтобы добавить бесшаблонный способ проверки пользовательского ввода. Например, класс RegisterRequest:

```
<?php namespace App\Http\Requests;
class RegisterRequest extends FormRequest {
 public function rules()
 {
  return [
   'email' => 'required|email|unique:users',
   'password' => 'required|confirmed|min:8',
  ];
 }
 public function authorize()
 {
  return true;
 }
}
```

У этого запроса есть правила для проверки данных и функциональность авторизации, которая определяет, кто может использовать этот запрос. В этом примере это может быть кто угодно, поэтому любой может зарегистрироваться, используя поля **email** и **password**.

Чтобы использовать этот запрос, вам надо только намекнуть на объект запроса в вашем контроллере, и вы получите проверку запроса очень простым путём:

```
<?php namespace App\Http\Controllers\Auth;
use Illuminate\Routing\Controller;
use Illuminate\Contracts\Auth\Guard;
use App\Http\Requests\Auth\LoginRequest;
use App\Http\Requests\Auth\RegisterRequest;
class AuthController extends Controller {
 protected $auth;
 public function __construct(Guard $auth)
 {
  $this->auth = $auth;
 }
 public function register(RegisterRequest $request)
 {
 // Форма регистрации прошла проверку, создать пользователя...

  $this->auth->login($user);
  return redirect('/');
 }
}
```

Вы видите, что в этом контроллере нет какого-либо связующего кода или кода проверки, всё сделано раздельно и чисто.

У запросов форм есть функциональность перенаправления для случаев ошибок проверки ввода, сообщений для представлений и т.д.

1. «Что такого прекрасного в Laravel?», Laravel по-русски, автор: Daniel V от 14.01.15 (https://laravel.ru/posts/177).

5.2. Индивидуальные задания

В рамках данного учебного пособия на примере создания портала по обмену информацией, поэтапно будет рассказано и показано, как от установки и настройки всех нужных компонентов разработать полноценный web-сервис, перенести его на удаленный сервер и протестировать.

Таблица 7

№	Задание
0	Разработка защищенной автоматизированной системы по обмену информацией
1	Разработка защищенной автоматизированной системы для работы с графическим контентом
2	Разработка защищенной автоматизированной системы для работы с аудио-контентом
3	Разработка защищенной автоматизированной системы для проведения тендеров
4	Разработка защищенной автоматизированной системы для работы с видео-контентом
5	Разработка защищенной автоматизированной системы для записи на прием к специалисту в больнице
6	Разработка защищенной автоматизированной системы для учета коммунальных показаний
7	Разработка защищенной автоматизированной системы для составления сметы на строительные работы с возможностью экспорта данных
8	Разработка защищенной автоматизированной системы для отчетной деятельности высшего учебного заведения.
9	Разработка защищенной автоматизированной системы для оказания консультационных услуг
10	Разработка защищенной автоматизированной системы для составления расписания общественного транспорта

5.3. Установка сервера

В связи с тем, что в рамках работы предстоит использовать базу данных, а для описания логики работы системы потребуется язык PHP, необходимо наличие сервера, который возьмет на себя функции интерпретатора и хранилища. На начальном этапе разработки использовать выделенный сервер не целесообразно, оптимальным решением является применение виртуального сервера. Одним из популярных вариантов является сборка XAMPP, включающая в себя Apache, MySQL, PHP, phpMyAdmin, и др. Дистрибутив доступен для скачивания на сайте разработчика:

https://www.apachefriends.org/ru/download.html

После установки убеждаемся в возможности запуска требуемых компонентов. В случае, если индикатор (рисунок 24) не загорается зеленым цветом, то это означает, что требуемые порты используются сторонними приложениями (виртуальные машины, Skype, сетевые утилиты) и требуется индивидуальное разрешение конфликтов.

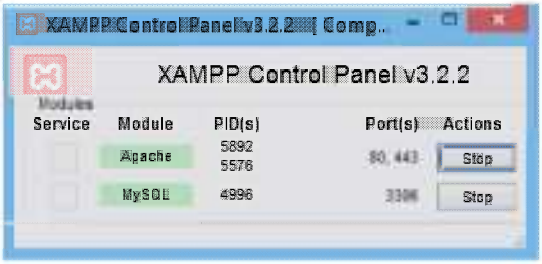


Рисунок 24 – Панель управления XAMPP

Далее необходимо добавить путь к файлу php.exe в системную переменную PATH (в противном случае php-команды не будут выполнены в среде разработки). Расширенное значение переменной показано на рисунке 25.

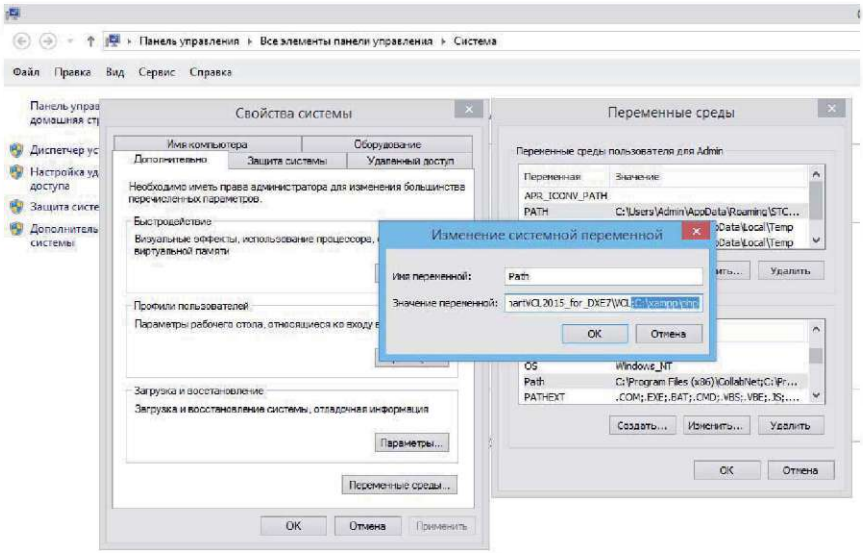


Рисунок 25 – Расширенное значение переменной PATH

Для разрабатываемых проектов предусмотрен выделенный каталог «htdocs». Для наглядности будем указывать короткие пути относительно каталога установки XAMPP (полный путь в данном случае выглядел бы «C:\xampp\htdocs»). Для работы с проектом создаем пустой каталог, например «new-system», рисунок 26.

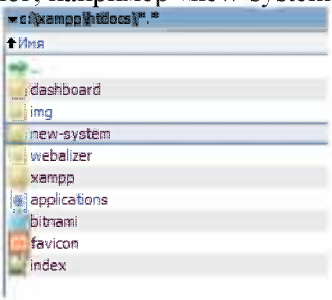


Рисунок 26 – Новый каталог для разрабатываемой системы

Важно помнить, что случае смены структуры каталогов обязательно требуется перезапуск Apache т.к. в противном случае изменения не вступят в силу.

5.4. Настройка клиента для работы с БД

Для управления базами данных MySQL существует ряд клиентов, такие как phpMyAdmin (встроен в XAMPP), SQLWorkBench, HeidiSQL и другие. Мы не будем пользоваться встроенным phpMyAdmin в связи с его медлительностью, а воспользуемся бесплатным решением HeidiSQL. Это легкий и быстрый клиент, обеспечивающий необходимый функционал. Этот продукт не требует установки и для работы достаточно скачать исполняемый файл с сайта разработчика:

https://www.heidisql.com/download.php

После запуска необходимо добавить подключение к серверу. Добавляем локальный хост 127.0.0.1, логин root, пароль пустой (рисунок 27).

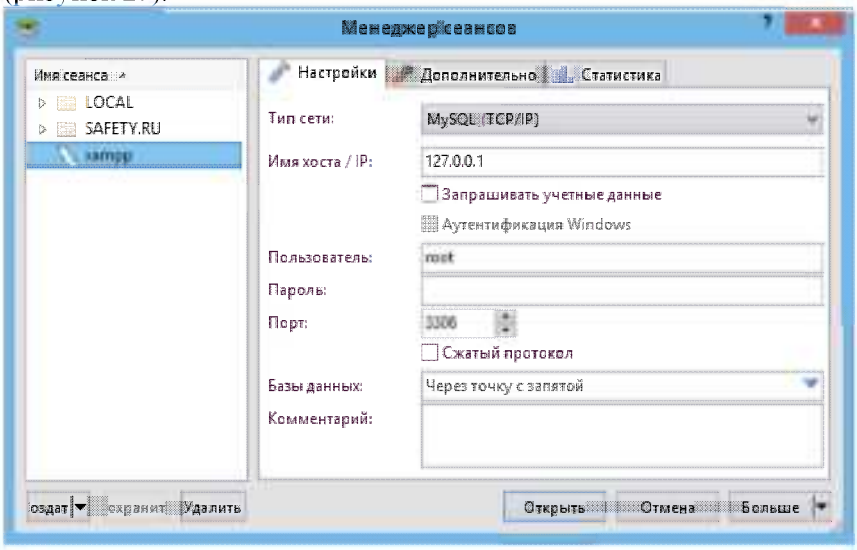


Рисунок 27 – Настройка подключения к базе данных

Создаем новую базу данных (рисунок 28), для корректного отображения символов кириллицы указываем кодировку utf8_general_ci (рисунок 29).

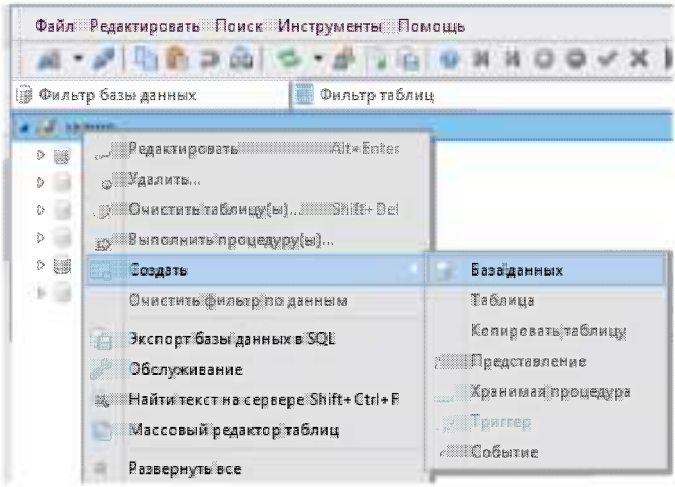


Рисунок 28 – Создание новой базы данных

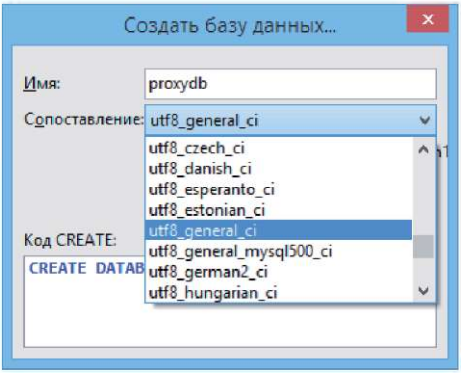


Рисунок 29 – Выбор кодировки символов

На этом создание пустой базы данных успешно завершено. Создание объектов базы вручную не требуется, т.к. это будет производиться на уровне миграций в среде разработки. Данный механизм позволяет обеспечить прозрачность изменений в базе данных, дает возможность выборочного отката изменений.

5.5. Установка среды разработки

Для работы с проектом мы будет пользоваться кросс-платформенной интегрированной средой разработки PhpStorm или Visual Studio Code, представляющих собой интеллектуальный редактор для PHP, HTML и JavaScript с возможностями анализа кода на лету, предотвращения ошибок в коде и автоматизированными средствами рефакторинга для PHP и JavaScript. Установочный дистрибутив доступен на сайте разработчика:

https://www.jetbrains.com/phpstorm/download/#section=windows

Главное окно программы после запуска показано на рисунке 30.

Рисунок 30 – Окно программы после первого запуска PhpStorm

5.6. Создание нового проекта

Известно, что разработка сложных распределенных систем ведется группой разработчиков, каждый из которых выполняет свою задачу. Для того, чтобы результаты были согласованы необходимо придерживаться определенных правил. Целесообразно разделять данные приложения, пользовательского интерфейса и управляющей логики на отдельные компоненты – модель, пред-

ставление (вид) и контроллер. При таком подходе (Model–View–Controller, MVC) модификация каждого компонента может осуществляться независимо. Одним из наиболее популярных бесплатных веб-фреймворков с открытым кодом, предназначенный для разработки с использованием архитектурной модели MVC является Laravel. Ключевые особенности, лежащие в основе архитектуры:

1. Eloquent ORM, предполагает реализацию шаблона проектирования ActiveRecord на PHP. Позволяет строго определить отношения между объектами базы данных. Стандартный для Laravel построитель запросов Fluent поддерживается ядром Eloquent.
2. Миграции – система управления версиями для баз данных. Позволяет связывать изменения в коде приложения с изменениями, которые требуется внести в структуру БД, что упрощает развёртывание и обновление приложения.
3. Логика приложения – часть разрабатываемого приложения, объявленная либо при помощи контроллеров, либо маршрутов (функций-замыканий).
4. Обратная маршрутизация связывает между собой генерируемые приложением ссылки и маршруты, позволяя изменять последние с автоматическим обновлением связанных ссылок. При создании ссылок с помощью именованных маршрутов Laravel автоматически генерирует конечные URL.
5. REST-контроллеры – дополнительный слой для разделения логики обработки GET- и POST-запросов HTTP.
6. Составители представлений – блоки кода, которые выполняются при генерации представления (шаблона).

В установленной среде PhpStormсоздаем новый проект проект на базе менеджера зависимостей Composer. Это позволит нам использовать нужные программные модули напрямую с выделенных ресурсов разработчиков и актуализировать версии с учетом возможных обновлений (в рамках отдельного составляющего проекта). Выбираем каталог «new-system» для размещения файлов проекта. Версия по умолчанию предполагает использование самой свежей стабильной версии. С использованием кнопки выбора, указываем интерпретатор PHP и нажимаем кнопку создания проекта (рисунок 31).

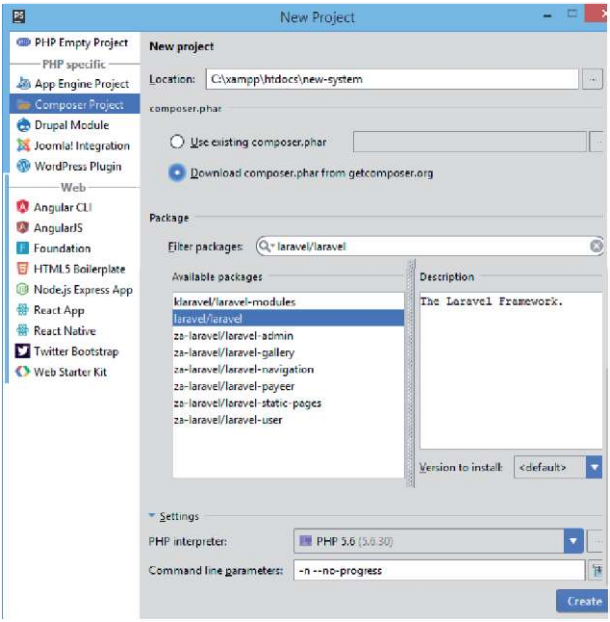


Рисунок 31 – Создание нового проекта на базе Composer+Laravel

После того, как Composer скачает необходимое окружение, мы получим структуру каталогов, показанную на рисунке 32.[2]

В корне свежеустановленного фреймворка вы можете видеть следующие каталоги:

app – здесь располагается собственно приложение. Ниже мы рассмотрим содержимое этого каталога подробнее.

bootstrap – содержит файлы, которые осуществляют первоначальную загрузку (bootstraping) фреймворка и настраивают автозагрузку классов.

config – здесь находятся конфигурационные файлы приложения.

database – каталог для файлов миграций БД и "посева" данных.

public – является DocumentRoot домена вашего приложения и содержит статические файлы - css, js, изображения и т.п.

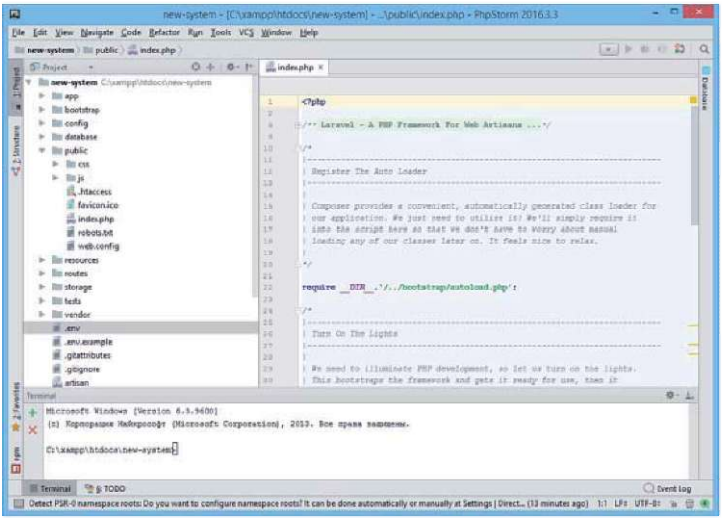


Рисунок 32 – Окружение laravel

resources – здесь находятся шаблоны (Views), файлы локализации и, если таковые имеются, рабочие файлы LESS, SASS и js-приложения на фреймворках типа AngularJS или Ember, которые потом собираются внешним инструментом в папку public.

storage – этот каталог должен иметь права для записи в него извне и в нём Laravel хранит скомпилированные Blade-шаблоны, файлы сессии, файловый кэш и другие сгенерированные файлы, нужные для работы.

test – каталог для юнит-тестов

vendor – в этот каталог Composer устанавливает пакеты, указанные в composer.json.

Внутри App находятся несколько дополнительных каталогов, таких как Console, Http и Providers. Первые два каталога, как следует из названия, содержат классы, предоставляющие API к вашему приложению по протоколам CLI (командная строка) и HTTP (работа через браузер). В Console находятся классы Artisan-команд, а в Http – контроллеры, фильтры и реквесты, т.е. классы валидации пользовательского ввода. Такой подход должен подтолкнуть новичков к отходу от общепринятого, но вредного подхода писать

весь код в контроллерах, и абстрагировать логику приложения от метода обращения к нему.

