МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт радиоэлектроники и информационной безопасности
Кафедра «Радиоэлектроника и телекоммуникации»

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

по дисциплине «Коммуникации в сети интернет и принципы построения сайтов»

для

студентов очной формы обучения по направлениям

общеуниверситетского пула

1 часть

«Создание интерфейса сайта»

Севастополь
2020

УДК 621.396
Учебно-методическое пособие Коммуникации в сети Интернет и принципы построения сайтов [Электронный ресурс] / СевГУ; сост. Д. И. Табакаев. — Севастополь : Изд-во СевГУ, 2020 — 67с.

Цель пособия: Целью учебно-методического пособия является оказание помощи студентам очной формы обучения при выполнении лабораторных работ по дисциплине «Коммуникации в сети Интернет и принципы построения сайтов».

Учебно-методическое пособие утверждено на заседании кафедры «Радиоэлектроника и телекоммуникации» от 23.03.20 г., протокол № 11.

Учебно-методическое пособие рассмотрено и рекомендовано к изданию на заседании методической комиссии Института радиоэлектроники и информационной безопасности от 24.04.20 протокол №9.

Рецензент д. т. н., профессор **И.Л. Афонин**
Редактор: к. т. н., доцент **В. Г. Слёзкин**

Ответственный за выпуск:
заведующий кафедрой «Радиоэлектроника и телекоммуникации»
д. т. н., профессор **И. Л. Афонин**

Издательский номер: №127/2020

Оглавление

ВВЕДЕНИЕ

В последние два десятилетия Интернет прочно вошел в нашу жизнь. В наше время он охватывает почти все сферы жизни: культура, искусство, бизнес. Наряду с обычными музеями, библиотеками, магазинами все большую популярность во всем мире завоевывают аналогичные Интернет-ресурсы. Даже бизнес становится виртуальным: создаются *Web*-представительства фирм, электронные магазины и торговые площадки, электронные банки, которые дают возможность осуществлять типичные для этих сфер операции с большей скоростью. В связи с этим профессия *Web*-дизайнера и *Web*-программиста стала востребованной и у нас в России. Интернет-технологии создания указанных *web*-ресурсов требует владения такими языками для написания серверных скриптов, как *Perl, PHP*, *ASP*, владения языком *XML*, а также умения создавать динамические страницы и сайты, используя скрипты *VBScript* и *JavaScript*, которые позволяют осуществлять вычисления, работу с датой и временем, изменение эле-ментов страницы по желанию пользователя.

На первом этапе необходимо научиться создавать простые *HTML*-страницы. Не стоит считать, что в этом нет необходимости. Простого *HTML* вполне хватит для двухстраничной коммерции, которая распространяется сейчас по всему миру. *HTML* позволяет создать «заготовки» страниц, а уже потом можно добавлять в них различные скрипты. Поэтому без изучения HTML не обойтись. *HTML – Hyper-Text MarkUp Language* – язык разметки гипертекста. Гипертекст – расширенный текст, который может содержать в себе текст, иллюстрации, различные внедренные объекты и ссылки на другие ресурсы или на другие *Web*-страницы.

При передаче информации через Интернет используются как раз *HTML*-страницы – файлы *.htm и *.html, поскольку они содержат в себе гипертекст, имеют размер меньше, чем текстовые или иные файлы и для их просмотра ну-жен только браузер.

HTML-документ представляет собой обычный текстовый документ с управляющими конструкциями языка *HTML* – тегами, которые и производят действия «форматирования» над текстовыми блоками и осуществляют вставку различных объектов в документ.

Создавать *HTML*-страницы можно вручную – путем самостоятельного на-писания тегов в текстовом редакторе или используя *Web*-редакторы с автоматизацией ввода нужных тегов – например, *WebEdit, Arachnophilia* или же используя программы, которые создают теги сами в ответ на действие пользователя на странице – по принципу What You See Is What You Get – например, *MS Word, MS Frontpage*. Но они задают абсолютное форматирование и часто дописывают много лишнего кода к странице. К тому же для создания более уникальных страниц ими не обойдешься, для

этого лучше писать теги *HTML* самому и использовать каскадные таблицы стилей.

Данное учебно-методическое пособие призвано помочь в освоении базовых знаний и получении практических навыков в области сразу нескольких технологий веб-программирования: языков разметки *HTML*, каскадных стилей *CSS*, фреймворка *Bootstrap* позволяющего быстро разрабатывать веб- интерфейсы. Представляемые три лабораторные работы выполняются студентами направлений общеуниверситетского полу в рамках рабочей программы по дисциплине «Коммуникации в сети Интернет и принципы построения сайтов». Все работы выполняются и тестируются в рамках локальной вычислительной сети с использованием учебного веб-сервера. Перед выполнением каждой работы обязательным является ознакомление с учебно-методическим пособием.

1. ЛАБОРАТОРНАЯ РАБОТА №1. ИСПОЛЬЗОВАНИЕ *HTML* ДЛЯ РАЗРАБОТКИ СТАТИЧЕСКОГО *WEB* – САЙТА.

1.1.Цель работы

Освоение основ языка разметки гипертекста *HTML*, дескрипторов форматирования текста и технологии создания гиперссылок.

1.2.Теоретические сведения

1.2.1.Основы *WWW*

World Wide Web – это средство представления информации в Интернет. Более точно *WWW* можно определить как интерактивную мультимедийную гипертекстовую среду, использующую язык разметки и поддерживающую множество протоколов Интернет[1].

Мультимедийная среда – возможность размещать в Сети не только и не столько текстовую информацию, но и графику, звук, видеоклипы.

Гипертекстовая среда – возможность использования в тексте ссылок на другие порции информации (текст, графику, звуковые и видеоклипы и т.д.), дополнительные источники. Многообразие таких ссылок и порождает Всемирную паутину *WWW*.

В *Web* эти ссылки называют «Унифицированным указателем ресурсов *URL*». Каждый документ или файл в Интернет имеет собственный *URL*, что позволяет легко ссылаться на него из других документов.

*Я*зык разметки *HyperText Markup Language (HTML*) является основой *World Wide Web*. С помощью конструкций языка создаются *Web* страницы (сайты), которые можно просматривать с помощью специальных программ - обозревателей, или как их еще называют - браузеров. Наиболее известными браузерами являются *Microsoft Internet Explorer, Mozilla Firefox, Opera, Google Chrome*.

Протоколы *Internet*. *WWW* использует собственный протокол связи между отдельными узлами: *HTTP (HyperText Transfer Protocol).* Любой протокол - это набор правил, которые используются компьютерами для обмена информацией. Кроме *HTTP* можно использовать и другие протоколы, получая доступ и другим приложениям *Internet*: *UseNet* - для доступа к группам новостей, *FTP* - для загрузки файлов; *POP3, SMTP* - электронной почте, *Skype* для телефонного общения и др.

WWW - в отличие от телевидения, интерактивная среда. Пользователи сами определяют, какой из узлов посещать, как долго изучать полученную информацию. Более того, на *Web* страницах часто можно ввести диалог с их авторами.

Web использует технологию "клиент - сервер". Это означает, что где – то в сети существует компьютер, на котором работает программное обеспечение *Web* - сервера, управляющее доступом к информации. Различные пользователи со всего мира являются клиентами сервера и получают от него информацию с помощью своих браузеров.

1.2.2. Создание *HTML* – файлов

Основным понятием языка *HTML* является понятие тег - инструкция обозревателю, как отображать текст.

Теговая модель описывает документ как совокупность контейнеров, каждый из которых начинается и заканчивается тегами. Общая структура контейнера:

```
<"имя тега" "список атрибутов">
содержание контейнера
</"имя тега">
```

Большинство тегов спарены т.е. за открывающим тегом следует соответствующий закрывающий тег, а между ними содержится текст

или другие теги. В таких случаях два тега и часть документа, отделенная ими, образуют блок, называемый *HTML* элементом. Некоторые теги являются элементами *HTML* сами по себе, и в рамках спецификации *HTML* для них соответствующий конечный тег необязателен, однако для спецификации *XHTML* закрывать тег обязательно для валидности *WEB* документа.

Независимо от того, как выглядит домашняя страница, и какую информацию необходимо отобразить, существует три тега, которые в соответствии со стандартами *HTML* и *WWW* должны присутствовать в каждой *Web*-странице:

- **<*html*>** Сообщает браузеру, что документ написан на языке *HTML*.
- **<*head*>** Отмечает вводную и заголовочную части *HTML*-документа.
- **<*body*>** Отмечает основной текст и информацию.

Эти теги необходимы *Web*-браузеру для определения различных частей *HTML*-документа, но они не оказывают прямого влияния на внешний вид Web-страницы. Они необходимы для того, чтобы последующие нововведения в *HTML* правильно интерпретировали *Web* - страницу, а также для того, чтобы она выглядела одинаково во всех *Web*-браузерах. Например, на *Web*-сервере запущена программа, которая просматривает все *HTML*-документы и составляет их список. Она просматривает только текст, расположенный внутри тегов <***head***>, поскольку здесь находится название документа. Таким образом, если в вашей домашней странице нет тегов <***head***> и <***/head***>, она не будет

включена в список. Так работает большинство средств поиска наиболее популярных *Web*-серверов. Они берут информацию из тегов **<*head*>** (и других).

Схематично разметку любой страницы можно представить следующим образом:

```
<!Спецификация DOCTYPE>
<html>
<head (Заголовок)>
[<Надпись заголовка>]
[<Стили>]
[<Скрипты>]
</head>
<body (Тело страницы)>
[<Разметка страницы>]
</body>
</html>
```

Спецификация документа должна быть указана, поскольку в противном случае браузер не будет иметь возможности корректно интерпретировать разметку. В различных браузерах по умолчанию отображение различно!

В соответствии с терминологией *SGML*, *html*-страница является документом, а элемент *html* – корневым элементом.

1.2.3. Основные элементы разметки

1.2.3.1. Элемент *html*

Элемент ***html*** является обязательным для любой страницы. Определяет границы ***html***-документа.

1.2.3.2. Элемент *head*

Не является обязательным. Представляет собой контейнер, в котором размещаются элемента заголовка документа.

Внутри элемента ***head*** могут быть определены следующие элементы:

Таблица 1.1 — допустимые элементы внутри ***head***.

Тэг	Описание
<*title*>	Определяет заглавие документа. Обычно браузеры отображают этот заголовок в заголовке окна.
<*base* />	Определяет базовый адрес или адрес по умолчанию для всех ссылок на странице. Ссылки на странице, содержащие относительный путь будут дополняться базовым адресом. При

Тэг	Описание
	перемещении такого документа все ссылки будут сохранены.
<***link*** />	Определяет связи между документом и внешними ресурсами. Например подключение стилей отображения.
<***meta*** />	Определяет метаданные о документе. Среди них могут быть как дополнительные ключевые слова, используемые для поиска, так и указание кодировки страницы.
<***script***>	Определяет скрипты, выполняемые на стороне браузера.
<***style***>	Определяет информацию о стилях отображения в документе. В отличии от <***link***> эти стили будут внедрены в документ.