Каталог Exceptions содержит обработчики исключений, а так же здесь следует размещать сами классы исключений, которые используются в вашем приложении.

Переходим к настройке отображения главной страницы системы в браузере.

Для этого необходимо прописать адрес ресурса в файлы:

C:\xampp\apache\conf\extra\httpd-vhosts.conf
<VirtualHost *:80>
ServerAdmin webmaster@dummy-host.example.com
DocumentRoot "C:/xampp/htdocs/new-system/public"
ServerName new-system.local
ServerAlias www.new-system.local
ErrorLog "new-system.local-error.log"
CustomLog "new-system.local-access.log" common
</VirtualHost>
C:\Windows\System32\drivers\etc\hosts
127.0.0.1 new-system.local

и произвести перезапуск Apache в панели управления XAMPP. После этого, при указании в браузере адреса http://new-system.local/ будет показано главное окно проекта (рисунок 33 страница по умолчанию).

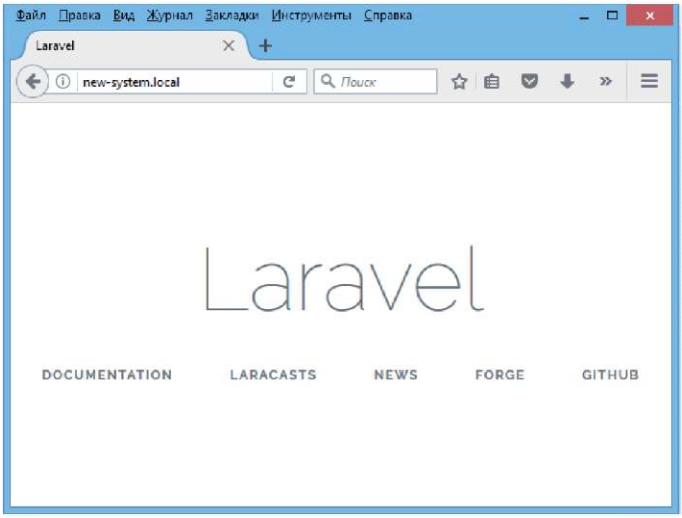


Рисунок 33 – Главное окно чистого проекта

Для того, чтобы работать с базой данных, необходимо выполнить ее подключение в файле .env

```
DB_HOST=127.0.0.1
DB_DATABASE=proxydb
DB_USERNAME=root
DB_PASSWORD=
```

На этом этап настройки окружения закончен.

5.7. Основные команды

Основные команды, которые потребуется нам для работы с фреймворком:

- php artisan auth:clear-resets : сброс одноразовых кодов подтверждения для запросов восстановления паролей
- php artisan db:seed : запуск классов генерации тестовых записей
- php artisan key:generate : генерация ключа приложения
- php artisan make:auth: подключение модуля авторизации и регистрации из "коробки»
- php artisan make:controller : создание контроллера
- php artisan make:migration: создание файла миграции
- php artisan make:model: создание файла модели
- php artisan make:seeder: создание файла для тестовых записей данных в таблицу
- php artisan migrate:refresh: перезапуск всех миграций
- php artisan migrate:reset : откат всех миграций
- php artisan migrate:rollback : откат файла последней миграции
- php artisan migrate:status : демонстрация статуса каждой миграции

5.8. Контрольные вопросы

1. Кем был создан Laravel?
2. В чем особенности фреймворка Laravel?
3. Что такое Composer? Функции Composer.
4. Где хранится секретная информация приложения?
5. Что такое запросы форм?
6. Что из себя представляет XAMPP?
7. Какие используют клинты для управления базами данных MySQL?
8. Ключевые особенности, лежащие в основе архитектуры Laravel?
9. При помощи какой команды можно создать файл миграции?

6. РАЗРАБОТКА БАЗЫ ДАННЫХ

6.1. Основные понятия

Авторизация – это процесс проверка прав пользователя на выполнение определенного действия. То есть, до начала процесса авторизации, пользователь уже должен быть аутентифицирован (залогинен) чтобы в коде можно было идентифицировать данного пользователя для проверки его прав. Хотя есть возможность проверить права и для любого другого пользователя.

Laravel 5 предлагает разработчикам достаточно удобные инструменты по созданию механизма авторизации и регистрации пользователей практически без необходимости внесения в него каких-либо изменений ручным способом.

Для начала, в проекте должна быть настроена работа с базой данных. В данном проекте будет использоваться MySQL, поэтому в файле config/database.php заполняется блок, имеющий отношение к данной СУБД.

После настройки взаимодействия с базой данных, добавление всех необходимых средств авторизации происходит с помощью нескольких консольных команд:

php artisan make:auth

php artisan migrate

После исполнения данных команд, по адресу /register данного проекта появляется страница регистрации, при помощи которой можно произвести первую регистрацию пользователя.

По умолчанию, таблица для хранения данных пользователей в Laravel состоит из семи полей и включает в себя идентификатор, имя пользователя, адрес электронной почты пользователя, пароль в зашифрованном с помощью bcrypt виде, код восстановления пароля (задается только при заполнении формы запроса на восстановление пароля), дат создания учетной записи и последней правки ее данных.

Механизм авторизации по умолчанию в Laravel со стороны безопасности

Форма регистрации пользователей в Laravel в состоянии «из коробки» имеет ряд потенциальных уязвимостей, среди которых:

1. Валидатор пароля в файле \app\Http\ Controllers\ Auth\ RegisterController.php разрешает принимать к регистрации любые пароли длиной от 6 знаков без дополнительных к ним требований, таких, как использование символов разного регистра, цифр и специальных знаков. Как правило, это означает, что пользователями часто и будут использоваться пароли длиной в шесть знаков, состоящие из легко угадываемых последовательностей символов вроде «123456», которые весьма легки к подбору при помощи автоматизированных средств. В то же время, установка вышеупомянутых требований к паролю позволит существенно уменьшить вероятность его автоматизированного подбора. Так, к примеру, по результатам анализа скорости перебора паролей к RAR архивам была показана скорость подбора от 8 до 37 паролей в секунду. При такой скорости подбор пароля длиной 6 символов, в котором использовались латинские буквы одного регистра и цифры (количество возможных вариантов 2176782336) займет около двух лет[1].

2. Пользователь регистрируется на сайте незамедлительно после валидации введенных в форму регистрации данных, после чего автоматически авторизуется. Данная мера способна позволить беспрепятственную регистрацию роботизированных аккаунтов для рассылки спама либо прочих вредоносных действий, проверенным временем способом защиты от которой является проверка адреса электронной почты регистрируемого пользователя путем отправки на него проверочного письма с одноразовым кодом. Перейдя по ссылке из письма, пользователь подтверждает свои контактные данные, при этом аккаунты простейших роботов, не поддерживающих проверку по электронной почте, остаются в базе данных проекта неактивированными и их вредоносная активность таким образом успешно пресекается.

3. Форма регистрации пользователей не защищена даже простейшими средствами защиты от автоматизированных регистраций. Разумеется, типовые коды captcha успешно разгадываются многими роботами, тем не менее, на сегодняшний день существует достаточно эффективный и удобный для легитимных пользователей способ защиты от автоматических регистраций – reCaptcha от

[1] «Утилиты для подбора паролей к архивам и документам», журнал «КомпьютерПресс» №3'2007 (http://www.compress.ru/article.aspx?id=17375)

Google. Также, с начала 2017-ого года, корпорацией Google была выпущена версия reCaptcha, которая для пользователей, в отношение которых отсутствуют подозрения в использовании автоматизированных механизмов взаимодействия с веб-сайтами, не отображается вовсе[2]. При всем этом, подключение и использование в коде проекта данной версии, reCaptcha Invisible, ничем не сложнее прежней, более простой версии, в чем можно убедиться, ознакомившись с документацией для разработчиков по адресу https://developers.google.com/recaptcha/docs/invisible.

В отличие от вышеупомянутых уязвимостей, атака типа CSRF[3] предусмотрена разработчиками изначально, поэтому во все формы, создаваемые средствами Laravel, добавляется поле для передачи одноразового токена, который используется для установления источника данных формы[4]. Использование одноразового токена предотвращает возможность использования формы на другом сайте, данные из которой передавались бы на данный проект, а также препятствует использованию скриптов регистрации вредоносных ботов, не предусматривающих получения формы с вебстраницы, а отправляющих данные для регистрации посредством POST-запроса непосредственно во время первого обращения к сайту.

6.2. Создание миграций

Миграции – что-то вроде системы контроля версий для вашей базы данных. Они позволяют вашей команде изменять структуру БД, в то же время оставаясь в курсе изменений других участников. Миграции обычно идут рука об руку с построителем структур для более простого обращения с архитектурой вашей базы данных.

Файл миграции создается для случаев, когда таблицу в базе данных проекта необходимо создать либо отредактировать. По су-

---

2 «Google ReCAPTCHA Invisible или долой дорожные знаки и витрины магазинов», Хабрахабр, автор: steff от 30.01.2017 (https://habrahabr.ru/post/320664/)

3 «Межсайтовая подделка запроса», Википедия (https://ru.wikipedia.org/wiki/Межсайтовая_подделка_запроса)

4 «Методы защиты от CSRF-атаки», Хабрахабр, автор: Flaker от 29.12.2016 (https://habrahabr.ru/post/318748/)

ти дела, система миграций в Laravel – это что-то вроде системы управления версиями, только для структуры базы данных.

Создание нового файла миграции происходит наиболее удобно из консоли следующей командой:

php artisan make:migration create_test_table

Результатом исполнения данной команды является создание пустого файла миграции в папке database\migrations.

Файл миграции имеет имя вида 2017_10_10_164010_create_tests_table.php, при помощи которого можно определить дату, от которой миграция была произведена, а также установить порядок миграций в рамках одного календарного дня. Порядок файлов в папке миграций играет важную роль, поскольку при некоторых типах правок крайне важным является именно применение миграций в определенном порядке. Наиболее ярким примером данного явления может быть случай, когда поле в одном файле миграции является внешним ключом поля из другого файла. В данном случае, применение вначале первого файла завершится ошибкой, поскольку поля, привязка к которому должна быть осуществлена, еще не существует в базе данных [10].

После создания нового файла миграции внутри можно найти две секции кода: в рамках фунции up() и функции down(). Нетрудно догадаться, что up() описывает действия, производимые со структурой базы данных при применении миграции, а down(), соответственно, при ее отмене.

Примерная схема миграции при создании новой таблицы имеет следующий вид:

```
Schema::create('posts', function(Blueprint $table) {
    $table->increments('id');
    $table->timestamps();
    $table->string('title', 255);
    $table->text('content');
    $table->integer('user_id')->unsigned();
});
```

В данной миграции устанавливается структура таблицы, согласно которой вхождения будут иметь:

1. Порядковый номер, устанавливаемый автоматически со стороны СУБД;

2. Данные о дате создания и последней правке;

3. Поле «title» длиной до 255 знаков, содержащее заголовок публикации;

4. Текстовое поле, используемое для хранения основного содержания публикации;

5. Поле «user_id», содержащее порядковый номер пользователя, опубликовавшего статью.

Кроме показанных в данном примере типов полей, система миграций в Laravel поддерживает множество других представленных в таблице 8.

Таблица 8

binary	Данные в двоичном формате (BLOB)	Может содержать файл или иные значения
boolean	Логическое значение	true/false
char	Строчное значение типа CHAR	Строка
date	Поле в формате DATE, содержит только дату	1970-01-01
dateTime	Поле в формате DATETIME, содержит дату и время	1970-01-01 00:00:01
double	Не целое число в формате DOUBLE максимальной точностью в 15 цифр. Не рекомендуется для хранения денежных значений	1.7976931348
float	Не целое число с плавающей точкой типа FLOAT точностью в 7 цифр	3.14

Продолжение Таблицы 8

increments	Порядковый номер вхождения, назначаемый автоматически в формате INT	5
integer	Целочисленное значение	От 0 до 4294967295 при UNSIGNED и от -2147483648 до 2147483647 в ином случае
ipAddress	IP-адрес	
json	JSON-массив	{“data”: “value”, “data1”: “value1”}
longText	Увеличенное в объеме текстовое поле типа LONGTEXT	Максимальный объем: 4,294,967,295 (232^−1) байт = 4 Гб
macAddress	MAC-адрес	00:0a:95:9d:68:16
rememberToken	Поле для токена, используется в Laravel и не является эквивалентом типа данных СУБД	VARCHAR(100) NULL
string	Строчное значение	VARCHAR не более 255 символов
text	Текстовое значение	Максимальный объем: 64 Кб
time	Время, эквивалент TIME	12:00:00
timestamp	Метка времени в формате UNIX timestamp	1507786306
timestamps	Создает поля created_at и updated_at, поддерживающие нулевые значения. Используется в Laravel.	

Кроме этих параметров, в файле миграции Laravel имеется возможность задать также и свойства полей. Некоторые из них представлены в таблице 9.

Таблица 9

nullable	Позволяет полю принимать нулевое значение
default($value)	Устанавливает значение поля по умолчанию
unsigned	Для полей типа INTEGER устанавливает режим, в котором значение является всегда положительным

Также, для полей в файле миграции можно задать индексы.

Таблица 10

primary('id');	Задать первичный ключ для поля `id`
primary(['first', 'last']);	Задать первичный композитный ключ для полей `first` и `last`
unique('email');	Сделать значения поля `email` уникальным во всей таблице
index('state');	Задать базовый индекс для поля `state`

Кроме всего прочего, для большей гибкости структуры таблиц в базе данных, файлы миграций Laravel поддерживают также и внешние ключи, которые следует задавать уже на основании прежде созданных полей. К примеру, взглянем на следующий фрагмент кода:

```
$table->integer('user_id')->unsigned();
$table->foreign('user_id')->references('id')->on('users');
```

Первая строка данного кода создает поле типа INTEGER под названием `user_id`, являющееся строго положительным числом.

Вторая строка указывает на то, что поле `user_id` является внешним ключом, имеющим отсылку к полю `id` таблицы `users`.

Обратите внимание, что при использовании подобной конструкции, на момент применения файла миграции таблица `users` с полем `id` уже должна существовать в базе данных.

Отметьте, что объявление внешних ключей происходит отдельной секцией кода в файле миграции, а не в основной:

```
Schema::table('posts', function($table) {
$table->foreign('user_id')->references('id')->on('users');
});
```

6.3. Пример выполнения задания

В рамках примера мы будем создавать портал по обмену информацией. Так как там потребуется авторизация и регистрация пользователей, то сразу после создания чистого проекта Laravel выполним команду **php artisan make:auth**. После выполнения данной команды мы должны получить сообщение об успешном создании модуля авторизации (рисунок 34).

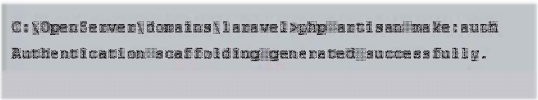


Рисунок 34 – Результат успешного создания авторизации

В состав данного модуля входит два файла миграции, расположенных в директории database\migrations, основным из которых является файл миграции для создания таблицы пользователей; файл представления домашней страницы, куда пользователи будут попадать после обработки дополнительных форм авторизации и регистрации, соответствующие представлениям контроллеры (app\Http\Controllers\Auth) и модель User.

В связи с особенностями технического задания, наша таблица, содержащая данные пользователей будет несколько отличаться от варианта «из коробки». Учитывая это, выполнять миграцию еще рано, канонично выполнять минимум действий с процессом миграции, чтобы не заниматься каждый раз их модификацией.

Далее приступим к модификации файла миграции таблицы `users`. Вот, что нам предлагает «коробочная» версия: идентификатор пользователя, уникальное имя (логин) пользователя, уникаль-

ный адрес электронной почты, пароль (рис. 2). В рамках нашего ТЗ нам необходимо расширить таблицу `users` следующими полями:

1. «status» – статус пользователя в системе (допустимые значения: «student» для обычных привилегий студента, «monitor» для старосты группы, «teacher» для преподавателей),

2. «gruppa» – данные о привязке пользователя к месту в структуре университета (для студентов и старост – группа, для преподавателей – кафедра),

3. «teacher_groups» – список идентификаторов групп, курируемых преподавателем (для преподавателей, у остальных категорий пользователей не используется),

4. «reg_ip» – IP-адрес пользователя на момент прохождения процедуры регистрации,

5. «verified» – для преподавателей и старост устанавливает статус прохождения верификации учетной записи пользователя после регистрации,

6. «is_admin» – значение, обозначающее является ли пользователь администратором или нет и имеет ли он право на верификацию новых регистраций преподавателей и старост.

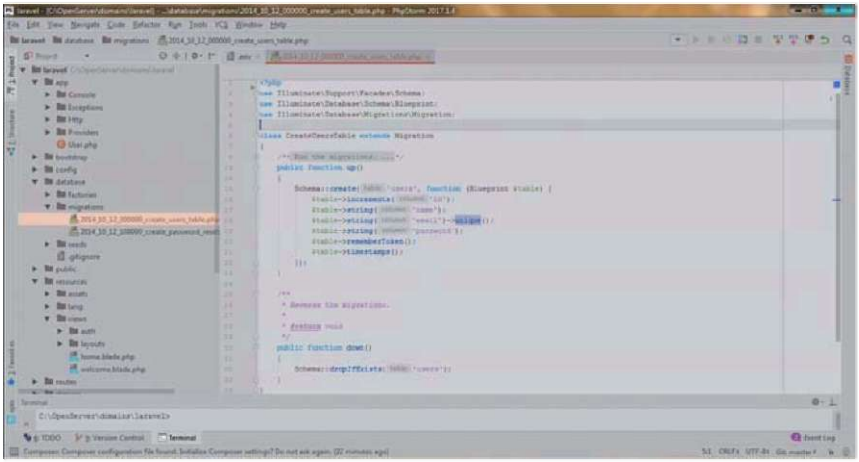


Рисунок 35 – Файл миграции таблицы `users` «из коробки»

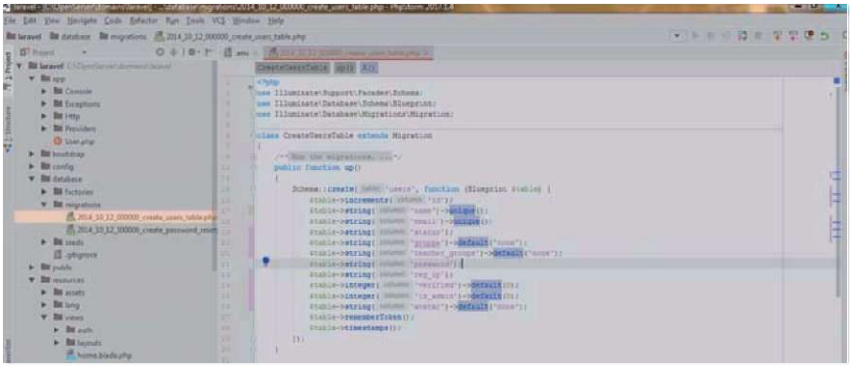


Рисунок 36 – Файл миграции таблицы `users` после модификации в соответствии с требованиями ТЗ

После выполнения всех необходимых манипуляций с файлами миграции, можно, наконец, выполнить команду **php artisan migrate** для записи необходимых полей и таблиц в нашу локальную базу данных MySQL. После сообщения об успешном создании миграций можно зайти в веб-клиент phpMyAdmin или в любой другой и убедиться, что все таблицы были созданы корректно.

Распространенные ошибки при работе с миграциями

1. Syntax error or access violation: 1071 Specified key was too long

Причины возникновения:

Связана с различиями при использовании сравнения `utf8mb4_unicode_ci` (`utf8mb4_general_ci`) вместо `utf8_unicode_ci` (`utf8_general_ci`). Поскольку в первом случае каждый символ кодируется большим количеством байт, то предельная длина строки уменьшается с доступных для последней 255 знаков до 191. Таким образом, при создании строчного поля длиной в 255 знаков с использованием сравнения utf8mb4 превышается лимит на объем поля в байтах [11].

Решение:

Для устранения данной проблемы необходимо ограничить предельную длину строки по умолчанию в символах либо для одного конкретного файла миграции, который вызывает ошибку, либо для всего проекта глобально. Для этого необходимо править

файл app\Providers\AppServiceprovider.php в части метода boot и внести в него нижеследующую директиву:

Schema::defaultStringLength(191);

Отметьте, что по умолчанию файл провайдера AppServiceprovider не использует класс Schema, поэтому для успешной работы данного исправления необходимо также добавить в начало файла use-обращение к данному классу:

use Illuminate\Support\Facades\Schema;

2. General error: 1215 Cannot add foreign key constraint

Причины возникновения:

I. Использование в файле миграции внешних ключей, ссылающихся на поля либо таблицы, которые еще не существуют,

II. Объявление внешних ключей непосредственно в основном блоке create файла миграции.

Решение:

В первом случае необходимо настроить порядок миграций таким образом, чтобы первой была задана таблица, на которую ссылается внешний ключ из другой таблицы. Поскольку файлы миграций исполняются в порядке, установленном в имени файла, достаточно будет изменить порядковый номер, следующий за датой, таким образом, чтобы определить приоритет при исполнении.

Во втором случае необходимо вынести все директивы, касающиеся установки внешних ключей, в отдельный блок файла миграции, следующий за созданием таблицы либо иными операциями над структурой.

6.4. Контрольные вопросы

1. Что такое авторизация?
2. Как производится хранение пароля при использовании "коробочной" авторизации?
3. Что такое миграция?
4. Каким образом производится накат (откат) миграций?
5. При помощи каких команд происходит добавление всех необходимых средств авторизации?
6. Какой командой происходит создание нового файла миграции ?
7. Какие действия описывает фунции up() и функции down()?

7. БЕЗОПАСНОСТЬ БАЗ ДАННЫХ

7.1. Основные понятия

Сложно представить современное многофункциональное веб-приложение без базы данных, которая позволяет хранить и обрабатывать информацию. Не исключением являются приложения, основанные на фреймворке Laravel. С одной стороны, база данных – это очень мощный инструмент для проектировки приложения, но с другой стороны и одна из главных целей атаки злоумышленника. Поэтому при разработке динамических запросов SQL следует обратить внимание на такую атаку как SQL-injection.

Уязвимости типа SQL-injection [sql ɪn'ʤekʃn] ("инъекция в SQL-запросы") возникает, когда разработчик дает возможность каким-либо способом влиять на запросы SQL, которые приложение динамически передаёт на сервер СУБД. Имея такую возможность, злоумышленник может внедрять свои собственные операторы, используя функциональность базы данных, с помощью специализированного языка структурированных запросов SQL. В результате чего, если запрос является синтаксически верным, то сервер выполнит его.

SQL-injection – уязвимость не только веб-приложений, любой код, который принимает входящие данные от ненадежного источника, а затем использует их для формирования динамических запросов SQL может быть подвержен данной атаке. Поэтому следует внимательно относиться к информации, получаемой от пользователей.

SQL-injection квалифицируют:

- по типу переменной (целочисленная, строковая);
- по типу SQL-запроса и месту SQL-injection в запросе;
- по типу SQL-injection (Union-based, Error-based, Blind-based, double-blind-based).

Процесс проведения атаки SQL-injection можно разделить на два этапа: поиск SQL-injection и эксплуатация. Для проведения атаки необходимо по завершении поиска SQL-injection обладать информацией о точках входа приложения, типах входящих параметров, типе СУБД, с которой взаимодействует приложение.

Поиск: при поиске SQL-injection в первую очередь следует определить множество точек входа приложения, которые взаимодействуют с базой данных. Такими точками входа могут быть поля для поиска и/или ввода данных, GET-параметры и POST-параметры запросов (исследование таких параметров производится при помощи проксирующих серверов – Burp Suite, OWASP ZAP), параметры, содержащиеся в URL и т.д.

Не всё множество точек входа приложения является инъектируемым посредством SQL-injection, например точка входа может быть не связана с SQL-базой данных. Для определения подмножества точек входа подходящих для эксплуатации уязвимости необходимо протестировать параметры SQL-запроса, сформированного с использованием входных данных, полученных из соответствующих точек входа.

Тестирование параметров заключается в определении его типа данных. Существует множество типов данных, используемых в приложениях, работающих с SQL-базами данных: строковые, с плавающей точкой (дробные числа), целые числа, дата и время. Рассмотрим тестирование параметров, содержащих строковый и числовой типы данных [12].

Строковый входящий параметр: предположим, что сформированный приложением SQL-запрос (оригинальный запрос) к БД выглядит следующим образом:

SELECT * FROM news WHERE autor='pupkin'

Теперь вставим символ «pupkin'» в качестве входящих данных для изменения параметра «autor», таким образом, модифицированный запрос будет выглядеть так:

SELECT * FROM news WHERE autor='**pupkin''**

Здесь наблюдаем нарушение синтаксиса и логики SQL-запроса, что приводит к невозможности СУБД правильно обработать запрос. В случае если включены сообщения об ошибках, приложение вернет следующее (варианты могут различаться для различных СУБД и их версий): «*mysql_query(): You have an error in your SQL syntax check the manual that corresponds to your MySQL server version for the right syntax to use near '1'*». Если же сообще-

ния об ошибках выключены, то для нахождения уязвимости следует протестировать параметр с помощью логических конструкций:

Таблица 1. Логические конструкции

Логические конструкции
autor='pupkin' -- '
autor='pupkin' or 1=1 -- '
autor="pupkin" and 1=1 -- "
autor="pupkin" and 1=2 -- "

Исходя из таблицы 1 модифицированный запрос будет выглядеть так:

SELECT * FROM news WHERE autor='**pupkin' or 1=1** -- '

Таким образом, для «MYSQL» запрос останется правомерным, так как с помощью символов « -- » часть SQL-запроса будет закомментирована, что позволит обработать запрос без синтаксических ошибок и вывести информацию такую же, что и для SQL-запроса «SELECT * FROM news WHERE autor='pupkin'»

Также, благодаря SQL-injection можно получить доступ к информации, находящейся в других полях таблицы. Предположим, имеется запрос:

SELECT * FROM student WHERE specialty='GT' and address='KZTZ'

Где в параметр «address» подставляются введенные пользователем данные. Злоумышленнику требуется узнать адреса студентов специальности IB «specialty='IB'», соответственно необходимо поменять логику этого запроса таким образом, чтобы СУБД вернула необходимую информацию. Чтобы реализовать такую SQL-injection следует, прежде всего, во вводимую строку добавить закрывающий спецсимвол «'» (для параметра поля «address») и добавочное условие, содержащее часть запроса целевой информации. В результате чего модифицированный SQL-запрос вернет информацию об адресах студентов, учащихся на специальности IB («specialty='IB'»):

SELECT * FROM student WHERE specialty='GT' and address='**1' or specialty=**'IB'

Таким образом, вернется интересующая информация через строковый параметр, соответствующий полю «address» в БД.

Числовой входящий параметр: запрос с числовым входящим параметром может выглядеть так:

SELECT * FROM news WHERE id=1

Обнаружить SQL-injection в данном случае возможно при помощи подстановки спецсимвола «'» в параметр «id», в результате чего возможно появление следующего сообщения об ошибке - «*mysql_query(): You have an error in your SQL syntax check the manual that corresponds to your MySQL server version for the right syntax to use near '1'*»

Если данного уведомления об ошибке нет, то возможны следующие предположения:

1) Приложение проверяет входящие данные от пользователя на наличие символа кавычки и не исполняет SQL-запрос (фильтрует символ);

2) Отчет об ошибках отключен или же функция отчета об ошибках не используется;

3) Проверяемый параметр не содержит уязвимость типа SQL-injection.

Для определения факта фильтрации символа кавычки следует передать в качестве входящего параметра любой текст, используя разделение пробелом, например:

SELECT * FROM news WHERE id=**1 sometext**

Так как запрос не является синтаксически верным, то появится ошибка - «*mysql_query(): You have an error in your SQL syntax check the manual that corresponds to your MySQL server version for the right syntax to use near '1 sometext*»

В случае, когда выключен отчет об ошибках, следует передать в качестве входящих данных следующее «1 -- »:

SELECT * FROM news WHERE id=1 **--**

Соответственно приложение отобразит информацию такую же, как и при запросе «SELECT * FROM news WHERE id=1». Следовательно, при передаче различных SQL конструкций можно анализировать работу приложения по ответам сервера.

Пример эксплуатации числового параметра:

SELECT * FROM news WHERE number_news=**1 or id=3**

Таким образом, злоумышленник получит информацию о новости с «id=3» через числовой параметр «number_news».

Определение типа и версии базы данных: для перехода на следующий этап проведения атаки SQL-injection необходимо понять с каким типом СУБД работает приложение. Так как в зависимости от типа будет меняться синтаксис инъекции.

Определить тип СУБД можно при помощи выводимой ошибки. Рассмотрим вывод из предыдущего пункта:

«*mysql_query(): You have an error in your SQL syntax check the manual that corresponds to your MySQL server version for the right syntax to use near '1 sometext*»

Изучив текст ошибки «*to your MySQL server*» можно понять, что приложение взаимодействует с СУБД MySQL.

Однако не всегда включен отчет об ошибках, поэтому для определения типа СУБД в этом случае следует составлять входные данные на основании синтаксиса характерного для определенной СУБД(MySQL, Oracle, PostgreSQL и т.д.) и анализировать работу приложения.

Таблица 2 - Различия СУБД

	MySQL	Oracle	PostgreSQL
Объединение строк	Concat(,) Concat_ws(delim,)	''||''	''||''
Комментарии	-- и /**/ и #	-- и /*	-- и /*
Объединение запросов	Union	union	Union и ;
Подзапросы	v.4.1 >=	Да	Да
Хранимые процедуры	Нет	Да	Да
Наличие information_schema или его аналога	v.5.0 >=	Да	Да

Для более точного определения версии и типа СУБД следует провести проверку при тестировании SQL-injection типа Union-based.

Эксплуатация: эксплуатация может различаться в зависимости от типа переменной, типа SQL-запроса, места SQL-injection запросе, но в основном можно выделить эксплуатацию с прямым вы-

водом на экран, когда данные из базы данных выводятся непосредственно на экран, и без прямого вывода, когда изменяется состояние базы данных при совершении каких-либо операций, но данные из базы данных не выводятся.

Union-based: SQL-injection типа union-based возможны в том случае, когда данные из базы выводятся непосредственно на экран. Предположим, что в базе данных существуют таблицы: news и users, содержащие 4 поля каждая. Оригинальный запрос выглядит так:

SELECT * FROM news WHERE id='1'

Злоумышленник знает названия полей таблиц и ему необходимо получить пароль пользователя с id=2, содержащийся в таблице users. Для использования оператора union в модифицированном запросе необходимо узнать количество полей в таблицах. Существует несколько способов:

1) Получение количества полей через оператор union;

2) Получение количества полей через оператор GROUP BY;

3) Получение количества полей через оператор ORDER BY.

Предположим, что количество полей в таблице news — 3:

SELECT * FROM news WHERE id=**1 union select 1,2,3 from news**

В случае, если отчет об ошибках включен, то приложение вернет следующее: «mysql_query(): The used SELECT statements have a different number of columns», что означает неправильное предположение о количестве полей. Аналогичные действия проводятся до тех пор, пока вывод ошибки не будет прекращен, что означает правильно подобранное количество полей. В случае, если отчет об ошибках выключен, то следует подбирать количество полей до тех пор, пока информация выводимая при запросе «SELECT * FROM news WHERE id='1'» не будет заменена на числовые значения после оператора select, таким же методом следует определить и выводимые столбцы базы данных.

Дальнейшая эксплуатация SQL-injection типа union-based зависит от определенного количества полей в таблицах.

В случае, когда количество полей таблиц совпадает или количество полей таблицы, используемой приложением больше коли-

чества полей таблицы, из которой необходимо получить данные, то модифицированный запрос будет выглядеть так:

SELECT * FROM news WHERE id=**-1 union select 1,id,password,4 from users**

Число «**-1**» в модифицированном запросе подставляется для того, чтобы исключить вывод ненужной информации для злоумышленника из оригинального запроса. Это может быть любое число, которое заведомо не содержится в таблице базы данных. Таким образом, вместо выводимых чисел необходимо подставить название полей таблицы, из которой нужно получить информацию.

В случае, когда количество полей таблицы, используемой приложением меньше количества полей таблицы, из которой необходимо получить данные, то следует воспользоваться объединением строк:

SELECT * FROM news WHERE id=**-1 union select 1,concat_ws(":", id, password),3,4 from user**s

Тем самым с помощью таких модифицированных запросов злоумышленник узнает пароль пользователя из другой таблицы.

Выше рассмотрен случай, когда злоумышленник заведомо знает имена полей таблиц, но чаще такой информации нет. Для этого необходимо обратиться к базе данных «information_schema» (информационная база данных), которая обеспечивает доступ к метаданным баз данных. Метаданные представляют собой данные относительно имени базы данных или таблицы, тип данных столбца или привилегии доступа. Внутри информационной базы данных имеется несколько таблиц только для чтения, в которых и содержатся данные о базах данных и таблицах. Пример использования:

SELECT column_name FROM information_schema.columns WHERE table_name='news'

Данный запрос вернет название полей таблицы «news».

В случае, когда доступ к «information_schema» закрыт, то для получения информации о именах таблиц базы данных следует прибегнуть к перебору имен по словарю.

Рассмотренный механизм может быть использован для определения типа и версии базы данных. Чтобы получить такую информацию необходимо вызвать функцию «version()» на место выводимых столбцов. Согласно пункту 4.2.2.1, таблицы 1 в зависимо-

сти от типа и версии СУБД синтаксис инъекции может меняться. Рассмотрим некоторые особенности эксплуатации для разных СУБД:

- PostgreSQL:
  SELECT * FROM news WHERE id=**-1; select 1,2,3,4**
- Oracle:
  SELECT * FROM news WHERE id=**-1 union select null, null, null from sys.dual**

Форма авторизации: одной из важных частей приложения является процесс аутентификации пользователей. В случае если фильтрация входных данных из формы авторизации не происходит должным образом, то возможен обход процесса аутентификации приложения. Предположим, что запрос к СУБД выглядит следующим образом:

SELECT * FROM users where login='admin' and pass='123'

Так как в большинстве систем есть учетная запись администратора с именем пользователя «admin», то злоумышленник может провести SQL-injection, обойдя проверку пароля с помощью спецсимволов « -- », «#» и т.д., которые в SQL обозначают комментарий. Соответственно запрос будет выглядеть так:

SELECT * FROM users where login='**admin' --** ' and pass='123'

Таким образом, злоумышленник смог обойти авторизацию и получить права администратора. Данный пример показывает уязвимость параметра «login», в случаи с параметром «pass» следует прибегнуть к такой логике изменения запроса:

SELECT * FROM users where login='admin' and pass='**1' or login='admin' --** '

Также при аутентификации может быть использован оператор «LIKE», который служит для сравнения строк:

SELECT * FROM users where login LIKE 'admin' and pass LIKE '123'

В таком случае, даже если приложение фильтрует кавычку, то инъекцию все равно можно провести благодаря спецсимволу «%», который оператор «LIKE» считает за любую строку, тогда наш запрос будет выглядеть так:

SELECT * FROM users where login LIKE 'admin' and pass LIKE '%'

Таким образом, данная SQL-injection приведет к обходу проверки пароля.

Error-based: не всегда через точку входа приложения возможен вывод информации из базы данных, но в случае если включен отчет об ошибках можно воспользоваться вектором Error-based. Основная идея этого вектора – вывод информации через выводимый текст ошибки. Существует множество техник и подходов к эксплуатации данной уязвимости, рассмотрим одну из них.