Следует отметить, что для страниц на русском языке обязательно необходимо указывать кодировку текста. Примеры:

```
<meta http-equiv="Content-Type" content="text/html; char-
set=utf-8" />
<meta http-equiv="Content-Type" content="text/html; char-
set=windows-1251" />
<meta http-equiv="Content-Type" content="text/html; char-
set=koi8-r" />
```

В настоящее время целесообразно использовать только *UNICODE*-кодировки, например *utf-8*.

1.2.3.3. Элемент *body*

Представляет собой контейнер, в котором размещаются элементы, описывающие тело (или содержимое) документа.

Исторически этот элемент имел дополнительные атрибуты, позволявшие выбрать цвет фона, шрифта, фоновый рисунок, однако эти атрибуты устарели и их использование запрещено.

Элемент содержит ряд стандартных атрибутов, таких как класс, стиль, заголовок элемента, язык.

1.2.4. Элементы разметки текста

Рассмотрим еще несколько часто используемых элементов, которые применяются для разметки текста.

Таблица 1.2 — Некоторые элементы разметки текста

Тэг	Описание
<***p***>	Параграф. Позволяет указать область текста, относящуюся к одному параграфу. Параграфы при отображении обычно отделяются увеличенным отступом. В рамках параграфа игнорируются переводы строк в исходном тексте. Формат переноса может быть задан стилями, но не имеет отношения к исходной разметке *Пример:* <***p***>Некоторый текст.</***p***>

Тэг	Описание
	Будет выведено: Некоторый текст.
<*br* />	Перевод строки в любом месте текста, включая параграфы. Параграф при этом не разрывается. *Пример:* **<*p*>**Некоторый **<*br* />**текст**.</*p*>** *Будет выведено:* Некоторый текст.
<*pre*>	Устаревший, но широко применяемый элемент, позволяющий обеспечить вывод текста, включая все переводы строк. **<*pre*>**Некоторый текст**.</*pre*>** *Будет выведено:* Некоторый текст.
<*ul*>	*Unordered list*. Позволяет разметить список, элементы которого в зависимости от значения атрибута type (устаревшего) будут отмечены как круг, квадрат или окружность. В разметке каждый элемент списка указывается тегами **<*li*>… .** *Пример:* **<ul type=”disk”>** ** **Кофе** ** ** **Чай** ** ** **Молоко** ** ** ** *Будет выведено:* Кофе Чай Молоко
<*ol*>	*Ordered list*. Позволяет разметить список, элементы которого будут отмечены порядковым номером. В разметке каждый элемент списка указывается тегами ** … .** Имеются атрибуты (устаревшие) *start*, позволяющий указать стартовый номер, а также *type*, позволяющий указать тип нумерации: 1(арабские числа), A, a (буквы латинского алфавита), *I,i* (римские числа). *Пример:* ** ** ** **Кофе** ** ** **Чай** ** ** **Молоко** ** ** ** *Будет выведено:* □□ Кофе □□ Чай □□ Молоко
<*h1*>…<*h6*>	Устаревшие теги, предназначенные для того, чтобы выделить заголовки соответствующего уровня. Рекомендуется использовать стили.

1.2.5. Создание таблиц в *HTML*

настоящее время таблицы в *HTML* используются, в основном, для форматирования и оформления страниц, хотя и первоначальное их назначение как метода представления информации не утратило своего значения. Таблицы дают широчайшие возможности для оформления интернет-страниц. Рассмотрим подробнее их использование[2].

Элемент **</*table*>** служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются с помощью тегов **<*tr*>** и **<*td*>.** Таблица начинается с тэга **<*table*>** и заканчивается тэгом **</*table*>**. Общая структура таблицы:

```
<table>
…
</table>
```

Тэг <***table***> может включать несколько атрибутов:

- ***align="left* | *center* | *right"*** – устанавливает расположение таблицы по отношению к полям документа;
- ***background="URL"*** – Определяет изображение, которое будет использоваться в качестве фонового рисунка таблицы;
- ***bgcolor*="цвет"** – Устанавливает цвет фона таблицы;
- ***width*="значение"** – Задает ширину таблицы. Ее можно задать в пикселях (например, *width* =400) или в процентах от ширины страницы (например, *width* =80%). Если общая ширина содержимого превышает указанную ширину таблицы, то браузер будет пытаться «втиснуться» в заданные размеры за счет форматирования текста. В случае, когда это невозможно, например, в таблице находятся изображения, параметр *width* будет проигнорирован, и новая ширина таблицы будет вычислена на основе ее содержимого;
- ***border* ="толщина"** – устанавливает ширину внешней рамки таблицы и ячеек в пикселях (например, BORDER=4). Если атрибут не установлен, таблица показывается без рамки;
- ***bordercolor* ="цвет"** – Устанавливает цвет рамки таблицы;
- Для задание контейнера для строк таблицы служит тег **<*tr*>**.

Общая структура:

```
<table>
<tr>…..</tr>
<tr>…..</tr>
…
</table>
```

Атрибуты тега:

- ***bgcolor*="цвет"** – Устанавливает цвет фона строки таблицы;
-
- ***bordercolor*="цвет"** – Устанавливает цвет рамки вокруг строки. Рамка показывается, когда установлен параметр *border* с ненулевым значением у тега **<*table*>**.
- ***align="left* | *center* | *right* | *justify***" – Задает выравнивание содержимого ячеек строки по горизонтали. Выравнивание осуществляется для всех ячеек в пределах одной строки.

— ***valign="top | middle | bottom | baseline"*** – Устанавливает вертикальное выравнивание содержимого ячеек в строке. По умолчанию содержимое ячейки располагается по ее вертикали в центре ***middle***.
— Допустимые значения:
— ***top*** – Выравнивание содержимого ячеек по верхнему краю строки;
— ***middle*** – Выравнивание посередине;
— ***bottom*** – Выравнивание по нижнему краю.
— **Пример.** Задать таблицу с синим фоном. Для первой строки установить золотой цвет.

```
<table bgcolor="blue" width="30%"> <tr bgcolor="#ffcc00">

<td> ячейка 1 </td>
</tr>
<tr>
<td> ячейка 2 </td>
</tr>
</table>
```

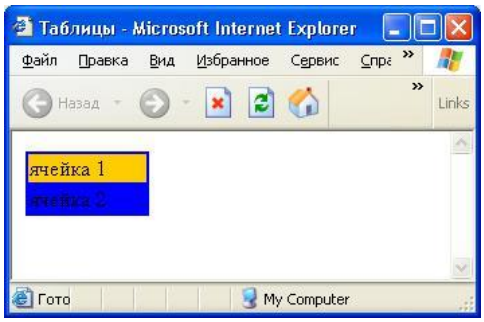

Рис. 1.1 — таблица с синим фоном

Пример. Задать таблицу с черным цветом сетки. Для первой строки установить красный цвет сетки

```
<table bordercolor="black" border="1" width="100%">
<tr bordercolor="red"> <td> ячейка 1 </td> </tr>
<tr>
<td> ячейка 2 </td> </tr>
</table>
```

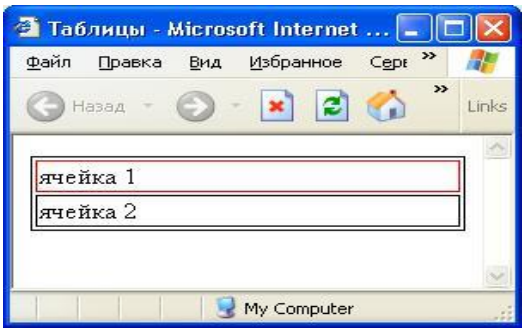

Рис. 1.2 — пример таблица

Для задание ячеек таблицы предназначен тег **<*td*>.** Контейнер должен размещаться внутри тега **<*tr*>.**

```
<table>
<tr>  <td>ячейка 1, 1-строки </td> …. <td> ячейка n, 1-строки
</td> </tr>
<tr>  <td>ячейка 1, 2-строки </td> …. <td> ячейка n, 2-строки
</td> </tr>
…
</table>
```

Атрибуты тега:

— ***align="left | center | right | justify"*** – Задает выравнивание содержимого ячейки по горизонтали.

— ***valign="top | middle | bottom | baseline"*** – Устанавливает вертикальное выравнивание содержимого ячейки.

— ***background="URL"*** – Определяет изображение, которое будет использоваться в качестве фонового рисунка таблицы.

— ***bgcolor*="цвет"** – Устанавливает цвет фона ячейки.

— ***bordercolor*="цвет"** – Устанавливает цвет рамки вокруг ячейки.

— ***colspan*="число"** – Устанавливает число ячеек, которые должны быть объединены по горизонтали.

— ***rowspan*="число"** – Устанавливает число ячеек, которые должны быть объединены по вертикали.

— ***height*="значение"** – Браузер сам устанавливает высоту таблицы и ее ячеек исходя из их содержимого. Однако при использовании параметра ***height*** высота ячеек будет изменена. Здесь возможны два варианта. Если значение ***height*** меньше, чем содержимое ячейки, то этот параметр будет проигнорирован. В случае, когда установлена высота ячейки, превышающая ее содержимое, добавляется пустое пространство по вертикали

— ***width*="значение"** – Задает ширину ячейки. Если общая ширина содержимого превышает указанную ширину ячейки, то браузер

будет пытаться «втиснуться» в заданные размеры за счет форматирования текста. В случае, когда это невозможно, например, в

ячейке находятся изображения, параметр ***width*** будет проигнорирован, и новая ширина ячейки будет вычислена на основе ее содержимого.

```
<table width="75%" border="2" cellspacing="0" cols="3" bgcol-
or="gold" align="center" bordercolor="black" >
<tr align="center">
<td rowspan="2">Пара</td>
<td colspan="2">День</td>
</tr>
<tr align="center">
<td>Понедельник</td>
<td>Вторник</td>
</tr>
<tr>
<td rowspan="2" bgcolor="red">8:00-9:30</td>
<td>Пение</td>
<td>Рисование</td>
</tr>
<tr>
<td>Информатика</td>
<td>Математика</td>
</tr>
<tr>
<td rowspan="2" bgcolor="red">9:40-11:10</td>
<td>Философия</td>
<td>История</td>
</tr>
<tr>
<td>Физика</td>
<td>Химия</td>
</tr>
</table>
```

Рис. 1.3 — Таблица расписания

1.2.6. Вставка изображений

Для вставки изображений применяется элемент <***img***>.Элемент ***img*** имеет обязательный атрибут ***src***, который должен содержать *URL*

графического файла. Любой современный браузер поддерживает отображение форматов *png, jpg, gif*. Рекомендуется использование формата *png* по причине отсутствия лицензионных ограничений.

Другой обязательный атрибут – ***alt***. Его использование необходимо для того, чтобы в браузерах, которые не могут отобразить изображение, вместо изображения был выведен текст.

Обратите внимание на то, что значение атрибута ***alt*** также используется поисковыми системами для того, чтобы ассоциировать изображение, указанное в атрибуте ***src*** с текстом.