Предположим, есть таблица «news» содержащая 3 поля, оригинальный запрос выглядит так:

SELECT * FROM news WHERE theme='black'

Теперь искусственно вызовем ошибку, через которую сможем получить информацию:

SELECT * FROM news WHERE theme='**-1' UNION SELECT 1,2,COUNT(*) FROM (SELECT 1 UNION SELECT 2)x GROUP BY MID(VERSION(), FLOOR(RAND(0)*2), 64) --** '

Ошибка возникает вследствие записи одинакового значения в поле уникального индекса «group_key» промежуточной таблицы «temporary table», которая временно создается СУБД для заполнения результатов вычисления функции COUNT.

Вывод ошибки «*ER_DUP_ENTRY: Duplicate entry '10.1.26-MariaDB-0+deb9u1' for key 'group_key'*». В результате чего вернулась версия используемой СУБД.

Таким образом, можно получить интересующую информацию через вывод ошибки. Недостатком данного метода является размер выводимой информации за одну инъекцию и составляет 64 символа. Для вывода последующих символов необходимо прибавить 64 к функции RAND «(RAND(0)*2)+64».

Blind-based: в том случае, когда через точку входа приложения не происходит вывод данных из базы на экран, а также отчет об ошибках выключен, то следует проверить вектор blind-based.

Для обнаружения данного вектора атаки существует два основных способа:

1) На основании изменений работы приложения;

2) По времени.

Рассмотрим пример обнаружения вектора blind-based на основании изменений работы приложения. Предположим, что оригинальный запрос к СУБД выглядит следующим образом:

SELECT * FROM news WHERE id=1

В ответ на данный запрос сервер отправляет служебную информацию, которая никак не отображается в приложении. Для обнаружения SQL-injection в данном случае следует передать логическую конструкцию or или and следующим образом:

SELECT * FROM news WHERE id=**1 and 1=1**

В данном случае логическая операция вернет значение True, что никак не отобразится на приложении.

SELECT * FROM news WHERE id=**1 and 1=2**

В данном случае логическая операция вернет значение False. Так как значение ложное, то возможно некоторое нарушение в работе приложения. Как только такое изменение было обнаружено, следует перейти к подбору интересуемой информации. Для этого необходимо использовать функции askii() и substring(). Тогда модифицированный запрос будет выглядеть:

SELECT * FROM news WHERE id=**1 and ASCII(substring((SELECT USER()),1,1))=114;**

В случае если числовое значение первого символа имени пользователя базы данных по таблицы askii будет равно 114, тогда вернется True, иначе False, таким образом, можно перебирать имена таблиц, столбцов, значение полей и т.д. Для перебора следующего символа необходимо изменить значение начальной позиции функции substring():

SELECT * FROM news WHERE id=**1 and ASCII(substring((SELECT USER()),2,1))=114;**

Рассмотренный пример базируется на изменении работы приложения путем изменения логики запроса, но иногда таковых нет. В таком случае следует использовать функцию sleep(), которая позволяет по задержкам определить правильность передаваемого условия:

SELECT * FROM news WHERE id=**1 or if(ASCII(substring((SELECT USER()),1,1))=114,sleep(5),null);**

Таким образом, в случае возвращения условием значение True, будет выполнена задержка на 5 секунд, соответственно, если False таковой не будет.

Для эксплуатации вектора blind-based следует использовать автоматизированные инструменты (sqlmap, iSQL и т.п.).

Обход защитных средств: для обеспечения защиты базы данных от SQL-injection применяются разнообразные способы фильтрации входных пользовательских данных. Рассмотрим некоторые из них.

Оригинальный запрос выглядит так:

SELECT * FROM news WHERE id=1

В случае, когда пользовательские данные фильтруются (проверяются на совпадение символы) функцией, которая проверяет строку на наличие символа пробела, следует заменить данный символ на символы комментария «/**/» или «/*!*/».

SELECT * FROM news WHERE id=**-1/**/or/**/id=2#**

Также помимо пробела есть ещё несколько пробельных символов, которые MySQL воспринимает как пробел:

%09 – табуляция;

%0A – символ новой строка;

%0D – возврат каретки;

%0B – вертикальная табуляция;

%0C – символ новой страницы.

Если всё вышеперечисленное фильтруется приложением, то следует воспользоваться скобками или апострофами:

SELECT * FROM news WHERE id=(-**1)union(select(1),2,(3)from(users)where(id=2))**

В случае, когда пользовательские данные фильтруются функцией, которая проверяет строку на наличие знака равенства, то следует применять знаки отрицания и неравенства (!=, <>,<,>) или оператор like.

SELECT * FROM news WHERE id=**3 and 1<2#**

SELECT * FROM news WHERE id=**-1 or id like 2#**

Так как язык SQL может обрабатывать строки в шестнадцатеричном представлении (hex), то при уязвимом числовом параметре с функциями, фильтрующими символы «'» и «"» для сравнения

строк следует использовать hex представление строки или функцию char().

SELECT * FROM news WHERE id=**-1 or id like 0x32#**

Где «0x32» 16-ое представление числа 2.

В случае, когда пользовательские данные фильтруются функцией, которая проверяет строку на наличие в ней ключевых слов или операторов SQL, следует менять шрифт в модифицированном запросе или разделять операторы символами комментариев:

SELECT * FROM news WHERE id=**-1 UnIoN SelECT 1,2,3**

SELECT * FROM news WHERE id=**-1 un/**/ion sele/**/ect 1,2,3**

Так как атака типа SQL-injection позволяет атакующему несанкционированно получать, изменять, записывать некоторые свои данные в базу данных, то вся работа веб-приложения становится под угрозой. Для предотвращения данной уязвимости веб-фреймворк Laravel использует PDO параметры для доступа к базам данных, что исключает возможность модифицировать запрос.

Таким образом, использование Laravel позволит исключить возникновение уязвимости SQL-injection.

7.2. Защита персональных данных

Мало какая система сегодня обходится без персональных данных людей. В федеральном законе №152 от 27.06.2006 «О персональных данных» в пункте 1 статьи 19 говорится о том, что оператор при обработке персональных данных обязан принимать необходимые правовые, организационные и технические меры для защиты персональных данных от неправомерного доступа к ним, уничтожения, изменения, блокирования, копирования, предоставления, распространения персональных данных.

Для обеспечения безопасности персональных данных необходимо:

1) определение угроз безопасности персональных данных при их обработке в информационных системах персональных данных;

2) применение организационных и технических мер по обеспечению безопасности персональных данных при их обработке в информационных системах персональных данных, необходимых

для выполнения требований к защите персональных данных, исполнение которых обеспечивает установленные Правительством Российской Федерации уровни защищенности персональных данных;

3) применение прошедших в установленном порядке процедуру оценки соответствия средств защиты информации;

4) оценка эффективности принимаемых мер по обеспечению безопасности персональных данных до ввода в эксплуатацию информационной системы персональных данных;

5) учёт машинных носителей персональных данных;

6) обнаружение фактов несанкционированного доступа к персональным данным и принятием мер;

7) восстановление персональных данных, модифицированных или уничтоженных вследствие несанкционированного доступа к ним;

8) установление правил доступа к персональным данным, обрабатываемым в информационной системе персональных данных, а также обеспечением регистрации и учета всех действий, совершаемых с персональными данными в информационной системе персональных данных;

9) контроль за принимаемыми мерами по обеспечению безопасности персональных данных и уровня защищенности информационных систем персональных данных.

Так же необходимым условием для работы с персональными данными в любой системе является наличие согласие субъекта на обработку персональных данных.

Согласие может быть дано либо в письменном виде, либо в электроном виде, но тогда оно должно быть подписано с помощью электронной подписи. Согласие на обработку персональных данных может быть отозвано субъектом персональных данных, и в этот случае оператор должен прекратить обработку персональных данных и безвозвратно удалить их [13].

Согласно 4 пункту 9 статьи федерального закона «О защите персональных данных» согласие в письменной форме субъекта персональных данных на обработку его персональных данных должно включать в себя следующее:

1) фамилию, имя, отчество, адрес субъекта персональных данных, номер основного документа, удостоверяющего его личность, сведения о дате выдачи указанного документа и выдавшем его органе;

2) фамилию, имя, отчество, адрес представителя субъекта персональных данных, номер основного документа, удостоверяющего его личность, сведения о дате выдачи указанного документа и выдавшем его органе, реквизиты доверенности или иного документа, подтверждающего полномочия этого представителя (при получении согласия от представителя субъекта персональных данных);

3) наименование или фамилию, имя, отчество и адрес оператора, получающего согласие субъекта персональных данных;

4) цель обработки персональных данных;

5) перечень персональных данных, на обработку которых дается согласие субъекта персональных данных;

6) наименование или фамилию, имя, отчество и адрес лица, осуществляющего обработку персональных данных по поручению оператора, если обработка будет поручена такому лицу;

7) перечень действий с персональными данными, на совершение которых дается согласие, общее описание используемых оператором способов обработки персональных данных;

8) срок, в течение которого действует согласие субъекта персональных данных, а также способ его отзыва, если иное не установлено федеральным законом;

9) подпись субъекта персональных данных.

7.3. Разработка моделей

Так что же такое модель? Если мы собираемся построить всё приложение на моделях, сначала мы должны понять, что именно они из себя представляют.

Модель должна содержать всю бизнес-логику вашего приложения. Или, другими словами, то, как приложение взаимодействует с базой данных.

Бизнес-логика – это:

1. Объекты реального мира, которые используются в вашем приложении
2. То, как эти объекты взаимодействуют друг с другом

3. Набор правил для доступа к этим объектам и для их обновления

Поэтому, например, Users станет важным объектом в Cribbb, потому что это социальное приложение.

1. Мы должны хранить информацию о наших пользователях, и поэтому нам нужна модель User и таблица User в базе данных.

2. Пользователям надо будут вводить имя, адрес электронной почты и пароль, а также другие детали профиля. Чтобы удостовериться, что они вводят правильно отформатированные данные, мы должны проверять их ввод.

3. Пользователи смогут создавать сообщения. Пользователь может иметь много сообщений, и каждое сообщение должно принадлежать пользователю.

Это основные особенности работы моделей в приложениях MVC. По существу для каждой важной вещи в приложении, вероятно, потребуется модель. Вам, возможно, понадобится проверять данные, используемые в вашей модели, а так же здесь должна быть вся логика, отвечающая за взаимодействие моделей друг с другом.

Проверка подлинности пользователя требуется почти в каждом современном веб-приложении. Вместо того чтобы заставлять вас писать свою собственную модель пользователя, в Laravel уже есть модель пользователя прямо из коробки.

Если вы зайдете в каталог app/models, то найдете там файл User.php. Все ваши модели должны помещаться в этом каталоге и именоваться таким же образом. Например, в вашем приложении есть модель для сообщений, тогда файл модели будет называться app/models/Post.php.

Как бы то ни было, если вы откроете модель пользователя, вы увидите довольно базовый шаблон модели.

После подключения модуля авторизации в формате «из коробки» автоматически создается модель User, которая описывает механизмы работы с пользователями.

Типовая структура модели, как правило, состоит из объявления пространства имен для подключаемых классов, списка подключаемых классов, некоторых правил и служебных переменных, после чего следует набор функций, описывающих конкретные ме-

ханизмы работы с базой данных. Явно объявляется название таблицы, с которой будет осуществляться взаимодействие (если она одна) и полей, в которые можно либо нельзя вносить данные. Другими словами, protected $filelable – это локальная защищенная переменная модели, которая является массивом с именами/названиями столбцов таблицы `users`, разрешенных для массового заполнения. Основным содержимым любой модели является набор функций, отвечающих за определенный функционал. Для модели User нам понадобится в список полей, разрешенных для массового заполнения, внести следующие поля: `name`, `email`, `password`, `status`, `gruppa`, `avatar`, `reg_ip`. В список скрытых в интересах безопасности полей следует внести `password` и `remember_token`.

Функция getUser обеспечивает формирование списка всех зарегистрированных в системе пользователей для дальнейшего с ним взаимодействия. Ниже представлен скриншот данной модели.

```
class User extends Authenticatable
{
    use Notifiable;

    public function getUser()
    {
        $users = User::all();
        return $users;
    }

    # Список полей в таблице `users`, подлежащих записи.
    protected $fillable = [
        'name', 'email', 'password', 'status', 'gruppa', 'avatar', 'reg_ip',
    ];

    # Список скрытых в интересах безопасности полей.
    protected $hidden = [
        'password', 'remember_token',
    ];
}
```

Рисунок 37 – Модель User

В качестве примера мы рассмотрим создание модели Students. Создание модели происходит с помощью простой команды:
php artisan make:model Students

Рисунок 38 – Создание модели

После выполнения типовых действий по объявлению используемых классов зададим используемую таблицу:

protected $table = 'users'

Таким образом, будет объявлено, что в рамках данной модели будет использоваться таблица `users`. В нашем проекте список студентов, старост и преподавателей для большей гибкости, и простоты обращения объединены со списком пользователей. Список методов, которые будут задействованы в данной модели: getStudents, getStudent, getStudentsByGroup, getMassStudentsByGroup, getStudentsByKaf, getUsernames, getUnverified, getUnverifiedSum, declineUser, verifyUser.

- Функция getStudents возвращает список пользователей из таблицы `users` с сортировкой по идентификатору в порядке возрастания.
- Функция getStudent осуществляет вывод данных о конкретном пользователе по переданному ей идентификатору.
- Функция getStudentsByGroup осуществляет вывод списка студентов, объединенных общим значением – идентификатором группы. Таким образом, данная функция позволит вывести всех студентов из одной группы в список, отсортированный в порядке возрастания по идентификатору.
- Функция getMassStudentsByGroup осуществляет вывод только студентов и старост (преподаватели в данном выводе не будут отображаться) по общему признаку – идентификатору группы. Однако, в отличие от предшествующей функции, данная работает не с отдельным значением, а со списком групп, что позволяет осуществлять получение данных о студентах одновременно нескольких групп с объединением в общий список.
- Функция getStudentsByKaf обеспечивает возвращение данных о пользователях по общему признаку – идентификатору группы. Вывод данной функции используется при формировании списка пользователей, в аккаунты которых другой пользователь может осуществлять загрузку файлов и документов, поэтому для препо-

давателей добавлено отдельное правило, позволяющее добавить к списку также и аккаунт того преподавателя, для которого формируется список. В конечном итоге, это позволяет преподавателю загружать файлы и документы также и в собственный аккаунт.

• Функция getUsernames возвращает ассоциативный массив информации о пользователях, позволяющий по идентификатору пользователя установить его имя и получить доступ к адресу загруженного пользователем аватара. Используется для отображения информации о пользователях в ленте документов.

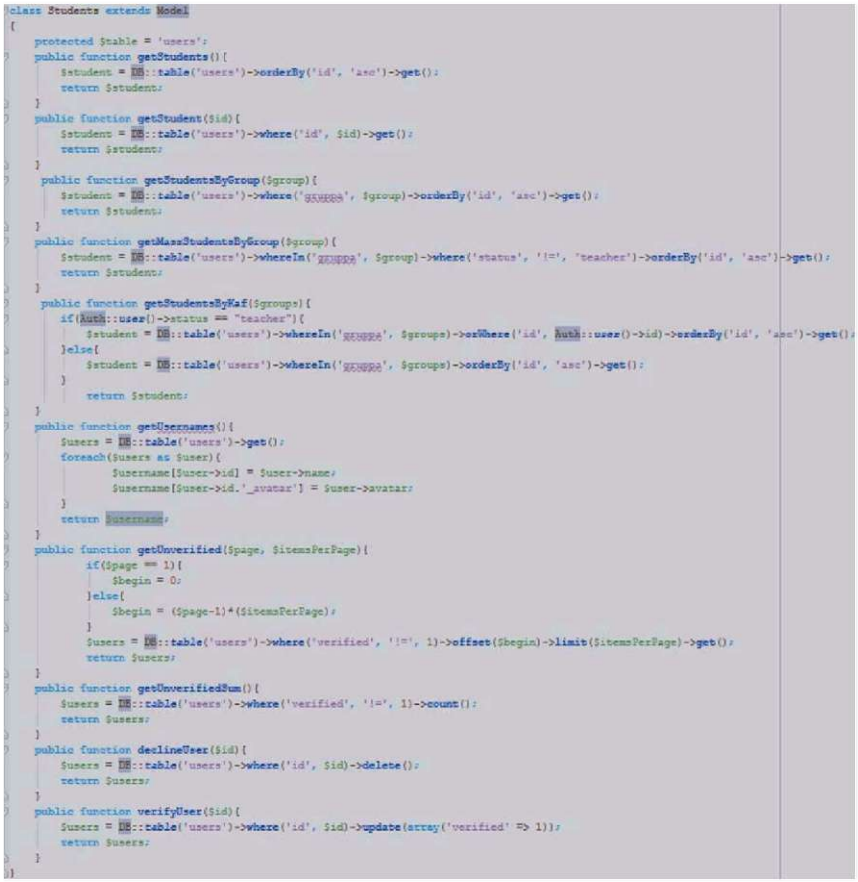
• Функция getUnverified позволяет вывести список пользователей, отправивших заявки на создание учетных записей, но еще не подтвержденных администратором ресурса. Поскольку данной функцией поддерживается вывод информации в постраничном режиме, то переменные $page и $itemsPerPage обеспечивают, соответственно, передачу в функцию информации о просматриваемой в данный момент странице и о количестве вхождений на страницу, которые должны быть возвращены в ответ. Если пользователь на первой странице, то мы начинаем выводить записи с самого начала (то есть, с нулевого идентификатора). Иначе же все происходит по формуле, согласно которой номер первой отображаемой записи равен номеру страницы минус один и умноженному на количество вхождений на страницу.

• Функция getUnverifiedSum возвращает количество пользователей, аккаунты которых не были верифицированы администратором ресурса. Используется в постраничной навигации.

• Функция declineUser реализует отказ в верификации аккаунта пользователя, имеющего идентификатор $id. Вызывается из панели верификации регистрации преподавателей и старост.

• Функция verifyUser реализует подтверждение (верификацию) регистрации преподавателя либо старосты. Вызывается из панели верификации регистраций преподавателей и старост.