Необязательные атрибуты ***width*** и ***height*** указывают ширину и высоту изображения в пикселах или процентах. Их использование необходимо для корректного отображения страницы до того, как браузер загрузил изображение. Помните о том, что каналы связи не идеальны, а скорости порядка 10-100 килобит до сих пор встречаются в условиях плохого покрытия у беспроводных модемов, включая технологии *3G* и *LTE*! Если не учитывать скорость загрузки страниц, пользователи, просматривающие такие страницы, будут видеть постоянно меняющуюся разметку.

Пример:

```
<img src="picture.png” width=”100” height=”200” alt=”Диаграмма” />
```

1.2.7. Вставка ссылок

Гиперссылки на *Web* - страницы - одно из основных свойств *WWW*. С помощью таких ссылок и родилась Всемирная паутина.

Для создания гипертекстовой ссылки используется специальный парный тег

```
<a href= "Значение ссылки”> </a>.
```

Рассмотрим подробнее структуру гиперссылки на примере:

```
<a href=”http://www.sevsu.ru ” title=”NEWS SevGU”> Новости
СевГУ </a>.
```

Здесь *http://www.sevsu.ru* - представляет *URL* - адрес файла в *Internet*.

«Новости СевГУ» - текст, который видит пользователь на экране обозревателя. Этот текст обычно выделен синим цветом, и при наведении на него мыши появляется характерный указатель в виде указательного пальца.

Если документ, на который ссылается данная ссылка, находится в той же самой директории, что и исходный документ, то в качестве *URL* - адрес файла можно просто указать имя файла, например.

```
<a href="Second_Page.html"> Следующая страница </a>.
```

Разумеется, ссылаться можно не только на *HTML*-страницы, но и на любые файлы, будь то картинки, будь то фильмы, будь то музыка, будь то архивы, будь то ещё всё, что угодно.

Последний атрибут - это *"title"*. Этот атрибут задаёт текст, который будет виден при наведении мышки на ссылку.

Что касается того, что внутри тега, то в большинстве случаев - это обычный текст. Очень часто делают картинку внутри тега ***<a>***, тогда эта ссылка будет в виде картинки.

Тег ****** означает конец гиперссылки.

1.3. Ход выполнения работы

В качестве примера, разработаем простой сайт, состоящий из 2 страниц. Главный *html* файл будет иметь имя *index.html*. В него запишем следующий код :

```
<!doctype html>
<html lang="ru">
 <head>
  <meta charset="utf-8" />
   <title> Главное меню </title>
 </head>
 <body background="images/фон.jpg">
  <img align="left" src="images/севгу.jpg" width="30%">
  <img align="right" src="images/ириб.jpg" width="19%">
         <h1 align="center"> Главное меню </h1>
         <h2 align="center"><i> Небольшое вступление </i></h2>
    <p align="left"> <font size="5"> Привет, меня зовут Вася и
этот сайт обо мне. <br>
     Обучаюсь в <u>Севастопольском Государственном
университете,</u> <br>

     в институте радиоэлектроники и информационной
безопасности, <br>
     на кафедре <font color="blue"><u> "Физика живых систем"
</u></font><br><br>
     Учусь в основном на <font
color="red"><b>"Отлично"</b></font></p><br>
        Про мои увлечения, также перейдя по этой ссылке: <a
href="hobbi.html" > Увлечения и хобби</a></font> <br>
</body>
</html>
```

Для корректного отображения текста на странице необходимо в секции ***<head>*** прописать обязательно кодировку страницы.

```
  <meta charset="utf-8" />
```

Что бы установить фон страницы, в теги <***body***> нужно указать атрибут ***background***, в котором нужно указать название изображения, которое будет выступать фоном.

```
<body background="images/фон.jpg">
```

Тег <***img***> как было описано в пункте 1.2.6. позволяет вставить изображение на страницу сайта. Для этого в атрибуте ***src*** пропишем ссылку на изображение:

```
  <img align="left" src="images/севгу.jpg" width="30%">
```

Атрибут ***align*** позволяет выровнять изображение по левому краю страницы.

Для перехода на другую страницу необходимо создать гиперссылку:

```
<a href="hobbi.html" > Увлечения и хобби</a>
```

В атрибуте ***href*** тега <***a***> указывается, на какую страницу будет осуществлять переход. Так как *html* файлы *index.html* и *hobbi.html* находятся в одной папке, можно указывать только имя файла, на который будет переход. Между открывающим тегом <***a***> и закрывающим тегом </***a***> указывается название гиперссылки.

Создадим второй файл под именем *hobbi.html* и запишем следующий код:

```
<!doctype html>
<html lang="ru">
 <head>
  <title> Увлечения и хобби </title>
 </head>
 <body background="images/фон.jpg">
         <h2 align="center"> О моих увлечениях и хобби </h2>
                 <p align="center"> <a
href="index.html">Главное меню</a>


<table align="center" border="1" width="40%">
     <td align="center" bgcolor="yellow"
colspan="3"><b>Увлечения</b></td>
    <tr >
         <td> <b>Увлечение</b> </td>
         <td> <b>Любимое</b> </td>
       <td> <b>Картинка</b></td>
     </tr>
    <tr >
         <td> <b><u> Музыка </u></b></td>
         <td> Би-2, Imagine Dragons </td>
       <td> <img src="images/Би-2.jpg" width="20%"> <img
src="images/ВД1.jpg" width="19%"> </td>
     </tr>
```

```
     <tr>
         <td> <b><u>Кино</b></u> </td>
         <td> Комедии, детективы, приключения, мюзиклы </td>
       <td> <img src="images/пила7.jpg" width="15%"> <img
src="images/Шерлок.jpg" width="20%"> </td>
     </tr>
     <tr>
         <td> <b><u>Книги</b></u> </td>
         <td> Энциклопедии </td>
       <td> <img src="images/Энциклопедия.jpg" width="15%">
<img src="images/Цвет.jpg" width="20%"> </td>
     </tr>
     <tr>
         <td> <b><u>Наука</b></u> </td>
         <td> Биология (альгология), физика, химия </td>
       <td> <img src="images/Водоросль.jpg" width="15%"> <img
src="images/Трофимова.jpg" width="15%"> </td>
     </tr>
    <tr>
         <td> <b><u>Игры</b></u> </td>
         <td> World of Warcraft, Dark souls, Spore </td>
       <td> <img src="images/Wow.jpg" width="15%"> <img
src="images/Spore.jpg" width="15%"> </td>
     </tr>
    <td align="center" bgcolor="yellow" col-
span="3"><b>Хобби</b></td>
    <tr>
         <td> <b><u>Азартные игры</b></u> </td>
         <td> Игры в карты, фокусы </td>
       <td> <img src="images/карты.jpg" width="15%"> </td>
     </tr>
    <tr>
         <td> <b><u>PaperCraft</b></u> </td>
         <td> Бумажные изделия в виде голов дракона, черепа
</td>
       <td> <img src="images/PaperDrako.jpg" width="15%"> </td>
     </tr>
</table>
</body>
</html>
```

В данной странице расположена краткая характеристика увлечений студента. Увлечения размещены в таблице с помощью парного тега ***<table>*.**

Для возращения на предыдущую страницу нужно обязательно прописать гиперссылку на эту страницу.

```
a href="index.html">Главное меню</a>
```

1.4. Порядок выполнения

1.4.1. Создать личный Web - сайт с информацией о себе, включающий минимум, две страницы. На первой странице с названием *index.html* поместите общую информацию о себе, фотографию. На второй более частные подробности, например, информацию о своих планах на следующую неделю, предыдущие места учебы, работы, Ваших увлечениях и т.п. и оформить в виде таблицы.

1.4.2. Подготовить отчет по работе.

1.5. Содержание отчета

1.5.1. Сформулировать цель работы.

1.5.2. Привести постановку задачи.

1.5.3. Привести краткие теоретические сведения о технологии *HTML*,и описать основные элементы разметки.

1.5.4. Привести тексты программ**.**

1.5.5. Составить таблицу, содержащею теги использованные для создания *HTML* – документа(см. таблица 1.3).

Таблица 1.3 — имя и назначение тегов

Имя тега	Назначения тега

1.5.6. Привести скриншоты выполнения программы.

1.5.7. Выводы.

1.6. Контрольные вопросы

1.6.1. Что такое технология *HTML*?

1.6.2. Что такое тег?

1.6.3. Привести структуру *HTML* документа. Описать назначение тегов ***<html>***, ***<head>, <meta>, <body>.***

1.6.4. Что такое атрибут тега? Формат записи атрибутов.

1.6.5. Опишите теги для форматирования текста в *HTML*

1.6.6. Как представляются гиперссылки в *HTML* документе?

1.6.7. Как включаются графические объекты в *HTML* документы?

2. ЛАБОРАТОРНАЯ РАБОТА №2. ИСПОЛЬЗОВАНИЕ *CSS* ПРИ РАЗРАБОТКЕ WEB -САЙТА.

2.1. Цель работы

Научиться работать с каскадными таблицами стилей, а также изучить основные методы работы с селекторами и построения сайта блочной верстки.

2.2. Теоретические сведения

2.1.1. Назначение стилевых таблиц

Собственные средства *HTML* (теги и их атрибуты) выполняют две основные роли: поддержку структуры документа (состав и взаимосвязи элементов) и определение внешнего вида визуальных элементов. Идея разделения описания внешнего вида документа от элементов, определяющих его структуру, воплотилась в технологии, называемой каскадными таблицами стилей (*Cascading Style Sheets, CSS*)[3]. Таблица стилей, действует подобно шаблону форматирования, может быть разработана отдельно от *HTML*-документа, а затем применена к нему. Изменяя содержимое таблицы стилей, можно изменять внешний вид *HTML*-документов, не затрагивая их структуры информационного содержания. Одна и та же таблица стилей может применяться к нескольким документам, и, наоборот, к одному и тому же документу может быть применено несколько таблиц стилей. В последнем случае браузер учитывает приоритеты таблиц и по определенным правилам разрешает возникающие конфликты, в результате чего таблицы выстраиваются неким каскадом.

Кроме технологичности стилизации *HTML*-документов, *CSS* обеспечивают еще две важные вещи: произвольное позиционирование элементов и создание визуальных эффектов, таких как полупрозрачность и трансформации графических изображений и текстов.

Основные проблемы, с которыми сталкивались разработчики сайтов до появления *CSS*:

— Проблема позиционирования (разметка страниц). Хотя для этого стали использовать таблицы (они небыли для этого предназначены изначально), они не решили всех проблем. Невозможно осуществить следующее:
1. Нельзя задать фиксированные размеры какого-нибудь блока. Например, текстовый блок будет разъезжаться при переполнении (даже если он в таблице).
2. Нельзя задать фиксированные координаты положения блока на странице.

3. Нельзя наложить один блок на другой. Например, попробуйте надвинуть картинку поверх таблицы или другой картинки.

— Вторая проблема заключалась в том, что приходилось каждый раз задавать на страницах размер и цвет шрифта, свойства ячеек таблицы и т.д. Это сильно увеличивало размеры страницы, а значит, она медленнее загружается. В случае если вам захотелось изменить цвет или размер шрифта, вам пришлось бы редактировать все страницы.

— Третья проблема заключалась в том, что все браузеры имеют индивидуальные настройки. Например, в браузере можно увеличить шрифт, что приведет к искажению всей страницы.

С помощью *CSS* эти проблемы можно решить.

2.1.2. Встраивание таблиц стилей в *HTML*-документ

Для применения каскадной таблицы стилей к *HTML*-документу необходимо ее связать с ним или встроить в него. Это можно сделать четырьмя способами:

1. Вставка непосредственно в заголовок *HTML*-документа. Правила таблицы стилей заключаются в контейнерный тег ***<style>;***

```
<head>
<style>
H1 {color: blue; font-size: 24pt}
b {font-style: italic}
</style>
</head>
```

2. Вставка непосредственно тег виде строки описания в атрибуте *Style;*

```
<h1 style="color: blue; font-size:24pt">
```

3. Импорт - вставка таблицы стилей из внешнего файла. Файл таблицы стилей является текстовым файлом с расширением ***css***. Оператор ***@import*** используется перед другими правилами таблицы стилей в контейнере <***style***> или в ***css***-файле.

```
<style>
@import: url(http://www.myserver.ru/css/mystyle.css)
</style>
```

4. Связывание с таблицей стилей в внешнем файле с помощью ссылки задаваемой тегом <***link***>, который помещаемого в контейнер <***head***>. Общий вид:

```
href="http:// www.myserver.ru/css/mystyle.css"> </head>
```

Таблица стилей, вставленная с помощью тега **<*style*>,** действует на элементы только текущего *HTML*-документа, в котором этот тег находится. Если ту же таблицу необходимо применить и к другим документам, то ее код придется повторить в соответствующих *HTML*-

страницах. При этом возрастает общий объем файлов сайта, а также трудозатраты в случае необходимости изменить его стиль. Чтобы избежать этого, используют импорт или связывание таблиц из внешних ***css***-файлов. Когда требуется изменить параметры стилей для отдельных элементов (например, их оформление), используют атрибут *style***.** Возможно также комбинирование всех способов встраивания таблиц.

В записях таблиц стилей можно задавать комментарии, которые задаются символами /* и */.

В тег **<*style*>** имеет следующие атрибуты:

type - для каскадных таблиц стилей всегда имеет значение ***text/css*;**

В ***media*** - определяет тип устройства вывода. Браузеры обычно используют следующие значения: ***screen*** (экран), ***print*** (печать) и ***all*** (все). Можно создать стили отдельно для отображения документа на экране монитора и для вывода на печать.

2.1.3. Типы селекторов

CSS состоит из **правил**, а каждое правило – из **селектора** и **блока объявлений**. Блок объявлений содержит *CSS*-свойства, определяющие отображение элемента веб-страницы в браузере. Селектор отвечает за выбор этого самого элемента. Селектор служит для однозначной идентификации *HTML* элемента средствами *CSS*. Он позволяет выбирать именно тот элемент (или группу элементов), который нужен. С помощью **простых** селекторов можно выбирать[3]:

- все объекты – универсальный селектор;
- объекты определенного типа;
- объекты с заданным классом;
- объект с определенным идентификатором;
- объекты с определенными характеристиками – селекторы атрибутов.

Объединяя простые селекторы можно выбирать объекты по более сложным правилам:

- объекты, находящиеся внутри какого-то объекта – селектор потомка;
- объекты, непосредственно вложенные в какой-то объект – дочерний селектор;
- объект, расположенный после другого объекта – сестринский се-
- лектор.

Также существуют селекторы псевдоклассов и псевдоэлементов. Они позволяют назначать стили элементам, которые зависят не только от разметки, но и от состояния документа. Все объявления *CSS* (официально они называются "селекторы") записываются в фигурных скобках:

```
ТЕГ:псевдокласс.класс { характеристика1: значение1;
характеристика2: значение2}
```

Описание каждого класса делается при помощи конструкции, подобной этой:

```
.small { font-size: 9pt; }
```

Сначала указывается имя класса - оно может быть произвольным, но желательно все-таки давать осмысленное название. Далее, в фигурных скобках {} перечисляются все необходимые параметры для данного класса. Параметры отделяются друг от друга точкой с запятой. Вот еще один пример, в котором используется более длинное описание:

```
.small { font-size: 9pt; color: #eeeeee; text-align: center; }
```

2.1.3. Универсальный селектор

Предназначен для выбора всех элементов. Стили, указанные для универсального селектора применяются ко всем элементам сразу.

Обычно применяется для сброса зависящих от браузера начальных значений стилей (в частности, отступов).

```
* {
padding: 0;
margin: 0;
}
```

2.1.4. Селектор типа

Предназначен для выбора всех элементов определенного типа. Стили применяются ко всем элементам указанного типа не зависимо от уровня вложенности. Применяется для задания общих, для всех элементов определенного типа, стилей. Общий вид:

```
Имя тега { описание }
```

Пример. Для всех абзацев (тег p) установить размер шрифта 12px.

```
p {font-size: 12px;}
```

2.1.5. Селектор класса

Предназначен для выбора всех элементов по имени класса (по значению атрибута *class*). Стили применяются к любым тегам с соответствующим классом. Важно учитывать, что в отличие от названий *HTML*-тегов, в названиях классов различаются большие и маленькие буквы. То есть *class*="*active*" и *class*="*Active*" – это совсем разные классы. Общий вид:

```
.Имя класса { описание }
```

Пример. Задать зеленый цвет текста в любых тегах с классом ***active***.

```
.active {color: #0f0; }
```

Комбинируя селектор класса и селектор типа, можно объединить их свойства.

Пример. Задать зеленый цвет текста только для абзацев с классом ***active***.

```
p.active {color: #0f0; }
```

HTML позволяет задавать в качестве значения атрибута ***class*** список разделенных пробелами названий (порядок следования не имеет значения). Другими словами, один элемент может иметь сразу несколько классов:

```
<div class="panel hint active"></div>
```

Стили, относящиеся к каждому из перечисленных классов, будут, объединяясь, применяться к этому элементу. *IE6* не понимает такой записи.

Пример. Правило распространяется только на элементы, в списке классов которых встречаются (в любом порядке) и класс *popup* и класс *active*.

```
.popup.active { color: #0f0;}
```

Концепция классов, наиболее часто применяется при верстке. Например, верстать страницу, используя в качестве контейнеров теги *div*, задавая им определенные классы (в соответствии с функциональным назначением):

```
<div class="header">
<!--элементы шапки сайта-->
</div>
<div class="sideBar">
<!--элементы панели меню-->
</div>
<div class="content">
<!--основное содержимое-->
</div>
```

2.1.6. Селектор идентификатора

Предназначен для выбора элемента по уникальному идентификатору (значению атрибута *id*). Позволяет задать стили конкретному *HTML*-элементу. Общий вид:

```
#Имя идентификатора { описание }
```

Пример. Задать зеленый цвет текста в любом теге с *id="active"*

```
#active { color: #0f0; }
```

Комбинируя селектор идентификатора и селектор типа, можно объединить их свойства. Учитывая, что id на странице должен быть уникальным, смысла в таком комбинировании не много. Как правило, идентификаторы применяются там, где предполагается работа скриптов. Например, в формах или в каких-то динамических элементах. Так же с помощью *id* можно подчеркнуть уникальность элемента, его присутствие в единственном экземпляре.

Пример. Задать зеленый цвет только для элемента списка с данным *id*.

```
li#active { color: #0f0; }
```

2.1.7. Псевдоклассы

В *CSS* есть такое понятие как псевдокласс. В отличие от обычного класса, действие псевдокласса распространяется не на весь текст, к которому применен данный стиль, а лишь на его часть и возможно лишь в определенном состоянии. Чтобы было понятнее, давайте разберем эффект, при котором ссылки подчеркиваются лишь при наведении на них курсора. Эффект достаточно распространенный, и его можно наблюдать на всех сайтах. Вот фрагмент таблицы стилей, который позволяет достигать вышеописанного эффекта:

```
a { text-decoration: none; }
a:hover { text-decoration: underline;}
```

Верхняя строчка - это переопределение стандартного тега <a>, которое запрещает подчеркивать ссылки, а вот нижняя - это определение стиля для псевдокласса hover, который описывает стиль ссылки в момент, когда курсор находится над ней.

А вот и другой пример псевдокласса - определение буквицы вначале абзаца:

```
p:first-letter { font-size: 200\%; font-weight: bold; }
```

Заметьте, что и в том, и в другом случае действие стиля распространяется либо на определенное состояние (пользователь собирается щелкнуть по ссылке), либо на фрагмент текста (изменяется только первая буква абзаца). В этом и заключается смысл псевдоклассов.

Основные псевдоклассы :

— A:active{} Таблица стилей для активных ссылок (при нажатии)
— A:link{} Таблица стилей для собственно ссылок
— A:visited{} Таблица стилей для посещённых ссылок

— A:hover {} Таблица стилей для ссылок при наведении указателя мыши

2.1.8. Единицы измерения

Единицы измерения *CSS* используются для указания размеров различных элементов. Есть абсолютные и относительные единицы измерения. Абсолютные единицы не зависят от устройства вывода, а относительные единицы определяют размер элемента относительно значения размера, используемого в родительском элементе. Ниже представлена таблица и подробное описание каждой единицы измерения *CSS*.

Абсолютные единицы:

Миллиметр - *mm*, сантиметр - *cm* и дюйм – *in*. Само собой разумеется, что это абсолютные единицы измерения. Один *cm* = 0.*39370in, 1in* = 2.54cm и 10mm = 1cm. Компьютерные дисплеи плохо вычисляют данные единицы измерения, таким образом, у этих величин ограниченное применение и обычно их используют при указании размера для вывода страниц на печать.

Пиксели – *px*. Пиксель это маленькая точка на экране. Пиксели определяют размер элемента. Использование пикселей дает вам точный контроль над размером элемента, позволяя вам точно вычислить его ширину и высоту, это будет полезным для точной разметки дизайна страницы. Однако, есть несколько минусов использования пикселей, вы должны быть осведомлены о них:

Во-первых, установка размера шрифта с помощью пиксельных единиц не позволяет пользователю изменять размер шрифта с помощью настроек в браузере. Если размер шрифта 12 пикселей, он всегда будет иметь высоту 12 точек, независимо от того, что пользователь установил размер шрифта по умолчанию в браузере. Так что, если вы решили указать размер шрифтов, следует подумать об использовании другой единицы измерения.
Во-вторых, когда речь заходит о печатных средствах массовой информации, пиксели не имеют реального значения. При разработке документа для печати, устройство печати должно будет само догадываться о том, что вы имели ввиду с точки зрения физических размеров. Хотя обычно можно просмотреть документ и внести изменения перед печатью
Точки - *pt* и пики – *pc*. Эти единицы измерения чаще всего используются для указания именно размера шрифта, например, в обычном блокноте, мы часто указываем размер текста равный 14, это значение как раз и есть единицы измерения *pt*.