Ниже представлен скриншот кода данной модели.



```
class Students extends Model
{
    protected $table = 'users';
    public function getStudents(){
        $student = DB::table('users')->orderBy('id', 'asc')->get();
        return $student;
    }
    public function getStudent($id){
        $student = DB::table('users')->where('id', $id)->get();
        return $student;
    }
     public function getStudentsByGroup($group){
        $student = DB::table('users')->where('группа', $group)->orderBy('id', 'asc')->get();
        return $student;
    }
    public function getManyStudentsByGroup($group){
        $student = DB::table('users')->whereIn('группа', $group)->where('status', '!=', 'teacher')->orderBy('id', 'asc')->get();
        return $student;
    }
     public function getStudentsByKaf($groups){
        if(Auth::user()->status == "teacher"){
            $student = DB::table('users')->whereIn('группа', $groups)->orWhere('id', Auth::user()->id)->orderBy('id', 'asc')->get();
        }else{
            $student = DB::table('users')->whereIn('группа', $groups)->orderBy('id', 'asc')->get();
        }
            return $student;
    }
    public function getUsernames(){
        $users = DB::table('users')->get();
        foreach($users as $user){
                $username[$user->id] = $user->name;
                $username[$user->id.'_avatar'] = $user->avatar;
        }
        return $username;
    }
    public function getUnverified($page, $itemsPerPage){
            if($page == 1){
                $begin = 0;
            }else{
                $begin = ($page-1)*($itemsPerPage);
            }
            $users = DB::table('users')->where('verified', '!=', 1)->offset($begin)->limit($itemsPerPage)->get();
            return $users;
    }
    public function getUnverifiedSum(){
        $users = DB::table('users')->where('verified', '!=', 1)->count();
        return $users;
    }
    public function declineUser($id){
        $users = DB::table('users')->where('id', $id)->delete();
        return $users;
    }
    public function verifyUser($id){
        $users = DB::table('users')->where('id', $id)->update(array('verified' => 1));
        return $users;
    }
}
```

Рисунок 39 – Модель Students

Далее в нашем тестовом проекте будут использоваться и другие модели, исходный код которых можно посмотреть по следующей ссылке:

Следующим этапом необходимо добавит такие фундаментальные для нашего проекта модели, как модель взаимодействия с кафедрами, факультетами и группами, ведь вся дальнейшая работа будет завязана именно на них.

Начнем с создания модели Facultets (рисунок 40).

```
C:\OpenServer\domains\laravel>php artisan make:model Facultets
Model created successfully.
```

Рисунок 40 – Создание модели Facultets

Настоящая модель является достаточно миниатюрной, поскольку не содержит большого количества функций, ведь работа с факультетами внутри проекта не носит слишком объемного характера. Все, что необходимо сделать, это объявить используемую таблицу:

protected $table = 'facultets';

После чего создаем всего две функции: getFacultets и getFacultet.

- Функция getFacultets возвращает при обращении к себе список факультетов, отсортированных по идентификатору в обратном порядке.
- Функция getFacultet возвращает строку из таблицы `facultets`, где идентификатор равен запрашиваемому. Используется для получения данных о конкретном факультете.

Полное содержание данной модели можно увидеть на рисунке 41.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use DB;

class Facultets extends Model
{
    protected $table = 'facultets';

    public function getFacultets(){
        $facultet = DB::table('facultets')->orderBy('id', 'desc')->get();
        return $facultet;
    }

    public function getFacultet($id){
        $facultet = DB::table('facultets')->where('id', $id)->first();
        return $facultet;
    }

}
```

Рисунок 41 – Модель Facultets

Модель Kafs, описывающая взаимодействие с таблицей кафедр университета, по сути дела, не отличается существенно от предшествующей модели Facultets, поэтому процесс ее создания будет очень похожим:

```
C:\OpenServer\domains\laravel>php artisan make:model Kafs
Model created successfully.
```

Рисунок 42 – Создание модели Kafs

Аналогично модели Facultets, в модели Kafs задается используемая таблица:

protected $table = 'kafs';

После чего создаются две функции: getKaf и getKafs.

- Функция getKafs возвращает список кафедр, зарегистрированных в системе, с сортировкой по названию в алфавитном порядке.
- Функция getKaf выводит информацию о конкретной кафедре из базы данных, выбирая ее по идентификатору.

Полное содержание данной модели можно увидеть на рисунке 43.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use DB;

class Kafs extends Model
{
    protected $table = 'kafs';

    public function getKafs(){
        $kaf = DB::table('kafs')->orderBy('name', 'asc')->get();
        return $kaf;
    }

    public function getKaf($id){
        $kaf = DB::table('kafs')->where('id', $id)->first();
        return $kaf;
    }

}
```

Рисунок 43 – Модель Kafs

Наконец, перейдем к более ответственному этапу – созданию модели Groups, описывающей взаимодействие с группами.

```
C:\OpenServer\domains\laravel>php artisan make:model Groups
Model created successfully.
```

Рисунок 44 – Создание модели Groups

Настоящая модель должна описывать несколько более обширный функционал, поскольку основная единица в структуре университета, с которой нам предстоит работать в нашем проекте – это именно группа.

Также, как и в прошлых случаях, мы задаем используемую таблицу следующим образом:

protected $table = 'groups';

В нашем случае, модель Groups будет состоять из следующего списка функций: getGroups, getGroup, getGroupsByKaf и getTeacherGroups.

- Функция getGroups возвращает список групп, отсортированный в алфавитном порядке.
- Функция getGroup выводит информацию об отдельной группе, поиск происходит по идентификатору.
- Функция getGroupsByKaf выводит список групп в рамках одной кафедры. Выборка происходит по идентификатору кафедры (kafedra_id).
- Функция getTeacherGroups выводит информацию о группах, установленных в профиле преподавателя в качестве подотчетных. В качестве входящих данных принимает список идентификаторов групп (строка), разделенных между собой запятыми. В последствии, список превращается в массив при помощи разбиения по запятой. Такой механизм был выбран как максимально удобный из-за того, что в базе данных пользователей данные о подотчетных группах преподавателя хранятся именно в таком формате.

Полное содержание данной модели можно увидеть на рисунке 45.

Рисунок 45 – Модель Groups

Следующим этапом создадим модель Post.

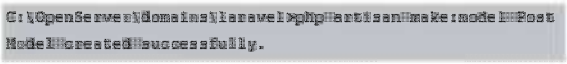


Рисунок 46 – Создание модели Post

В данной модели будет использоваться одноименная таблица `post`:

protected $table = 'posts';

Среди функций создаются следующие: getPost, create и ifInTeacherGroups.

- Функция getPost выводит список записей из таблицы `post` с сортировкой по идентификатору. В случае, если функции

была передана переменная $paginate, осуществляется также постраничное разбиение вывода.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Facades\Auth;
use DB;

class Post extends Model
{
    protected $table = 'posts';

    public function getPost($paginate){
        if($paginate == 0){
            $post = DB::table('posts')->orderBy('id', 'desc')->get();
        }else{
            $post = DB::table('posts')->orderBy('id', 'desc')->paginate($paginate);
        }
        unset($paginate);
        return $post;
    }

    public function create($info){
        $groupId = $info['gruppa'];
        if(Auth::user()->id == $info['user_id']){
            DB::table('users')->where('id', Auth::user()->id)->update([
                'gruppa'=>$groupId,
                'updated_at' => DB::raw ('CURRENT_TIMESTAMP'),
            ]);
        }
        return;
    }

    public function ifInTeacherGroups($id){
        $tgrp = DB::table('users')->where('id', Auth::user()->id)->first();
        if($tgrp->teacher_groups == "none"){
            $return = 'false';
        }else{
            $tgroups = explode( delimiter: ',', $tgrp->teacher_groups);
            if(in_array($id,$tgroups)){
                $return = 'true';
            }else{
                $return = 'false';
            }
        }
        return $return;
    }
}
```

Рисунок 47 – Модель Post

- Функция create осуществляет обновление значения поля `gruppa` в таблице `users` в случае, если идентификатор авторизованного пользователя, производящего настоящее действие, равен переданному в массиве $info.
- Функция ifInTeacherGroups осуществляет проверку, входит ли передаваемый данной функции идентификатор группы $id в список подконтрольных групп преподавателя, от имени которого функция в данный конкретный момент исполняется. Также уста-

новлена проверка, является ли авторизованный пользователь преподавателем.

С полным кодом модели можно ознакомиться на рисунке 46.

Одной из наиболее значимых в рамках данного проекта является модель Uploads, поскольку работает она с загруженными файлами и документами, что является основой всего проекта. Соответственно, данная модель является наиболее загруженной в сравнении со всеми остальными, что вполне логично.

Прежде всего, установим используемую таблицу:

protected $table = 'uploads';

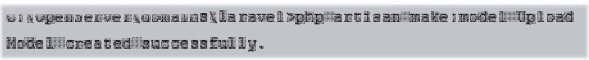


Рисунок 48 – Создание модели Upload

В рамках данной модели планируется реализация ряда функций, а именно: create, getUploadsById, getUploadsByOriginatingId, displayUploadsById, displayUploadsByOriginatingId, getUploadsByVisiblity, getUploadsCountByVisiblity, summaryCount, getFile, removeFile и getFileById.

- Функция create принимает массив данных $info, содержащий информацию о загружаемом файле или документе и обеспечивает регистрацию данных в таблице `uploads`.
- Функция getUploadsById обеспечивает вывод последних десяти загруженных файлов либо документов по одному конкретному пользователю, обозначенному идентификатором.
- Функция getUploadsByOriginatingId обеспечивает вывод десяти последних загруженных файлов либо документов по реальному (а не отображаемому, как getUploadsById) автору, обозначенному идентификатором.
- Функция displayUploadsById обеспечивает вывод всех загруженных файлов и документов с сортировкой по дате размещения в системе в порядке убывания по одному конкретному пользователю, обозначенному идентификатором. Функция поддерживает постраничный вывод, поэтому кроме $id (идентификатор пользователя) и $type (тип запрашиваемых загрузок) принимает также номер просматриваемой страницы ($page) и количество вхождений

на одну страницу вывода ($itemsPerPage), что позволяет обеспечивать постраничный вывод данных.

• Функция displayUploadsByOriginatingId осуществляет вывод списка загруженных документов либо файлов пользователя (поддерживается выбор типа загрузок) по общему признаку -- идентификатору реального автора (а не по идентификатору отображаемого, как displayUploadsById) с постраничным разбиением и сортировкой по дате добавления в порядке убывания.

• Функция getUploadsByVisiblity обеспечивает вывод загрузок пользователей с учетом указанных прав доступа по части области видимости. Данной функцией определяется статус пользователя в системе, на основании чего запрос на выборку данных меняется, поскольку для разных статусов предусмотрены различные уровни доступа. Сортировка данных в выводе происходит по дате размещения документа в системе в порядке убывания, также функцией поддерживается постраничный вывод. Используется в ленте загрузок.

• Функция getUploadsCountByVisiblity является дополнением к функции getUploadsByVisiblity и возвращает количество доступных к показу загруженных файлов на тех же условиях.

• Функция summaryCount выводит число, обозначающее количество файлов в системе, которые связаны с пользователем, идентификатор учетной записи которого передается через переменную $id.

• Функция getFile возвращает информацию о конкретной строке из таблицы `uploads`, содержащей информацию о загруженном файле либо документе и его авторах (реальном и отображаемом), с поиском по имени файла.

• Функция removeFile удаляет из таблицы `uploads` вхождение о том или ином загруженном документе либо файле.

• Функция getFileById возвращает информацию о конкретной строке из таблицы `uploads`, содержащей информацию о загруженном файле либо документе и его авторах (реальном и отображаемом), с поиском по идентификатору вхождения в таблице `uploads`.

С полным кодом данной модели можно ознакомиться в репозитории в виду ее излишней громоздкости, как для иллюстрации.

7.4. Контрольные вопросы

1. Что такое SQL-injection?
2. Какие бывают виды SQL-injection?
3. Почему возникает SQL-injection?
4. Где может возникать SQL-injection?
5. Какие меры защиты баз данных использует Laravel?
6. Что такое модель?
7. Что такое бизнес-логика?
8. Основные особенности работы моделей в приложениях MVC.
9. Из чего состоит типовая структура модели?
10. Что выполняет функция getUser?
11. При помощи какой команды происходит создание модели?

8. РАЗРАБОТКА КОНТРОЛЛЕРОВ

8.1. Основные понятия

Контроллер – это класс, используемый для размещения логики маршрутизации. Как правило, контроллер содержит несколько общедоступных методов, известных как действия. Действия можно рассматривать как прямую альтернативу замыканиям, они очень похожи по внешнему виду и функциональности [14].

Контроллеры позволяют вынести код обработчика роутера в отдельный файл. Контроллер решает какой вид использовать и какие данные запросить у модели. Вместо функции запрос пользователя обрабатывает контроллер.

Контроллеры приложения хранятся в директории /app/Http/Controllers. Создание нового контроллера происходит с помощью команды: php artisan make:controller ControllerName, где ControllerName – это имя контроллера.

Контроллеры обычно хранятся в папке app/controllers, а этот путь по умолчанию зарегистрирован в настройке **classmap** вашего файла composer.json. Тем не менее контроллеры технически могут жить в любом каталоге или подкаталоге. Объявления маршрутов не зависят от расположения на диске файла класса контроллера. Поэтому, пока Composer знает, как автоматически загружать класс контроллера, этот файл может быть размещён где угодно.

Для примера создадим новый контроллер ArticleController (рисунок 49).

Рисунок 49 – Результат создания контроллера

После создания контроллера, необходимо реализовать логику его работы. Ниже представлен пример контроллера.

```
class ArticleController extends Controller
{
  public function showIndex(){
    return view('index');
  }

  public function showSingle($articleId){
    return view('single', ['articleId' => $articleId]);
  }
}
```

У нас имеется две функции, showIndex() и showSingle(), первая отвечает за отображение всех существующих статей, а вторая за конкретную статью. Оба этих действия связанны с концепцией статьи, именно поэтому их логика содержится в одном файле.

Все контроллеры должны наследовать класс BaseController. Этот класс также может хранится в папке app/controllers и в него можно поместить общую логику для других контроллеров. BaseController расширяет стандартный класс Laravel, Controller.

Контроллеры предоставляют простой способ объединений всей логики вместе, но они бесполезны, если не использовать их вместе с маршрутами. К счастью, метод связывания URI с контроллером, аналогичен методу маршрутизации, используемому для функций замыкания.

Чтобы связать URI с контроллером, мы должны объявить новый маршрут, однако, в этот раз, во второй параметр мы должны передать строку. Это строка состоит из двух частей и разделяется знаком (@), первой частью является имя контроллера, а второй используемое действие. Контроллеры поддерживают все методы класса Route. Ниже представлены пример таких маршрутов.

```
Route::get('/','ArticleController@showIndex');
Route::post('article/new','ArticleController@newArticle');
```

Laravel предоставляет решение для создания RESTful контроллеров. Зачастую, создавая контроллер, нам бывают необходимы CRUD (Create, Read, Update, Delete) действия.

Создадим новый контроллер и добавим в него следующий код:

```
    public function index() {
  return 'index';
}
public function create() {
  return 'create';
}
public function store(Request $request) {
  return 'store';
}
public function show($id) {
  return 'show';
}
public function edit($id) {
  return 'edit';
}
public function update(Request $request, $id) {
  return 'update';
```

```
}
public function destroy($id) {
   return 'destroy';
}
```

И объявим новый маршрут:

Route::*resource*('/','ArticleController');

Этот единственный маршрут, будет обеспечивать все действия, представленные в нашем контроллере. Как вы можете видеть, использование такого метода для маршрутизации к вашим контроллерам, обеспечивает явное преимущество перед оригинальным методом маршрутизации.

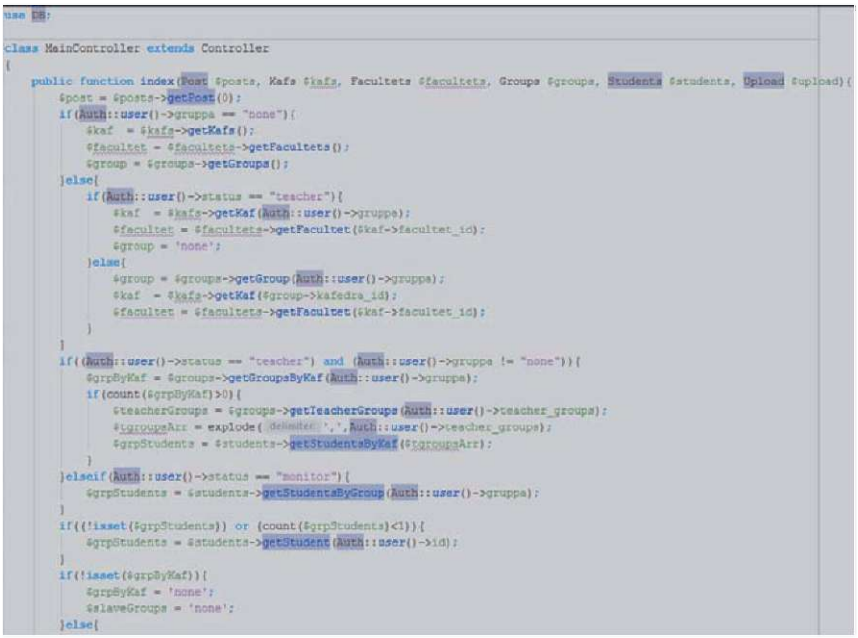


Рисунок 50 – Контроллер MainController

По своей структуре контроллер очень близок к модели, но у них есть много радикальных отличий: модель работает напрямую с БД, а контроллер работает с данными, которые преимущественно

предоставляются моделью. Ниже представлена часть кода контроллера MainController, который отвечает за основной функционал сайта.

На данном рисунке показан процесс инициализации метода index, который в качестве параметров принимает данные, формируемые следующими моделями: Post, Kafs, Facultets, Groups Students и Upload. Результат исполнения отправляется в шаблон 'profile' в качестве параметров.