Точка (*1pt*) равна 1/72 дюйма, в то время как пика (*1pc*) равна 1/6 дюйма (*1pc = 12pt*). Документы, предназначенные для печати, будут иметь

возможность сообщить устройству именно тот размер шрифта, который следует использовать при печати. Цифровым дисплеям, однако, придется самим догадываться, как конвертировать эти единицы
в пиксели, и нет никаких реальных универсальных способов узнать, как это будет сделано. Поскольку точки (*pt*) были использованы с первых дней Интернета, большинство браузеров автоматически могут установить соотношения между пикселями и точками, но это по сути неправильно. Помните, что небольшие дисплеи сегодня могут иметь высокое разрешение, так что определить то, насколько большой «дюйм на экране» будет практически невозможно на устройствах. Для указания размера элементов на веб-страницах следует избегать использования этих единиц.

Относительные единицы:

— **Процент - %.** Самая простая единица измерения, это процент (%), она не имеет напрямую никакого отношения к размеру шрифта или элемента в целом и может быть использована в комбинации с другими единицами измерения, указывающими величину. Размер установленный в процентах, напрямую зависит от размера родительского элемента, например, размер шрифта задается относительно размера шрифта родительского элемента, также высота и ширина выражается в процентах относительно высоты и ширины родительского элемента;

— **Вычисляемая x-высота – *ex*.** Единица измерения *ex* используется достаточно редко. В качестве основы для размера *1ex* используется высота символа "x" в нижнем регистре указанного шрифта. Большинство браузеров не поддерживает эту единицу измерения должным образом, и она не рекомендуется для использования в документах, предназначенных для браузеров;

— **Вычисляемая единица – *em*.** *Em* является относительной единицей измерения. Один *em* равен *16px*. Если *em* используется для определенного элемента, то за 1em принимается размер шрифта его родителя.

Таблица 2.1 — Примеры использования единиц измерения.

а	**Описание**	**Пример**
%	проценты	**p{border: 5%;}**
in	дюйм	**p{word-spacing: 2in;}**
cm	сантиметры	**div {margin-left: 2cm;}**
mm	миллиметры	**p{font-size: 15mm;}**
em	1em равен текущему размеру шрифта. Если текущей размер шрифтп составляет 12pt	**p{letter-spacing: 7em;}**
ex	1ex равен высоте символа "x" в нижнем регист-	**p{line-height: 3ex;}**

а	Описание	Пример
	ре указанного шрифта	
pt	1pt равен 1/72 дюйма	**body{font-size: 18pt;}**
pc	1pc равен *12pt*	**p{font-size: 20pc;}**
px	пиксель - это маленькая точка на экране	**p{padding: 25px;}**

Пример. Использование относительных единиц

```
<head>
<style type="text/css">
.em, .ex, .px, .percent {
font-family: Verdana, Arial, sans-serif;
.em { font-size: 2em; }
.ex { font-size: 2ex; }
.px { font-size: 30px; }
.percent { font-size: 200%; }
</style>
</head>
<body>
<p class="em">Размер 2 em</p>
<p class="ex">Размер 2 ex</p>

<p class="px">Размер 30 пикселов</p> <p class="percent">Размер
200%</p> </body>
```

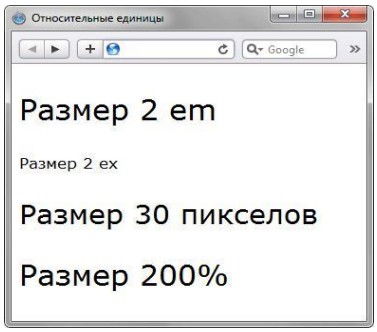

Рис. 2.1 — Использование относительных единиц

Пример. Использование абсолютных единиц

```
<style type="text/css">
.in, .mm, .pt {
font-family: Verdana, Arial, sans-serif;
}
.in { font-size: 0.5in; }
.mm { font-size: 8mm; }
.pt { font-size: 24pt; }
</style>
</head>
<body>
<p class="in">Размер 0.5 дюйма</p>
```

```
<p class="mm">Размер 8 миллиметров</p> <p class="pt">Размер 24
пункта</p> </body>
```

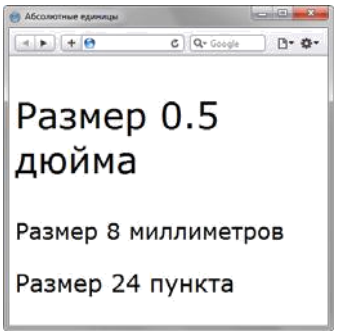

Рис. 2.2 — Использование абсолютных единиц

2.1.9. Описание шрифтов

Для описания шрифтов с помощью *CSS* применяются следующие стилевые параметры:

1. ***font-family*** - определяет список семейств шрифтов. Обычно задаются несколько похожих шрифтов в порядке предпочтения на случай, если в компьютере пользователя нет нужного шрифта при указании имени группы шрифтов, как показано выше, браузер подбирает подходящий для отображения шрифт данной группы из имеющегося набора шрифтов. Если название шрифта состоит из нескольких слов, то оно заключается в кавычки. Список шрифтов желательно завершить родовым именем шрифта: *serif***,** *sans-serif***,** *cursive***,** *fantasy* или *monospace***.** Например, для шрифта *Times* родовым является *serif***,** для *Helvetica* – *sans-serif***,** для *Courier* – monospace**.**

Пример. Прямое указание гарнитуры шрифта:

```
<SPAN STYLE="font-family:symbol; padding-left:65px;">N</SPAN>
<BR>
<SPAN STYLE="font-family:symbol; font-size:24px;">A
<SUB>x,y</SUB>=e(y<SUB>a</SUB>+x<SUB>a</SUB>) </SPAN> <BR>
<SPAN STYLE="font-family:symbol; padding-left:60px; font-
size:20px;">a=1 </SPAN>
```

Рис 2.3 — Использование шрифтов

2. ***font-size*** – размер шрифта. Значение может быть задано различными способами:

− абсолютный размер, задаваемый с помощью ключевых слов**: *xx-small, x-small, medium, large, x-large, xx-large***. Эти значения представляют индексы таблицы размеров шрифтов, поддерживаемой браузером. По умолчанию используется значение ***medium*** (средний);

− относительный размер, задаваемый с помощью ключевых слов: ***larger*** (больше) и ***smaller*** (меньше). Эти значения интерпретируются относительно таблицы размеров шрифтов браузера и размера шрифта элемента-родителя. Например, если элемент-родитель имеет шрифт размером ***medium***, то значение ***larger*** для элемента-потомка сделает размер шрифта равным ***large*** (большой). В *CSS1* масштабирующий множитель равен 1,5, в *CSS2* – 1,2. Размер шрифта относительно элемента-родителя можно задавать и в процентах;

− размер, задаваемый в абсолютных единицах длины (обычно pt=0,35мм).

Пример. Задание размера шрифта.

```
<P STYLE="font-size:12pt;">
Кегль параграфа установлен в 12 пунктов</P> <P STYLE="font-
size:12px;">
Кегль параграфа установлен в 12 пикселов</P> <P STYLE="font-
size:120%;">
Кегль параграфа установлен в 120% от размера букв
охватывающего параграф элемента</P>
Рис 2.4 - Задание размера шрифта
<P STYLE="font-size:large;">Размер кегля large</P>
<P STYLE="font-size:small;">Размер кегля small</P>
<P STYLE="font-size:x-small;">Размер кегля x-small</P>
<P STYLE="font-size:xx-small;">Размер кегля xx-small</P>
```

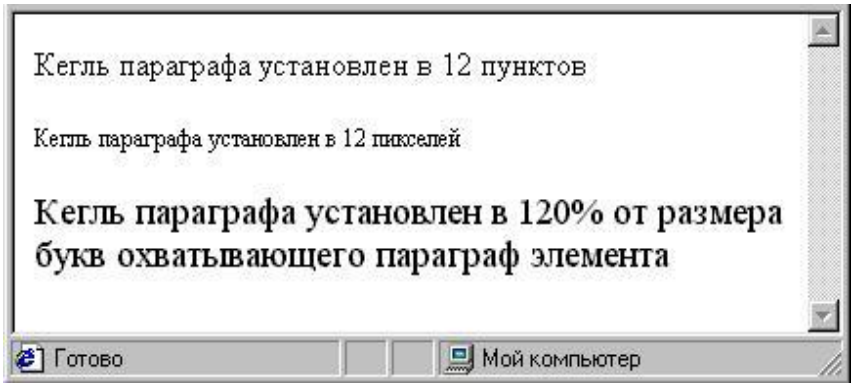

Рис 2.5 — Задание размера шрифта

3. ***font-weight*** – жирность шрифта. Возможные значения: ***normal***, ***bold*** (жирный), ***bolder*** (жирнее), ***lighter*** (светлее) или одним из девяти целых чисел от 100 до 900. Ключевому слову ***normal*** соответствует числовое значение 400; ключевое слово ***bold*** задает обычный жирный (полужирный) шрифт и соответствует числовому значению 700. Следует иметь в виду, что в текущем семействе может и не быть шрифта с заданной степенью жирности. В этом случае можно лишь гарантировать, что шрифт с большим значением параметра **font-*weight*** будет не светлее, чем шрифт с меньшим значением этого параметра.
4. **font-*style*** – стиль шрифта. Возможные значения: ***normal*** (нормальный прямой), ***italic*** (курсив) и ***oblique*** (наклонный). По умолчанию применяется значение ***normal***. Если уже имеется готовый шрифт, соответствующий заданному стилю, то он будет применен. В противном случае текущий шрифт будет изменен программным способом.
5. ***font-variant*** – вариант стиля шрифта. Возможные значения: ***normal*** (принимается по умолчанию) и ***small-caps***. Значение ***normal*** не влияет на отображение шрифта. Значение ***small-caps*** заменяет строчные буквы прописными, но делает их несколько меньшими по размеру, чем прописные буквы текущего шрифта.

Пример. Задания свойств шрифта

```
<html>
<head>
<style>
#MyType1 {font-family: "Times New Roman", serif; font-size:
larger}
#MyType2 {font-weight: bolder; font-style: oblique; font-
variant:small-caps}
</style>
<body style="font-family:fantasy; font-size:18pt">
<p>Текст Текст Текст <span id=MyType1>Текст Текст Текст Текст
</span>Текст Текст <span id=MyType2>Текст Текст Текст Текст
</span>Текст Текст Текст Текст Текст Текст Текст Текст Текст
</p>
</body>
</html>
```

Рис 2.6 — Использование шрифтов

6. **font** – позволяет установить сразу несколько свойств шрифта в одном определении: *font-style, font-variant, font-weight, font-size, line-height, font-family*. Значения этих параметров указываются через пробел в том порядке, в котором они были перечислены. Допустимо не указывать первые три параметра, что соответствует значению *normal*, принятому для них по умолчанию. Если задается высота строки (*line-height*), то ее значение отделяется от размера шрифта (*font-size*) прямым слэшем (/). Если список семейств шрифтов (*font-family*) содержит более одного элемента, то последние отделяются друг от друга запятыми.

Пример. Задания свойств шрифта одним определением.

```
<html>
<head>
<style>
#MyType1 {font: larger "Times New Roman"}

#MyType2 {font: oblique small-caps bolder 18pt} </style>
<body style="font-family:fantasy; font-size:18pt">

<p>Текст Текст Текст <span id=MyType1>Текст Текст Текст Текст
</span>Текст Текст <span id=MyType2>Текст Текст Текст Текст
</span>Текст Текст Текст Текст Текст Текст Текст Текст Текст
Текст Текст
</p>
</body>
</html>
```

2.1.10. Задание цвета и фона

Цвет текста определяется с помощью параметра ***color***, а цвет фона элемента – с помощью параметра ***background-color***. По умолчанию цвет фона элемента определен как прозрачный (***transparent***). Значения этих параметров задаются именем цвета или числовым представлением в системе *RGB*.

Числовое представление цвета допускает следующие варианты:

— шестнадцатеричное число (*#ffff00*);

— тройка целых десятичных чисел, каждое из которых представляет яркость одной из *RGB*-составляющих цвета в диапазоне от 0 до 255. (*rgb(255,255,0));*

— тройка вещественных чисел со знаками **%**, каждое из которых представляет яркость одной из **RGB**-составляющих цвета в диапазоне от 0 до 100%. *rgb*(100%, 100%, 0%).

Пример. Способы задания цвета.

```
<html>
<head>
```

```
<style>
/* текст синий, фон серый */
body {color: blue; background-color:#e0e0e0}
/* заголовок 1-го уровня красный на белом фоне */
h1 {color: #ff0000; background-color:white}
/* заголовок 2-го уровня зеленый на сером фоне */ h2 {color:
rgb(0, 255, 0)}
/* заголовок 3-го уровня зеленый на сером фоне */
h3 {color: rgb(0, 255, 0)}
/* заголовок 4-го уровня оранжевый на черном фоне */
h4 {color: rgb(100%, 50%, 0%); background-color: black}
</style>
<body>
<h1> Заголовок 1 </h1>
<h2> Заголовок 2 </h2>
<h3> Заголовок 3 </h3>
<h4> Заголовок 4 </h4>
<p>Текст Текст Текст Текст Текст Текст Текст Текст Текст
Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
Текст </p> </body>
</html>
```

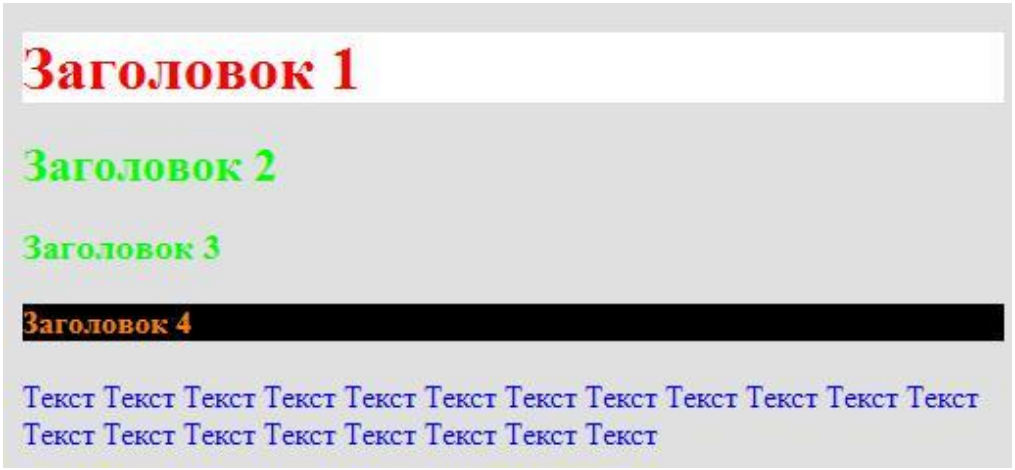

Рис. 2.7 — Способы задания цвета

В качестве фона элемента или всего документа, можно использовать изображение из файла по указанному *URL*-адресу. При этом можно задать способ заполнения, позиционирование изображения, а также указать, должно ли оно перемещаться при прокрутке документа, с помощью следующих параметров:

— ***background-image*** - принимает в качестве значения *url* (URL-адрес изображения) или none (отсутствие изображения).

Пример. Задание фонового изображения

```
<style>
body {background-image: url(/img/picture.jpg)}
```

```
</style>
```

- ***background-repeat*** - определяет способ заполнения области элемента изображением. Возможны следующие значения:
  1) ***repeat*** - заполнение по горизонтали и по вертикали (по умолчанию);
  2) ***repeat-x*** - заполнение по горизонтали;
  3) ***repeat-y*** - заполнение по вертикали;
  4) ***no-repeat*** - изображение выводится в единственном экземпляре.
- ***background-attachment*** - определяет, будет ли фоновое изображение прокручиваться при пролистывании документа. Значением по умолчанию является ***scroll*** (будет прокручиваться). Для фиксации изображения используется значение ***fixed***;
- ***background-position*** – определяет начальное положение фонового изображения с помощью двух значений (горизонтальной и вертикальной координат), разделенных запятой. Значения задаются в виде процентов, в абсолютных значениях длины или в виде ключевых слов: ***top***, ***center***, ***bottom***, ***left***, ***right***. Значения по умолчанию: ***left***, ***top***;
- ***background*** - позволяет установить значения рассмотренных выше свойств: *background-color*, *background-image*, *background-repeat* и *background-attachment*. Эти значения указываются через пробел. Значение *transparent* означает отсутствие фона. Пример. Способы задания фона

```
<html>
<head>
<style>
h1{background-image: url(23.jpg);}
h2{background-image: url(22.jpg); background-repeat:no-
repeat;}
h3{background-image: url(22.jpg); background-repeat:no-repeat;
background-position:100% 0%}
body {background: silver url(26.jpg) repeat-x} </style>
<body> <h1>Заголовок 1</h1>
<h2>Заголовок 2 Заголовок 2</h2> <h3>Заголовок 3</h3>
<p>Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
Текст </p>
</body>
</html>
```

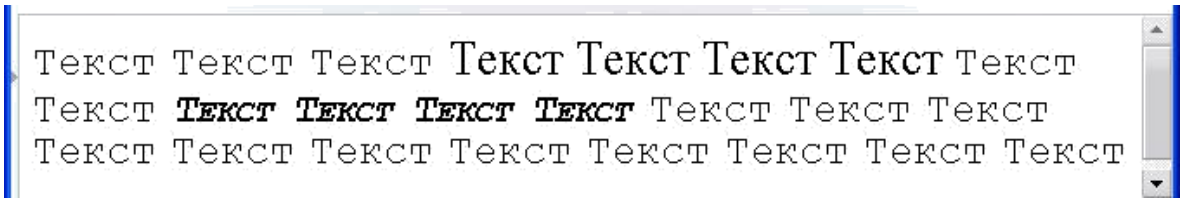

Рис 2.8 — Способы задания фона

2.1.11. Блочная модель документа. Размеры, поля, отступы, границы

Блочные элементы (блоки текста или ***box***) позволяют оперировать с текстом в терминах прямоугольников, которые этот текст занимает[4]. При этом блок текста становится элементом дизайна страницы с теми же свойствами, что и картинка, таблица или прямоугольная область приложения. Блок текста обладает свойствами: высоты (***height***), ширины (***width***), границы (***border***), отступа (***margin***), набивки (***padding***), произвольного размещения (***float***), управления обтеканием (***clear***) (рисунок 2.9).

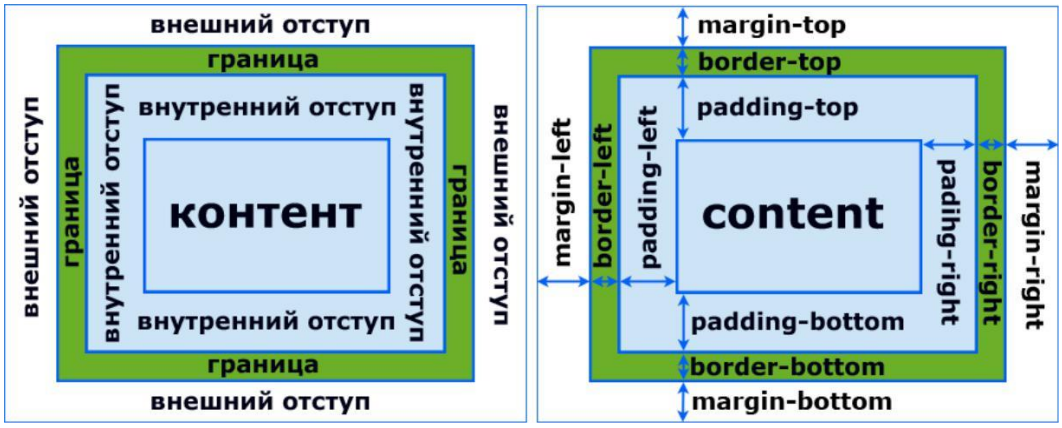

Рис. 2.9 — Блочная модель документа (*Padding* – отступ, набивка или внутренний отступ, ***Margin*** – поле или внешний отступ)

Размеры отступов, границ и полей можно задавать по своему усмотрению. Однако по умолчанию они имеют значение 0, т. е. не видны и не занимают место. Параметры ***height*** *и* ***with*** определяют размеры содержимого элемента, отступы задают положение содержимого относительно рамки, внешние поля позволяют позиционно отделить содержимое элемента с рамкой от других элементов документа.

Для полей и отступов можно задать размеры 4-х сторон с помощью следующих параметров:

— ***margin-top***, ***margin-right***, ***margin-left***, ***margin-bottom*** – размеры полей;

— ***padding-top***, ***padding-right***, ***padding-left***, ***padding-bottom*** – размеры отступов.

Значения этих параметров могут быть заданы числом с указанием единиц измерения или ключевым словом ***auto***.

Кроме того, можно использовать параметры ***margin*** и ***padding*** с перечислением через пробел размеров всех четырех сторон в следующем порядке: ***top***, ***right***, ***left***, ***bottom***. Если указано только одно значение, то оно будет задавать размер всех сторон. Если указаны два или три значения, то остальные будут равны размерам соответствующих противоположных сторон.