```
C:\OpenServer\domains\laravel>php artisan make:controller MainController
Controller created successfully.
```

Рисунок 51 – Создание контроллера MainController

В контроллере может объявляться огромное число функций. В рамках данного курсового проекта помимо функции index, в данном контроллере существуют такие функции, как setGroup, update_avatar, loadDocument, setTeacherGroups и remove. Почти все эти функции принимают в качестве входящих параметров экземпляры классов Request либо Post, далее происходит обращение к методу с передачей ему данных отправленных с форм, затем используется метод редирект для возврата на страницу.

- Функция index обеспечивает вывод заглавной страницы профиля пользователя в полном объеме, учитывая при этом статус пользователя в системе. Результатом работы данной функции является передача в соответствующее представление полностью готового блока данных, на основании которого происходил формирование информации, отображаемой пользователю. Поскольку на данной странице осуществляется вывод только последних десяти загруженных файлов либо документов, постраничная навигация для данного представления не предусматривается.
- Функция setGroup исполняет обязанности обработчика POST-запроса на установку пользователем места в структуре университета.
- Функция update_avatar обеспечивает загрузку и обновление аватара пользователя средствами личного кабинета.
- Функция loadDocument описывает механизм загрузки файлов и документов в систему. Скриптом определяется тип за-

гружаемого файла либо документа, проверяется его соответствие указанным в форме данным и выполняется запись полученных данных в соответствующую таблицу базы данных.

- Функция setTeacherGroups устанавливает список подконтрольных преподавателю групп, получая его посредством POST-запроса из соответствующей формы в виде массива и преобразуя в установленный строчный формат.
- Функция remove реализует удаление загруженных файлов с сервера. В данном случае, реализуется только физическое удаление самого контента, соответствующие вхождения в базе данных данной функцией не затрагиваются, информация об успешном удалении лишь передается другой функции.

С полным кодом контроллера можно ознакомиться в репозитории.

Вторым по значимости является контроллер HomeController, описывающий работу страниц секции последних загрузок пользователей. По сути дела, контроллер состоит лишь из одной основообразующей функции – index, поскольку весь функционал данной секции сводится лишь к отображению контента.

```
C:\OpenServer\domains\laravel>php artisan make:controller HomeController
Controller created successfully.
```

Рисунок 52 – Создание контроллера HomeController

- Функция index обеспечивает формирование домашней страницы проекта (Ленты файлов), на которой отображаются последние загрузки пользователей с учетом их области видимости, а также с учетом статуса пользователя.

На рисунке 53 можно ознакомиться с образцом кода данного контроллера.

Рисунок 53 – Контроллер HomeController

Немаловажной частью в данном проекте является и контроллер MyController, который обеспечивает формирование страниц с загрузками конкретного пользователя.

```
C:\OpenServer\domains\laravel>php artisan make:controller MyController
Controller created successfully.
```

Рисунок 54 – Создание контроллера MyController

Данный контроллер также является достаточно легким по количеству встроенных функций, поскольку обеспечивает лишь обработку события отображения страниц личного кабинета проекта посредством функции index.

- Функция index обеспечивает формирование страницы с собственными загрузками пользователя.

На рисунке 55 можно ознакомиться с образцом кода данного контроллера.

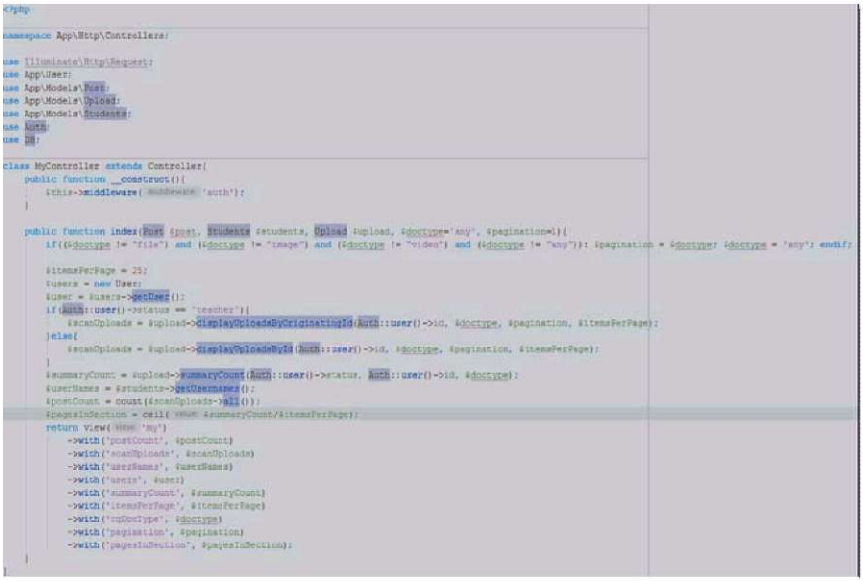


Рисунок 55 – Контроллер MyController

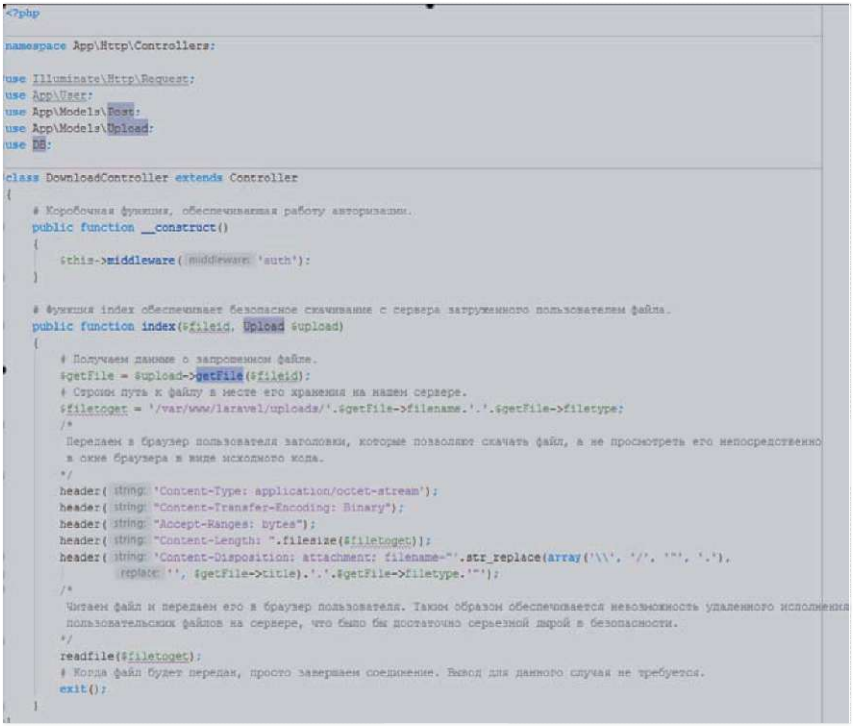
Следующий контроллер также относится к весьма легким и обрабатывает события скачивания файлов, загруженных пользователями на сервер.

```
C:\OpenServer\domains\laravel>php artisan make:controller DownloadController
Controller created successfully.
```

Рисунок 56 – Создание контроллера DownloadController

DownloadController, как и предшественники, состоит из одной функции index, исполняющей чтение загруженного пользователем на сервер файла и его передачу в браузер пользователя. Данный метод передачи файлов используется вместо предоставления прямой ссылки в интересах безопасности, поскольку в полной мере исключает удаленное исполнение загруженных на сервер файлов.

- Функция index обеспечивает безопасное скачивание с сервера загруженного пользователем файла.



```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\User;
use App\Models\Post;
use App\Models\Upload;
use DB;

class DownloadController extends Controller
{
    # Коробочная функция, обеспечивающая работу авторизации.
    public function __construct()
    {
        $this->middleware( middleware: 'auth');
    }

    # Функция index обеспечивает безопасное скачивание с сервера загруженного пользователем файла.
    public function index($fileid, Upload $upload)
    {
        # Получаем данные о запрошенном файле.
        $getFile = $upload->getFile($fileid);
        # Строим путь к файлу в месте его хранения на нашем сервере.
        $filetoget = '/var/www/laravel/uploads/'.$getFile->filename.'.'.$getFile->filetype;
        /*
         Передаем в браузер пользователя заголовки, которые позволяют скачать файл, а не просмотреть его непосредственно
         в окне браузера в виде исходного кода.
        */
        header( string: 'Content-Type: application/octet-stream');
        header( string: "Content-Transfer-Encoding: Binary");
        header( string: "Accept-Ranges: bytes");
        header( string: "Content-Length: ".filesize($filetoget));
        header( string: 'Content-Disposition: attachment; filename="'.str_replace(array('\\', '/', '"', '.'),
                replace: '', $getFile->title).'.'.$getFile->filetype.'"');
        /*
         Читаем файл и передаем его в браузер пользователя. Таким образом обеспечивается невозможность удаленного исполнения
         пользовательских файлов на сервере, что было бы достаточно серьезной дырой в безопасности.
        */
        readfile($filetoget);
        # Когда файл будет передан, просто завершаем соединение. Вывод для данного случая не требуется.
        exit();
    }
```

Рисунок 57 – Контроллер DownloadController

Последний контроллер, VerifyController, обеспечивает работу механизма верификации учетных записей пользователей, созданных в статусе преподавателей и старост. В связи с тем, что данный функционал подразумевает множество действий, набор встроенных функций контроллера состоит уже из большего количества: index, accept и decline.

```
C:\OpenServer\domains\laravel>php artisan make:controller VerifyController
Controller created successfully.
```

Рисунок 58 – Создание контроллера VerifyController

- Функция index обеспечивает вывод страницы со списком пользователей, ожидающих верификации регистраций.

- Функция accept обеспечивает одобрение заявки на регистрацию учетной записи преподавателя со стороны администратора ресурса.
- Функция decline обеспечивает отклонение запроса на регистрацию учетной записи преподавателем со стороны администратора ресурса.

С полным кодом контроллера можно ознакомиться на рисунке 58.

```
class VerifyController extends Controller{
    # Коробочная функция, обеспечивающая работу системы авторизации.
    public function __construct(){
        $this->middleware( middleware: 'auth');
    }

    # Функция index обеспечивает вывод страницы со списком пользователей, ожидающих верификации регистраций.
    public function index(Students $students, $pagination=1){
        # На странице выводится не более 25 пользователей.
        $itemsPerPage = 25;
        # Получаем список полльзователей из очереди на верификацию.
        $unverified = $students->getUnverified($pagination, $itemsPerPage);
        # Считаем количество пользователей, отданных базой данных по этому запросу.
        $unverifiedQ = $unverified->count();
        # Получаем суммарную длину очереди пользователей на верификацию.
        $unverifiedSum = $students->getUnverifiedSum();
        # Считаем количество страниц, на которое может быть разбит вывод.
        $pagesInSection = ceil( value: $unverifiedSum/$itemsPerPage);
        return view( view: 'verify')
            ->with('unverified', $unverified)
            ->with('unverifiedQ', $unverifiedQ)
            ->with('pagination', $pagination)
            ->with('itemsPerPage', $itemsPerPage)
            ->with('pagesInSection', $pagesInSection)
            ->with('unverifiedSum', $unverifiedSum);
    }

    # Функция accept обеспечивает одобрение заявки на регистрацию учетной записи преподавателя со
    # стороны администратора ресурса.
    public function accept(Students $students, $user_id){
        if(Auth::user()->is_admin == 1){
            $students->verifyUser($user_id);
        }
        return redirect()->back();
    }

    public function decline(Students $students, $user_id){
        if(Auth::user()->is_admin == 1){
            $students->declineUser($user_id);
        }
        return redirect()->back();
    }
}
```

Рисунок 59 – Контроллер VerifyController

Помимо описанных функций, каждый контроллер содержит также функцию __construct() (рис. 8.10), добавляемую механизмом авторизации Laravel для обеспечения корректной работы системы авторизации и распознания авторизованных пользователей в рамках данного контроллера.

Рисунок 60 – Функция __construct()

8.2. Контрольные вопросы

1. Что такое контроллер?
2. Где хранятся контроллеры?
3. Как создать новый контроллер?
4. Как обеспечивается маршрутизация контроллеров?
5. В чем отличие контролера от модели?

9. РАЗРАБОТКА МАРШРУТОВ

9.1. Основные понятия

Для выбора логики формирования ответов по URL запроса приложение использует маршрутизатор (роутер), выбирающий контроллер по правилам, описываемым маршрутами (роутами). Контроллер, в свою очередь, реализует логику приложения, которая необходима для обработки запроса по полученному URL.

Маршрутизация позволяет использовать URL-адреса, не сопоставляемые с определенными файлами на веб-узле. Поскольку URL-адрес не сопоставляется с файлом, можно использовать в веб-приложении URL-адреса, описывающие действия пользователя, вследствие чего они более понятны пользователям.

Посмотрите на представленный ниже запрос:

http://domain.com/my/page

В данном примере используется HTTP протокол для получения доступа к некоторому приложению, расположенному по адресу domain.com. Часть URL содержащая «my/page» используется для маршрутизации веб-запроса к соответствующей ему логике.

Список маршрутов (роутов) вашего приложения находиться в файле routes/web.php. Изначально данный файл будет содержать в себе маршрут для приветственного представления Laravel (рисунок 61).

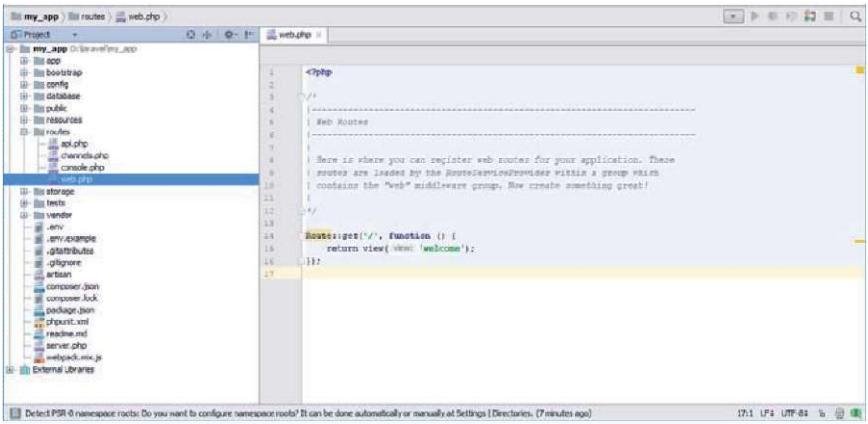


Рисунок 61 – Исходный код файла web.php

Создадим маршрут для указанного выше примера URL-адреса. Для этого необходимо дописать в файл web.php следующий код:

```
Route::get('my/page', function (){
    return 'Hello world!';
});
```

Теперь введем в адресную строку браузера http://domain.com/my/page, заменив domain.com адресом вашего приложения (как правило, это localhost или 127.0.0.1). Если все было сделано правильно, в вашем браузере появится строка «Hello world!» (рисунок 62).

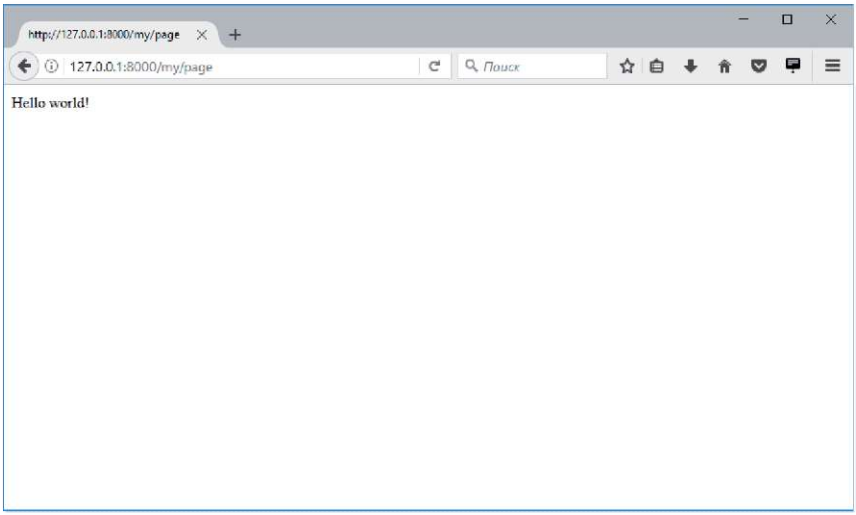


Рисунок 62 – Результат обращения к /my/page

Объявление маршрута всегда происходит с помощью класса Route. Он отвечает за перехват запросов, сделанных с помощью различных HTTP-команд для конкретного URL-адреса.

Большинство запросов, обрабатываемых веб-браузером, являются глаголами. Зачастую, этим глаголом будет GET, который используется для запроса веб-страницы. Запрос GET отправляется

каждый раз, когда вы вводите новый веб-адрес в свой веб-браузер. Однако, это не единственный запрос.

Существует также POST, который используется для запроса и отправки определенных данных вместе с ним. Как правило, это результат отправки формы, когда данные должны быть отправлены на веб-сервер без отображения в URL-адресе.

Список других методов, доступных классу маршрутизации (Route):

– Route::get();
– Route::post();
– Route::put();
– Route::delete();
– Route::any();

Все эти методы принимают одинаковые параметры, поэтому используемый метод зависит от ситуации. Основными видами запросов являются GET и POST.

Метод Route::any() используется для соответствия любому HTTP запросу.

Вернемся к нашему примеру маршрута:

```
Route::get('my/page', function (){
   return 'Hello world!';
});
```

Первым параметром, передаваемым в метод, является URI ('my/page'), это – универсальный идентификатор ресурса (Universal Resource Identifier). Следующий параметр отвечает за обеспечение логики приложения для обработки запроса. Здесь мы используем Closure, который также известен как анонимная функция. Замыкания – это просто функции без имени, которая может быть присвоена как переменным, так и другим простым типам.

Также эта функция может быть записана иначе:

```
$logic = function () {
   return 'Hello world';
};
Route::get('my/page', $logic);
```

В данном случае, мы присвоили переменной logic функцию замыкание, а затем передаем ее методу Route::get(). Эта функция выполнится только тогда, когда будет совершен GET запрос с соответствующем ему URI (my/page).

Параметры маршрутов могут использоваться для вставки заполнителей в определение маршрута. Это позволит создать шаблон, в котором сегменты URI могут быть собраны и переданы в логический обработчик приложения.