Пример. Задание поле и отступов.

```
<html>
<style>
P.1 {margin-left:50px;margin-right:5px; margin-top:15px; mar-
gin-bottom:50px; padding:0px;text-align:left; }
P.2 {padding-left:100px;text-align:right;} </style>
<body>
<p class=1>
Text Text Text Text Text Text Text Text Text Text Text Text
Text Text Text Text Text Text Text Text Text Text Text Text
Text Text </p> <p class=2>
Text Text Text Text Text Text Text Text Text Text Text Text
Text Text Text Text Text Text Text Text Text Text Text Text
Text Text </p>
</body>
</html>
```

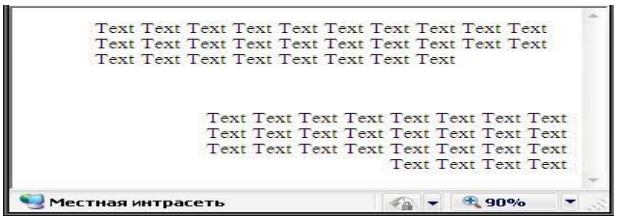

Рис. 2.10 — Задание поле и отступов

Для описания границ блоков применяются следующие атрибуты:

— размеры границы (рамки): ***border-top-width***, ***border-right-width***, ***border-left-width***, ***border-bottom***-width. Значения можно задавать в абсолютных единицах (% - нельзя) или ключевые слова: ***thin*** (тонкая), ***medium*** (средняя), ***thick*** (толстая), по умолчанию - ***medium***. Также можно использовать параметр ***border-width*** с перечислением размеров всех четырех сторон.

— цвет сторон границы: ***border-top-color***, ***border-right-color***, ***border-left-color***, ***border-bottom-color***. Параметр ***border-color*** можно использовать для задания сразу всех или только некоторых сторон;

— стиль границы: ***border-top-style, border-right-style, bor-der-left-style, border-bottom-style***. Может принимать значения: ***none***, ***dotted***, ***dashed***, ***solid***, ***double***, ***groove***, ***ridge***, ***inset***, ***outset***. Согласно спецификации ***CSS1***, может быть задан для каждой из границ блока. Указание типа линии границы поддерживается не всеми браузерами.

Параметр ***border-style*** можно использовать для задания стиля сразу всех или только некоторых сторон границы. Если стиль границы не задан в явном виде, другие ее параметры не будут действовать. Задание стиля (типа) границы в явном виде необходимо, чтобы она была отображена (по умолчанию граница не видна).

2.3. Ход выполнения работы

Для иллюстрации приемов блочной верстки возьмем задачу разработки сайта об автомобилях. Сайт преднозначен для посещения автолюбителей, интересующихся историей появления автомобиля, новинками автомобилестроения и т.п.

Сайт будет состоять из нескольких страниц, структуру которых надо спроектировать заранее.

Макет страницы может выглядеть следующим образом (рис.2.11):

Рис.2.11 — Верстка страниц сайта

Создадим папку в которой будет располагаться проект, имя данной папки будет, например «*lab2*».В этой папке нужно создать 2 папки:

— ***img*** - в ней будут располагаться все изображения использованные в проекте;

— ***css*** – в ней будут храниться файлы *css* для создания дизайна сайта.

В корне создадим файл с именем *index.html*.

Файловая структура разрабатываемого сайта представлена на рис 2.12.

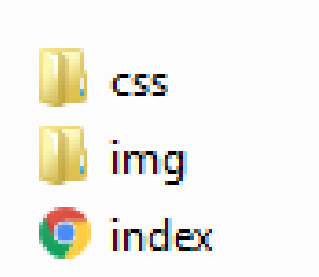

Рис. 2.12 — Файловая структура проекта

Далее открываем файл с именем *index.html* и сформируем разметку страницы:

```
<!DOCTYPE html>
<html>
<head>
    <title>сайт посвященный машинам</title>
<link rel="stylesheet" type="text/css" href="css/style.css">
   <meta charset="UTF-8">
</head>
<body>
</body>
</html>
```

Для того, чтобы применить таблицу стилей к *HTML*-документу, подключаем через тег <***link***> файл. *style.css* расположенный в директории ***css***.Пример подключения внешнего файла**:**

```
    <link rel="stylesheet" type="text/css"
href="css/style.css">
```

Для реализации блочной верстки сайта в тег <***body***> добавим код:

```
<div id="container">
   <div id="header">
      <h2>header (шапка сайта)</h2>
   </div>
   <!--меню сайта-->
   <div id="mainmenu">
   </div>
   <!--Конец меню сайта-->
<!-- Область Контента -->
   <div id="content">
      <h2>контент</h2>
   </div>
<!--конец области контента-->
   <div id="clear">
   </div>
```

```
<!--Подвал сайта-->
   <div id="footer">
      <h2>footer (подвал)</h2>
   </div>
</div>
```

Согласно макету изображённого на рисунке 2.11, с помощью тега <***div***> ,были созданы 5 обёрточных элементов для подключения стилей оформления сайта:

- **Container** – данный элемент предназначен для формирования общего контейнера для всех других элементов разметки страницы;
- **Header** – предназначен для формирования шапки сайты. Здесь можно вставлять, например баннер сайта с контактами фирмы;
- **Mainmenu** – В этом элементе размещается меню для перехода на другие страницы сайта;
- **Content** – область контента, в которую помещается информативная часть, например, описания машин, либо их изображения;
- **Footer** – так называемый подвал. Данный элемент предназначен для указания либо контактной информации, либо указания время «**жизни**» сайта и кто является его автором.

Среди параметров некоторых тегов появился параметр *“id=”*. Это один из параметров, позволяющих использовать стили.

В папке *css* создадим файл с именем *style.css.*

Для реализации контейнера в которым будет располагаться вся вёрстка сайта, добавим в файл *style.css* следующим код **:**

```
#container {
    margin: auto auto;
    text-align: center;
    width: 50%;
    height: 455px;
    border: solid;
}
```

С помощью свойства ***width*** задаем размеры сайта по ширине. Ширина блока будет составлять 50% от размера страницы браузера. Свойство ***height*** задает высоту элемента, которая будет составлять 455 пикселей. Свойством ***border*** задаем рамку.

Для формирования шапки сайта в файле стилей вставим следующий код:

```
#header {
    background: #ebeef5;
    width: 100%;
    height: 100px;
}
```

В ***id = header***, с помощью свойств ***width*** и ***height*** выставляется ширина и высота блока относительно *id* с именем ***container***.

Строка **`width: 100%`** означает, что данный блок будет занимать по ширине 100% места блока ***container***.

Следующим этапом будет написание стилей для меню сайта.

Для формирования меню в теги ***div id = mainmenu*** пропишем через тег списка <***ul***> ссылки на страницы:

```
<div id="mainmenu">
   <ul>
      <!-- Описание ссылок в меню и сами ссылки. -->
      <li><a href="#">Новости</a></li>
      <li><a href="#">Марки машин  </a></li>
      <li><a href="#">Контанты </a></li>
   </ul>
</div>
```

В файле стилей пропишем следующий код для меню:

```
#mainmenu {
    background: #1a77ff;
    position: relative;
    overflow: hidden;
    height: 40px;
    margin: 30px 0;
}
#mainmenu ul li {
    position: relative;
    left: -50%;
    float: left;
    margin: 0 10px;
    height: 40px
}
#mainmenu ul li a {
    color: #fff;
    display: block;
    text-decoration: none;
    padding: 0 15px;
    line-h eight: 40px;
}
```

С помощью селектора #mainmenu , создается внешний контейнер меню.В таблице 2.2 представлены свойства использованные для построения меню сайта.

Таблица 2.2 — Значение свойств меню

Название свойства	Назначение свойства
position	указывает, как элемент позиционируется в документе. Значение *relative* позволяет позиционировать элемент в соответствии с

Название свойства	Назначение свойства
	нормальным потоком документа
overflow	Свойство ***overflow*** управляет отображением содержания блочного элемента, если оно целиком не помещается и выходит за область заданных размеров. Значение ***hidden*** позволяет отображать только область внутри элемента, остальное будет скрыто.
height	позволяет выставить высоту элемента
margin	Устанавливает величину отступа от каждого края элемента. Отступом является пространство от границы текущего элемента до внутренней границы его родительского элемента .Если число значений установлено 2 то , первое значение устанавливает отступ от верхнего и нижнего края, второе — от левого и правого.
left	Для позиционированного элемента определяет расстояние от левого края родительского элемента, не включая отступ, поле и ширину рамки, до левого края дочернего элемента.
float	Определяет, по какой стороне будет выравниваться элемент, при этом остальные элементы будут обтекать его с других сторон. Когда значение свойства ***float*** равно ***none***, Элемент показывается как блочный. Применение этого значения для встроенных элементов, например тега <***span***>, заставляет его вести подобно блокам — происходит перенос строк в начале и в конце содержимого
display	Многоцелевое свойство, которое определяет, как элемент должен быть показан в документе. Значение ***block*** Временно удаляет элемент из документа. Занимаемое им место не резервируется и веб-страница формируется так, словно элемента и не было. Изменить значение и сделать вновь видимым элемент можно с помощью скриптов, обращаясь к свойствам через объектную модель. В этом случае происходит переформатирование данных на странице с учетом вновь добавленного элемента.
text-decoration	Добавляет оформление текста в виде его подчеркивания, перечеркивания, линии над текстом и мигания. Одновременно можно применить более одного стиля, перечисляя значения через пробел. Значение ***none*** отменяет все эффекты, в том числе и подчеркивания у ссылок, которое задано по умолчанию.
line-h eight	Устанавливает интерлиньяж (межстрочный интервал) текста, отсчет ведется от базовой линии шрифта. При обычных обстоятельствах расстояние между строками зависит от вида и размера шрифта и определяется браузером автоматически. Отрицательное значение межстрочного расстояния не допускается.
background	Устанавливает фон элемента
color	Устанавливает цвет элемента

Для того что бы, при наведение курсора по ссылки она подсвечивалась, пропишем следующий код:

```
#mainmenu ul li a:hover {
    background-color: #666;
}
```

Теперь перейдем в область контента. В файл стилей добавим следующий код:

```
#content {
    width: 100%;
    height: 280px;
    font-family: Arial, sans-serif;
    font-size: 14px;
}
```

С помощью свойства ***width*** , растянем область контента на 100% относительно контейнера страницы сайта. Свойством ***height*** выставим высоту блока контента. Для оформления текста используем свойства ***font-family font-size***,где прописываем названием шрифтов и размер текста.

Последим этапом, будет добавление стилей оформление блока ***footer***(подвал). Для этого в файле стилей для ***footer*** добавим следующий код:

```
#footer {
    background: #33ffcc;
    width: 100%;
    height: 40px;
}
```

После добавления всех стилей и логических блоков в теги <***body***>, при запуске страницы *index.html* в браузере, появится следующая веб-страница (Рис 2.13).