```
// Маршруты для раздела книг
Route::get('/books', function () {
    return 'Раздел книг.';
});
Route::get('/books/{genre}', function ($genre) {
    return 'Книги из категории {$genre}';
});
```

В этом примере, мы исключили необходимость во всех маршрутах, являющихся жанрами книг, включив в него местозаполнитель маршрута. Заполнитель {genre} передаст все что находится после /books/ в параметры функции замыкания, и тем самым позволит использовать эту информацию в нашей логической части.

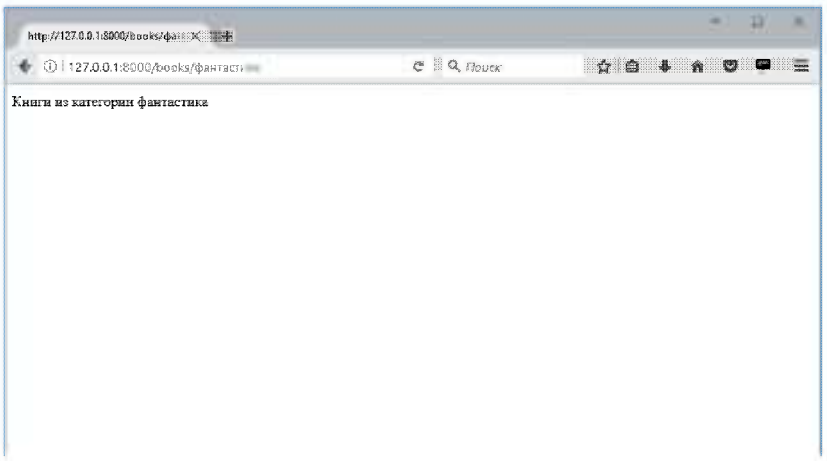


Рисунок 63 – Отображение параметра, полученного из URI

Например, если вы перейдете по следующему адресу:
http://127.0.0.1:8000/books/фантастика
То увидите, как данный раздел отобразится на странице сайта (рисунок 63).

Мы также можем удалить индексный маршрут для книг (/books), используя необязательный параметр. Для этого необходимо добавить вопросительный знак (?) в конце его имени. Например:

```
// Маршрут для раздела книг
Route::get('/books/{genre?}', function ($genre = null) {
    if ($genre == null) return 'Раздел книг.';
    return 'Книги из категории'.$genre;
});
```

Практическая часть.

При обращении через веб-браузер, в адресной строке отображается определенный URL, по которому сервер определяет, какой именно ресурс запрошен и что следует выдать в ответ на данный запрос. В Laravel 5 обработка этих адресов происходит посредством файла routes/web.php, который описывает существующие на сайте страницы и методы обращения к ним.

Поскольку данный файл входит в минимальный набор Laravel, создавать его отдельно нет никакой необходимости, вместо этого можно сразу приступать к его редактированию с целью внесения требуемых по проекту путей к страницам сайта.

Laravel 5 поддерживает множество различных методов HTTP, среди которых, кроме привычных GET и POST имеются также PUT, DELETE, OPTIONS и PATCH. В нашем проекте будет использоваться только два базовых метода – GET и POST.

По сути, запросы через метод GET, как правило, -- это запросы отображения, передающие те или иные параметры, модифицирующие вывод на странице, быть то порядковый номер страницы при постраничном разбиении, номер конкретной сущности, которую следует вывести на странице и т.д. В то же время запросы через метод POST – это, как правило, передача скрипту данных

форм, отправленных браузером пользователя по адресу action, указанному в корневом теге формы.

Рассмотрим на примере типовое вхождение файла routes для GET-запроса:

Route::get('/','HomeController@index')->name('home');

Данное правило указывает, что при обращении к корню сайта (или же просто по доменному имени или IP-адресу, что для веб-сервера равнозначно обращению «/») запрос пользователя будет обрабатываться функцией index контроллера HomeController. Кроме того, обратиться к этому пути можно откуда угодно далее по коду, поскольку для него задана именованная ссылка. Таким образом, к данному маршруту можно будет обратиться по имени home в представлении, и использовать адрес вместо указания вручную.

Теперь рассмотрим более сложный вариант, когда в запросе имеются параметры, которые нам необходимо получить и передать в контроллер:

Route::get('/my/{doctype}s/page-{pagination}','MyController@index');

В данном случае, {pagination} преобразуется в переменную $pagination, при помощи которой контроллеру будет передаваться номер запрошенной страницы при постраничном разбиении отображения контента.

Кроме всего прочего, Laravel позволяет явно указывать, какие данные следует пропускать в переменную, а какие стоит фильтровать при получении:

Route::get('/my/{doctype}s/page-{pagination}','MyController@index')->where(pagination, '[0-9]+');

Таким образом, в переменную $pagination будут пропущены только числовые значения, что будет соответствовать изначальному предназначению данного параметра.

Аналогично ситуация обстоит и с обработкой запросов по методу POST:

Route::post('/profile/set', 'MainController@setGroup')->name('post');

В данном примере, POST-запрос, пришедший на URL /profile/set отдается на обработку в функцию setGroup контроллера MainController, а кроме этого пути задается имя post.

Полное содержание файла routes/web.php можно увидеть на рисунке.

```
use App\Models\Post;


Auth::routes();

Route::get('/','HomeController@index')->name('home');
Route::get('/page-{pagination}','HomeController@index');
Route::get('/my','MyController@index')->name('my');
Route::get('/my/page-{pagination}','MyController@index');
Route::get('/my/{doctype}s','MyController@index');
Route::get('/my/{doctype}s/page-{pagination}','MyController@index');
Route::get('/download/{fileid}', 'DownloadController@index');
Route::get('/profile', 'MainController@index')->name('profile');
Route::get('/remove/{identifier}', 'MainController@remove');
Route::get('/users/verify', 'VerifyController@index');
Route::get('/users/verify/page-{pagination}', 'VerifyController@index');
Route::get('/users/verify/{user_id}/accept', 'VerifyController@accept');
Route::get('/users/verify/{user_id}/decline', 'VerifyController@decline');
Route::post('/profile/set', 'MainController@setGroup')->name('post');
Route::post('profile', 'MainController@update_avatar');
Route::post('/loadDocument', 'MainController@loadDocument');
Route::post('/setTeacherGroups', 'MainController@setTeacherGroups');
```

Рисунок 64 – Список маршрутов проекта

9.2. Контрольные вопросы

1. Что такое маршрут (route)?
2. Перечислите список методов доступных классу Route.
3. Что такое URI?
4. Что такое анонимная функция (замыкание)?
5. В каком файле хранятся маршруты приложения?
6. Какие параметры передаются в методы класса Route?

10. РАЗРАБОТКА ПРЕДСТАВЛЕНИЙ

10.1. Основные понятия

Представления – это визуальная часть вашего приложения. В схеме Model-View-Controller они составляют часть View.

Представления являются шаблонами, содержащими в себе HTML разметку и имеющие формат .blade.php. Это означает, что в них также может выполняться PHP код.

В фреймворке Laravel представления находятся в директории resources/views. Для примера создадим шаблон простого представления (рисунок 65), содержащего следующий HTML код:

```
<!DOCTYPE HTML>
<html>
<head>
   <meta charset="UTF-8">
   <title>Представления</title>
</head>
<body>
<p>Пример представления</p>
</body>
</html>
```

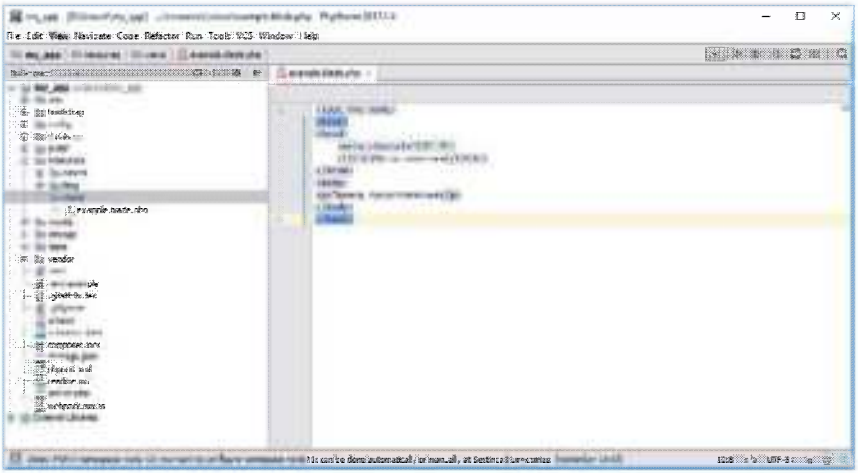


Рисунок 65 – Пример шаблона представления

Теперь, нам необходимо отобразить данное представление в браузере. Для этого мы должны вернуться к маршрутам и добавить туда несколько строчек:

```
Route::get('/', function () {
    return view('example');
});
```

Перейдя по адресу вашего приложения, вы сможете увидеть шаблон вашего представления (рисунок 66).

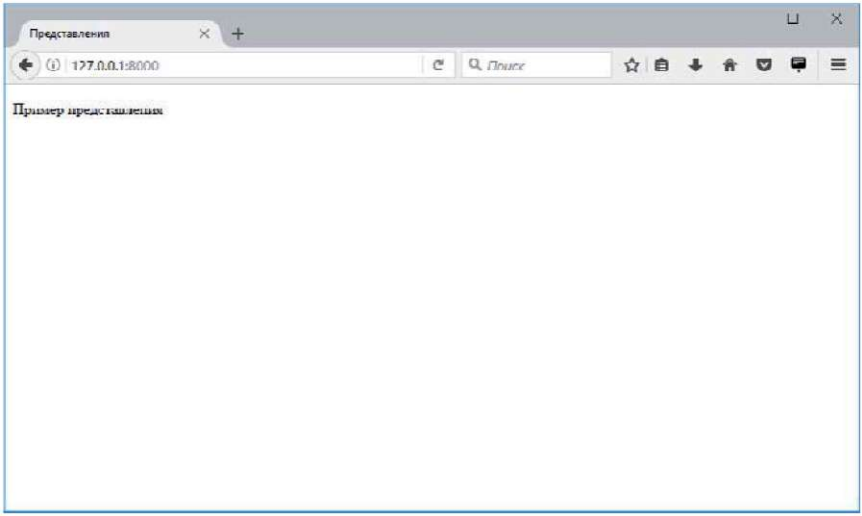


Рисунок 66 – Отображение шаблона представления

Используя метод view(), вы создаете новый экземпляр объекта View. Первым параметром, передаваемым в метод, является шаблон представления, который мы хотим использовать. Вы могли заметить, что мы использовали не полный путь resources/views/example.blade.php, а только имя файла. Такое происходит благодаря тому, что Laravel упрощает разработку и автоматически находит файл в директории представлений, но если вы размещаете свой шаблон в одной из подпапок директории, то ее необходимо указать [15].

Помимо отображения шаблонов, у нас есть возможность передачи в него данных из функции замыкания.

```
Route::get('/{name}', function ($name){
  return view('name', ['name' => $name]);
});
```

В приведенном выше маршруте мы берем параметр $name и передаем его в наше представление. Теперь нам необходимо организовать вывод параметра в шаблоне. Для этого необходимо вставить переменную в месте вывода.

```
<!DOCTYPE HTML>
<html>
<head>
   <meta charset="UTF-8">
   <title>Name View</title>
</head>
<body>
<p>Привет, {{ $name }}</p>
</body>
</html>
```

Теперь вы можете отрыть ваше приложение в браузере, добавив после него имя. Если все было сделано правильно, вы увидите страницу содержащую приветствие (рисунок 67).

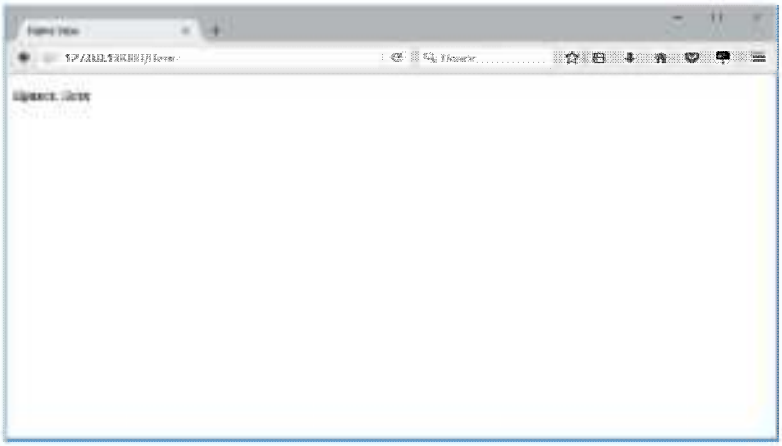


Рисунок 67 – Страница с приветствием

10.2. Контрольные вопросы

1. Что такое представление (view)?
2. Что такое шаблон?
3. Какой формат используют представления?
4. В каком каталоге располагаются представления?
5. Какие параметры можно передавать в метод представления?

11. ПОДКЛЮЧЕНИЕ ВНЕШНИХ ШАБЛОННЫХ ФОРМ

11.1. Работа с PHPWORD

Установку PHPWord выполним автоматическим способом аналогично установке Laravel, используя инструмент Composer.

После того как Composer, завершил скачивание и установку библиотеки, необходимо подключить файл autoload.php, который расположен в папке vendor, к Вашему проекту.

```
require 'vendor/autoload.php';
```

Далее, создаем объект главного класса библиотеки.

```
$phpWord = new \PhpOffice\PhpWord\PhpWord();
```

Обратите внимание, что PHPWord, в своей структуре, использует пространства имен, поэтому для доступа к классу, необходимо использовать полное квалификационное имя, если мы работаем в глобальном пространстве имен. На этом установка библиотеки завершена.

Создание документа MS Word

Теперь, мы можем сформировать свой первый документ MS Word средствами языка PHP. Для этого, первым делом определим шрифт, используя метод setDefaultFontName(имя шрифта), который будет использоваться, по умолчанию, для отображения текстовых данных.

```
$phpWord->setDefaultFontName('Times New Roman');
```

Затем зададим, размер шрифта, при помощи метода setDefaultFontSize(размер шрифта).

```
$phpWord->setDefaultFontSize(14);
```

Перед добавлением текстовых данных, необходимо определить параметры всего документа в целом. Для получения объекта параметров документа, используем метод getDocInfo().

```
$properties = $phpWord->getDocInfo();
```

Используя полученный объект, зададим основные параметры документа.

```
$properties->setCreator('Name');
$properties->setCompany('Company');
$properties->setTitle('Title');
$properties->setDescription('Description');
$properties->setCategory('My category');
```

```
$properties->setLastModifiedBy('My name');
$properties->setCreated(mktime(0, 0, 0, 3, 12, 2015));
$properties->setModified(mktime(0, 0, 0, 3, 14, 2015));
$properties->setSubject('My subject');
$properties->setKeywords('my, key, word');
```

При этом использовались следующие методы (каждый метод устанавливает определенный глобальный параметр документа):

setCreator() – автор документа;

setCompany() – организация автора документа;

setTitle() – заголовок документа;

setDescription() – краткое описание документа;

setCategory() – категория документа;

setLastModifiedBy() – автор последнего редактирования документа;

setCreated() – дата создания документа;

setModified() – дата редактирования документа;

setSubject() – тема документа;

setKeywords() – ключевые слова документа.

Собственно, теперь мы можем добавить необходимые текстовые данные в будущий документ MS Word, но перед этим мы должны определиться с понятием раздела, которое используется библиотекой PHPWord, для работы с документом.

Итак, раздел или секция – это специальная область прямоугольной формы, внутри которой размещаются элементы документа, такие как текст, изображения, списки, таблицы и т.д. А значит, перед добавлением информации в будущий документ, необходимо создать раздел, что мы собственно и выполним, используя метод addSection(массив стилей).

```
$sectionStyle = array(
'orientation' => 'landscape',
'marginTop' =>
\PhpOffice\PhpWord\Shared\Converter::pixelToTwip(10),
'marginLeft' => 600,
'marginRight' => 600,
'colsNum' => 1,
'pageNumberingStart' => 1,
'borderBottomSize'=>100,
```

```
'borderBottomColor'=>'C0C0C0'
);
$section = $phpWord->addSection($sectionStyle);
```

Данный метод, в качестве результата работы возвращает объект созданного раздела, который мы сохраним в переменную. При вызове метода, в качестве первого параметра, можно передать массив настроек, которые будут использоваться для создания раздела (каждая настройка представляет собой отдельную ячейку массива). В примере Выше, мы использовали следующие настройки:

orientation — расположение раздела, в виде альбомного листа (значение по умолчанию portrait);

marginTop – верхний отступ;

marginLeft – отступ от левого края;

marginRight – отступ от правого края;

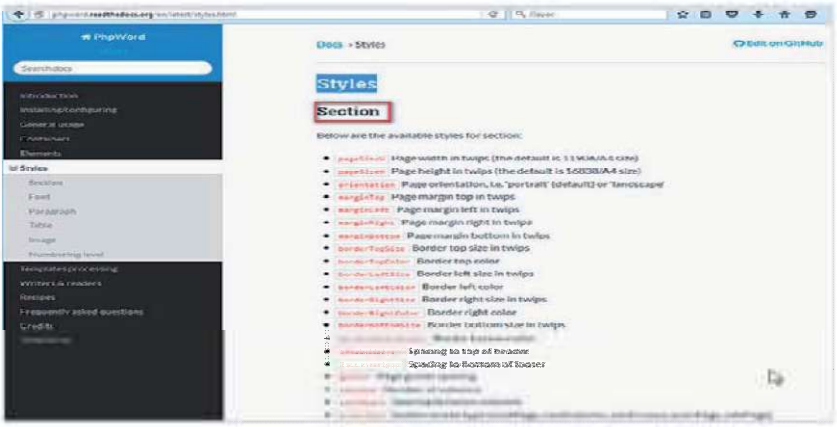
colsNum – количество колонок, в которых будут отображаться данные;

pageNumberingStart – страница, с которой будет начата нумерация страниц;

borderBottomSize – размер нижней рамки;

borderBottomColor – цвет нижней рамки.

Полный список настроек приведен в разделе “Styles”, официальной документации PHPWord.



При, этом у Вас, скорее всего, возник вопрос, в каких единицах измерения проставляются значения размеров и отступов? В качестве единиц измерений используются типографические твипы.

Твип (англ. twip) — типографская единица измерения, равная одной двадцатой пункта, или 1/1440 дюйма, или 1/567 сантиметра (приближённо).

Конечно, разработчикам не совсем удобно использовать данную единицу измерения, для определения размеров, поэтому библиотека PHPWord, содержит в своем составе, специальный класс конвертер, основных известных единиц в твипы. К примеру, для конвертации "пискселей в твипы", необходимо использовать следующий метод pixelToTwip() , который переведет значение в пикселях, передаваемое в качестве первого параметра в твипы.

\PhpOffice\PhpWord\Shared\Converter::pixelToTwip(10);

Для добавления текста, в будущий документ, необходимо использовать метод addText($text, [$fontStyle], [$paragraphStyle]). В качестве параметров, при вызове данного метода, необходимо передать следующее:

$text – текст, который необходимо отобразить на странице документа. При этом текст не должен содержать тегов HTML, поэтому, как правило, его обрабатывают функцией htmlspecialchars().

$fontStyle – массив с настройками шрифта, который будет использоваться для отображения текста. Полный список доступных настроек, Вы найдете на странице "Styles" в разделе "Font", официальной документации.

$paragraphStyle – массив с настройками параграфа, или абзаца, в котором будет отображен текст. Полный список доступных настроек, Вы найдете на странице "Styles" в разделе " Paragraph", официальной документации.

Теперь, используем рассмотренный Выше метод и добавим текст, в будущий документ.

$text = "PHPWord is a library written in pure PHP that provides a set of classes to write to and read from different document file formats.";

$fontStyle = array('name'=>'Arial', 'size'=>36, 'color'=>'075776', 'bold'=>TRUE, 'italic'=>TRUE);

$parStyle = array('align'=>'right','spaceBefore'=>10);

```
$section->addText(htmlspecialchars($text), $fontStyle,$parStyle);
```

Настройки шрифта, использованные в примере:

name – имя шрифта;

size – размер шрифта;

color – цвет шрифта;

bold – если, true, будет использован жирный шрифт;

italic — если, true, будет использован курсив.

Настройки параграфа из примера:

align – выравнивание текста в параграфе, в нашем случае по правому краю;

spaceBefore – расстояние до параграфа.

Теперь давайте, непосредственно, создадим документ MS Word.

```
$objWriter =
\PhpOffice\PhpWord\IOFactory::createWriter($phpWord,'Word2007');
$objWriter->save('doc.docx');
```

Для создания документа, необходимо создать объект специального класса Word2007, используя статический метод createWriter(), класса IOFactory. Класс IOFactory – реализует шаблон проектирования Factory, и необходим для создания объектов других классов, имена которых мы передаем в качестве, второго параметра при вызове метода createWriter(). Далее вызывая метод save() и передавая в качестве первого параметра, имя будущего файла, мы формируем документ MS Word.

Как Вы видите, документ успешно создан. Если, не нужно создавать файл, а сформированный документ, необходимо отдать пользователю на скачивание, то при вызове метода save(), в качестве первого параметра, необходимо передать строку «php://output». При этом, так же, необходимо указать определенный набор заголовков.

```
header("Content-Description: File Transfer");
header('Content-Disposition: attachment; filename="first.docx"');
header('Content-Type:application/vnd.openxmlformats-
officedocument.wordprocessingml.document');
header('Content-Transfer-Encoding: binary');
header('Cache-Control: must-revalidate, post-check=0, pre-
check=0');
```

```
header('Expires: 0');
$objWriter =
\PhpOffice\PhpWord\IOFactory::createWriter($phpWord, 'Word2007');
$objWriter->save("php://output");
```

Добавление списков

Для формирования списков, в будущем документе, необходимо использовать метод addListItem($text, [$depth], [$fontStyle], [$listStyle], [$paragraphStyle]), который за каждый свой вызов, формирует элемент списка. Параметры, которые необходимо передать при вызове метода:

$text – текст, элемента списка;

$depth – глубина вложенности. Если создается одноуровневый список, то данный параметр равен 0.

$fontStyle – массив настроек шрифта, по аналогии с добавлением простого текста.

$listStyle – массив настроек списка.

$paragraphStyle – массив настроек параграфа, по аналогии с добавлением текста.

Массив настроек списка $listStyle, поддерживает настройку – listType, то есть, тип списка, к примеру, нумерованный или же нет. В качестве значений, доступны специальные константы класса \PhpOffice\PhpWord\Style\ListItem:

TYPE_SQUARE_FILLED FILLED – не нумерованный список. В виде маркеров используются квадраты.

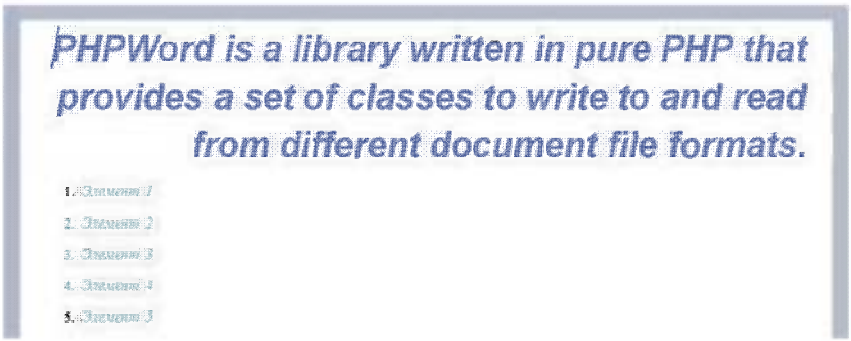
TYPE_BULLET_FILLED – не нумерованный список (значение по умолчанию). В виде маркеров используются точки.

TYPE_BULLET_EMPTY FILLED – не нумерованный список. В виде маркеров используются не закрашенные окружности.

TYPE_NUMBER – нумерованный список.

TYPE_NUMBER_NESTED – многоуровневый нумерованный список.

TYPE_ALPHANUM – нумерованный список, с использованием букв, в качестве маркеров.

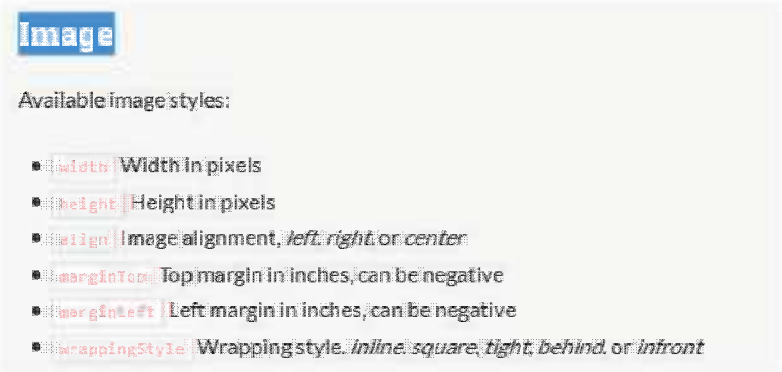


Таким образом, следующий код, добавит одноуровневый нумерованный список в документ.

```
$fontStyle = array('name' => 'Times New Roman', 'size' => 16,'color' => '075776','italic'=>true);
$listStyle = array('listType'=>\PhpOffice\PhpWord\Style\ListItem::TYPE_BULLET_EMPTY);
$section->addListItem('Элемент 1',0,$fontStyle,$listStyle);
$section->addListItem('Элемент 2',0,$fontStyle,$listStyle);
$section->addListItem('Элемент 3',0,$fontStyle,$listStyle);
$section->addListItem('Элемент 4',0,$fontStyle,$listStyle);
$section->addListItem('Элемент 5',0,$fontStyle,$listStyle);
```

Добавление изображений

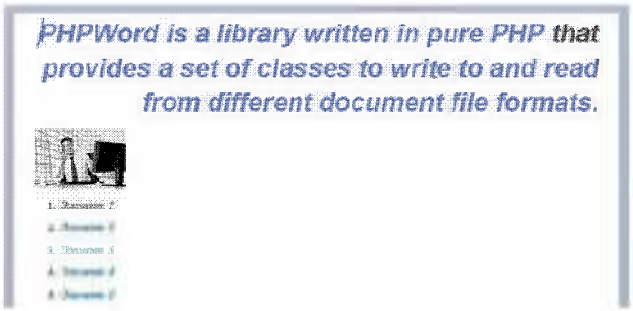
Для добавления изображений, необходимо использовать метод addImage($path,[$imgStyle]). При вызове данного метода, в качестве первого параметра, передается путь к изображению, которое необходимо добавить в документ. В качестве второго, необязательного параметра, можно передать массив с настройками отображения изображения. Полный список настроек изображения, Вы найдете на странице “Styles”, в разделе “ Image”.



Соответственно, давайте добавим изображение в создаваемый документ.

```
$section->addImage('picture.jpg', array(
'width' => 100,
'height' => 100,
));
```

При этом, в качестве настроек, мы определили ширину и высоту добавляемого изображения.



Как Вы видите, библиотека PHPWord, обладает огромнейшим функционалом и позволяет формировать документы MS Word различной сложности.

Использование шаблонов

Библиотека PHPWord поддерживает еще один интересный метод создания документов Word - использование шаблонов. В качестве метки в PHPWord используется комбинация **${NAME}**, где

NAME - имя метки. Пример генерирования с использованием меток:

```
$template->setValue('Name', 'Иванов'); //Производим замену метки на значение
$template->setValue('Surname', 'Иван'); //И еще одну метку
$template->save('document.docx'); //Сохраняем результат в файл
$template = $word->loadTemplate('template.docx'); //Загружаем шаблон
```

Сохранение файла

Для сохранения файла на жесткий диск или вывода его для скачивания пользователю используется метод **save** класса PHPWord_Writer_Word2007. В качестве единственного аргумента метода указывается строка **'php://output'**, если требуется вывести документ для скачивания пользователем или имя файла, если документ нужно сохранить на жесткий диск. При выводе файла для скачивания не забудьте отправить браузеру пользователя соответствующие заголовки:

```
header('Content-Type: application/vnd.openxmlformats-officedocument.wordprocessingml.document');
header('Content-Disposition: attachment;filename="document.docx"');
header('Cache-Control: max-age=0');
$writer = PHPWord_IOFactory::createWriter($word, 'Word2007');
$writer->save('php://output');
```

Сохранение файла на жесткий диск:

```
$writer = PHPWord_IOFactory::createWriter($word, 'Word2007');
$writer->save('filename.docx');
```

11.2. Индивидуальное задание

Подключить внешнюю шаблонную форму MS Office. Организовать экспорт данных в шаблон MS Office.

Вариант	Данные
0	Карточка пользователя
1	Список студентов кафедры X
2	Список старост факультета X
3	Список преподавателей кафедры X
4	Карточка старосты кафедры Y
5	Карточка студента кафедры X
6	Список всех кафедр на факультете X
7	Список всех факультетов
8	Список групп, которые курирует преподаватель Y
9	Список студентов кафедры Y
10	Список преподавателей факультета X

11.3. Пример выполнения задания

Первым пунктом добавим кнопку для создания и экспорта файла. Для этого добавим в файл resources/views/profile.blade.php следующий код:

<input type="button" class="pull-right btn btn-sm btn-primary" value="Экспорт профиля" style="margin-left:5px;">.

В файле routes/web.php пропишем маршрут:

Route::get('/my/export', 'MyController@export');

Добавим в файл app/Http/Controllers/MyController.php функцию, которая будет подставлять в созданный нами шаблон MS Office данные из базы данные, и сохранять его:

```
private function createFile($user) {
$templateProcessor = new TemplateProcessor(__DIR__ . '/../../../template.docx');
 $templateProcessor->setValue('${fio}', $user->name);
$templateProcessor->setValue('${email}', $user->email);
```

```
$templateProcessor->setValue('${date}', $user->created_at);
$templateProcessor->setValue('${status}', $this->getStatus($user->status));
$templateProcessor->saveAs($user->id . '.docx');
return $user->id . '.docx';
}
```

Далее опишем функцию, которая реализует отправку файла на скачивание:

```
public function export() {
$user = Auth::user();
$filename = $this->createFile($user);
return response()->download($filename)->deleteFileAfterSend(true);
}
```

Далее нужно создать шаблон MS Office, который мы положим в папку нашего проекта. В нашем случае это файл template.docx, которые выглядит следующим образом:

Карточка пользователя

Фотография пользователя:

${photo}

ФИО: ${fio}

E-mail пользователя: ${email}

Дата регистрации: ${date}

Статус пользователя: ${status}

11.4. Контрольные вопросы

1. В каких единицах измерения проставляются значения размеров и отступов?
2. При помощи какого метода задается размер шрифта?
3. Перечислите основные параметры документа.
4. Перечислите основные параметры для настройки шрифта.
5. Расскажите процесс добавления списков.
6. Расскажите процесс добавления изображений.

ЗАКЛЮЧЕНИЕ

В рамках учебного пособия рассмотрены особенности функционирования и настройки Web-серверов, подробно описаны принципы работы с формами, фреймами, файлами, СУБД, регулярными выражениями, графикой и последовательно описан процесс создания клиент-серверного приложения на базе современного php-фреймворка. Особенное внимание уделено рассмотрению потенциальных уязвимостей при работе с БД, показан вариант обеспечения информационной безопасности с подключением пакетов расширений и их актуализацией на уровне проекта.

В первом разделе описаны основы разработки представлений и простейшие вычисления, для этого описание и принцип работы с HTML, CSS, JS, JQuery, JSON. Во втором разделе отражены принципы адаптивной верстки с применением современного фронтенд-фреймворка Twitter Bootstrap, далее рассмотрены директивы и принцип построения защищенных приложений на AngularJS. Четвертая глава посвящена рассмотрению интеграции внешних элементов и виджетов.

Начиная с пятого раздела и до конца пособия последовательно рассматриваются этапы создания защищенного приложения на базе php-фреймворка Laravel «с нуля». Подробно рассмотрены вопросы безопасности баз данных, их создание в режиме миграций, что особенно важно при создании и развитии систем несколькими разработчиками. Для приобретения практического навыка, разделы сопровождаются фрагментами исходного кода, набором индивидуальных заданий и контрольными вопросами для самоконтроля.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. htmlbook.ru | Для тех, кто делает сайты [Электронный ресурс] // http://htmlbook.ru/ (Дата обращения: 15.09.2018).
2. HTML5BOOK.RU - HTML, CSS, JavaScript и jQuery [Электронный ресурс] // https://html5book.ru/ (Дата обращения: 29.09.2018).
3. Bootstrap · The most popular HTML, CSS, and JS library in the world. [Электронный ресурс] // https://getbootstrap.com// (Дата обращения: 23.08.2018).
4. Bootstrap по-русски [Электронный ресурс] // http://mybootstrap.ru/ (Дата обращения: 15.12.2017).
5. AngularJS — Superheroic JavaScript MVW Framework [Электронный ресурс] // https://angularjs.org/ (Дата обращения: 12.08.2017).
6. Информационная безопасность [Текст] : учеб. пособие: / Спеваков А.Г., Таныгин, М.О., Панищев В.С.; Юго-Зап. гос.ун-т. Курск, 2017. 196 с.
7. Программно-аппаратные средства защиты информационных систем [Текст] : учеб. пособие: / Спеваков А.Г., Калуцкий И.В.; Юго-Зап. гос.ун-т. Курск, 2014. 182 с.
8. Романец, Ю. В. Защита информации в компьютерных системах [Текст] / Ю.В. Романец, П. А. Тимофеев, В. Ф. Шаныгин; под ред. В. Ф. Шаныгина. – 2–е изд., перераб. и доп. – М. : Радио и связь, 2001. – 376 с.: ил.
9. Статьи | Laravel по-русски [Электронный ресурс] // https://angularjs.org/ (Дата обращения: 15.09.2018).
10. Laravel - The PHP Framework For Web Artisans [Электронный ресурс] // https://laravel.com/ (Дата обращения: 15.09.2018).
11. Таныгин М.О. Обнаружение при программном управлении работой устройства команд, выданных посторонними программами / В сборнике: Оптико¬электронные приборы и устройства в системах распознавания образов, обработки изображений и символьной информации. Распознавание ¬ 2005. сборник материалов 7¬ й Международной конференции . Ответственный редактор В.С. Титов. 2005. – с. 202¬203.

12. Марухленко С.Л., Дегтярев С.В., Марухленко А.Л. Программная модель для автоматизации расчета показателей риска техногенных аварий // Информационно-измерительные и управляющие системы. – 2010. Т. 8. – № 11. С. 35-39.
13. Марухленко А.Л., Мирзаханов П.С., Марухленко С.Л. Мониторинг и имитационное моделирование процессов взаимодействия абонентов вычислительной сети, Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. 2012. № 2-3. С. 236-241.
14. Типикин, А.П., Таныгин, М.О. Методы аутентификации устройств защиты информации и управляющих программных средств. // Телекоммуникации. – 2005. – №9. – С.37 – 42. – Библиогр.: с. 42.
15. Марухленко А.Л., Мирзаханов П.С. Программный комплекс для моделирования процесса передачи и обработки сетевых потоков данных // Известия Юго-Западного государственного университета. 2013. № 2. С. 175.

Учебное издание

Марухленко Анатолий Леонидович
Марухленко Леонид Олегович
Ефремов Михаил Александрович

Разработка защищённых интерфейсов Web-приложений

Учебное пособие

Ответственный редактор *С. Краснова*
Верстальщик *Д. Ананьева*

Издательство «Директ-Медиа»
117342, Москва, ул. Обручева, 34/63, стр. 1
Тел/факс + 7 (495) 334–72–11
E-mail: manager@directmedia.ru
www.biblioclub.ru