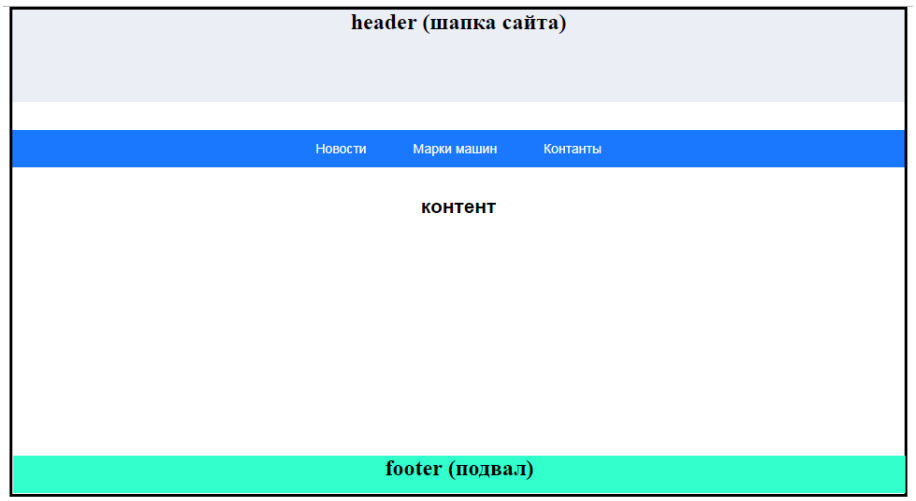

Рис. 2.13 — макет разработанного сайта

2.4. Порядок выполнения работы

2.4.1. Создать сайт, используя *CSS*, состоящий из нескольких веб - страниц согласно варианту, который выдается преподавателем. В таблице 2.3 представлены варианты заданий:

Таблица 2.3 — варианты заданий

№ варианта	Организация, для которой выполняется проектирование учебного сайта
1	магазин;
2	ресторан;
3	санаторий;
4	заказ такси;
5	гостиница;
6	поликлиника;
7	офис по продаже недвижимости;
8	бюро по трудоустройству и профориентации
9	атлетический клуб
10	больница (стационар)
11	автомобильные грузоперевозки
12	продажа железнодорожных билетов
13	малое производственное предприятие;
14	деканат ВУЗа
15	автосервис
16	туристическая фирма
17	парикмахерская
18	троллейбусный парк (организация движения)
19	библиотека

2.4.2. Оформить отчет.

2.5. Содержание отчета

2.5.1. Сформулировать цель работы.
2.5.2. Привести постановку задачи.
2.5.3. Привести краткие теоретические сведения о технологии *CSS*, и описать основные элементы разметки.
2.5.4. Привести макет разработанного сайта.
2.5.5. Привести тексты программ**.**
2.5.6. Составить таблицу, содержащею свойства *CSS* стилей использованные для создания верстки документа(см. таблица 2.4)

Таблица 2.4 — имя и назначение тегов

Имя свойства	Назначения свойства

2.5.7. Привести скриншоты выполнения программы.

2.5.8. Выводы.

2.6. Контрольные вопросы

2.6.1. Что такое *CSS*? Зачем он нужен?
2.6.2. Способы реализации (подключения) *CSS*.Область видимости.
2.6.3. Селекторы, типы селекторов, специфичность.
2.6.4. Классы и идентификаторы.
2.6.5. Структура правила. Группа стилей. Пример.
2.6.6. Какие единицы измерения в *CSS* есть? Примеры.
2.6.7. Блочная модель документа.
2.6.8. Каким образом можно задать шрифт текста в CSS? Пример кода.
2.6.9. Каким образом можно задать фон в *CSS*? Пример кода.

3. ЛАБОРАТОРНАЯ РАБОТА №3

ИСПОЛЬЗОВАНИЕ ФРЕЙМВОРКА *BOOTSTRAP* ДЛЯ РАЗРАБОТКИ ДИЗАЙНА ВЕБ – САЙТА.

3.1. Цель работы

Освоить навыки создания адаптивного макета и стилизации сайта с помощью фреймворка *Bootstrap*.

3.2. Теоретические сведения

3.2.1. Основные понятия *Bootstrap*

Bootstrap - это открытый и бесплатный *HTML*, *CSS* и *JS* фреймворк, который используется веб-разработчиками для **быстрого создания адаптивных дизайнов сайтов[4]**.

Фреймворк *Bootstrap* используется не только независимыми разработчиками, но и целыми компаниями. Основная область его применения – это разработка фронтенд составляющих сайтов и интерфейсов админок. Среди аналогичных систем (*Foundation, UIkit, Semantic UI, InK* и др.) фреймворк *Bootstrap* является самым популярным.

Bootstrap - это просто набор файлов (*CSS* и *JavaScript*). После подключения этих файлов к странице станет доступны для верстки дизайна большое количество классов и готовых компонентов. Используя их можно очень быстро и качественно создать современный адаптивный дизайн сайта.

Классы *Bootstrap* можно разбить на 3 большие группы:

- Классы для создания сетки (адаптивного макета страницы);
- Классы для стилизации контента (текста, кода, изображений, таблиц и другой информации);
- Служебные классы (для решения наиболее часто встречающихся вспомогательных задач, таких как выравнивание, управление отображением, добавление границ и др.).

Кроме классов во фреймворке *Bootstrap* имеются ещё и **компоненты** (готовые объекты интерфейса). Это кнопки, хлебные крошки, формы, навигационные меню, выпадающие списки, всплывающие панели и др.

Применение фреймворка *Bootstrap* при создании сайтов даёт следующие преимущества:

- Очень быстрое создание качественных адаптивных дизайнов сайтов (достигается благодаря использованию хорошо продуманных и протестированных огромным количеством веб-разработчиков классов и готовых компонентов).

— Современный дизайн (оформление *HTML* элементов и компонентов *Bootstrap* выполнено в едином стиле в последних тенденциях веб-дизайна).
— Нет необходимости иметь глубокие знания по *HTML, CSS, JavaScript* и *jQuery* (достаточно знать только основы вышеперечисленных технологий).
— Является кроссбраузерным и кроссплатформенным (адаптирован для всех популярных операционных систем и браузеров (*Mozilla Firefox, Google Chrome, Safari, Internet Explorer* и *Opera* и др.), работающих в этих системах).
— Является открытым и бесплатным. Фреймворк *Bootstrap* – это проект с открытым исходным кодом, доступным на *Github*. Он имеет лицензию *MIT*. Это означает, что его можно использовать бесплатно как для личного, так и для коммерческого использования.

Но, кроме преимуществ у *Bootstrap* есть также недостатки. Первый заключается в том, что он не подходит для проектов с уникальным дизайном**.** В этом случае из *Bootstrap* для проекта в основном берут только сетку. Второй – это когда вам для проекта нужны не все компоненты и классы, а только некоторые**.** Этот недостаток, не является проблемой. Например, на странице *Customize* можно очень просто собрать свою сборку, состоящую только из нужных классов и компонентов *Bootstrap*[4].

3.2.2. Распаковка архива *Bootstrap.*

После скачивания архива (с готовыми к применению *CSS* и *JavaScript* файлами), его необходимо распаковать в каталог веб-проекта.

Если рассмотреть архив, то можно заметить, что он имеет следующее содержимое (на примере *Bootstrap* 3.4.0):

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.min.css
│   ├── bootstrap-theme.css
│   └── bootstrap-theme.min.css
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
```

└── *glyphicons-halflings-regular.woff*

В каталоге *css* находятся стили фреймворка *Bootstrap*, а в *js* - плагины для обеспечения работы некоторых компонентов. Плагины написаны с использованием функций библиотеки *jQuery*. Поэтому перед *Bootstrap JS* необходимо подключить библиотеку *jQuery*.
Как вы можете заметить, в архиве есть 2 версии *CSS* и *JavaScript* файлов, т.е. с суффиксом *min* и без него. Версия файла **с** *min* ничем ни отличается от без *min*, она просто минимизирована (сжата).
В продакшене (на рабочем сайте) лучше использовать минимизированные версии файлов. Эти файлы имеют меньший размер, и, следовательно, обеспечивают более быструю загрузку страниц сайта. Не минимизированные версии более удобно использовать при разработке, а также для изучения.
Кроме этих файлов, в данный архив ещё входит иконочный шрифт "*Glyphicons*". Шрифт "*Glyphicons*" насчитывает более 250 иконок из набора "*Glyphicon Halflings*". Шрифт представлен с помощью 4 файлов*: glyphicons-halflings-regular.eot, glyphicons-halflings-regular.svg, glyphicons-halflings-regular.ttf, glyphicons-halflings-regular.woff).*

3.2.3. Подключение *Bootstrap* к *HTML* странице.

Процесс установки фреймворка *Bootstrap* 3 состоит из подключения следующих файлов к HTML странице:

1. *Bootstrap CSS* (*bootstrap.min.css*);
2. Последней версии библиотеки *jQuery* (необходима для работы *JS* плагинов *Bootstrap*);
3. Bootstrap *JavaScript* (*bootstrap.min.js).*
4. Файлы *JavaScript* лучше подключать перед закрывающим тегом ***</body>***, так как это обеспечит более быструю загрузку и отображение основного контента веб-страницы.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <!-- Кодировка веб-страницы -->
    <meta charset="utf-8">
    <!-- Настройка viewport -->
    <meta name="viewport" content="width=device-width, ini-
tial-scale=1, shrink-to-fit=no">
    <!-- Подключаем Bootstrap CSS -->
    <link rel="stylesheet" href="css/bootstrap.min.css" >
</head>
<body>
    <!-- Контент страницы -->
```

```
    <!-- Подключаем jQuery -->
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery
.min.js"></script>
    <!-- Подключаем Bootstrap JS -->
    <script src="js/bootstrap.min.js"></script>
</body>
</html>
```

3.2.4. «Строительные» элементы сетки Bootstrap

Сетка *Bootstrap* состоит из «строительных» элементов. Основными элементами сетки *Bootstrap* 3 и 4 являются:

- обёрточные контейнеры - элементы с классом ***.container*** или ***.container-fluid***;
- ряды - элемент с классом ***.row***;
- адаптивные блоки - элементы, с одним или несколькими классами ***col***.
- По существу сетка - это просто список предопределённых классов, с помощью которых можно задать необходимое поведение блокам (HTML элементам) и построить с их помощью определённый адаптивный макет сайта.

3.2.5. Обёрточные контейнеры

Обёрточный контейнер - это «строительный» элемент сетки, с которого начинается создание макета для всей страницы или её части (например, шапки, основного меню, основной области, футера) в зависимости от стратегии разработки.

Обёрточный контейнер - это первый элемент, с которого начинается создание макета страницы или некоторой его самостоятельной части. Его основное назначение - это установить ширину разрабатываемого макета. В *Bootstrap* 3 и 4 обёрточные контейнеры бывают 2 типов. Первый (***container***) предназначен для создания адаптивно-фиксированного макета, а второй (***container-fluid***) - для адаптивно-резинового (адаптивно-гибкого) макета[4].

Адаптивно-фиксированный макет характеризуется тем, что он имеет условно постоянную ширину, которая на одних диазонах *viewport* браузера имеет одно значение, а на дугих - другое.

Например, в *Bootstrap* 3 определено 4 диапазона (контрольные точки): ***xs*** (по умолчанию), ***sm*** (ширина *viewport* больше 768px), ***md*** (ширина *viewport* больше 992px), ***lg*** (ширина *viewport* больше 1200px).

Обёрточный контейнер (**container**) устанавливает макету:

— на диапазоне ***xs*** ширину, равную ширине *viewport* браузера;
— на диапазоне ***sm***, ширину равную 750px;
— на диапазоне ***md***, ширину равную 970px;
— на диапазоне **lg**, ширину равную 1170px.

Ширина же адаптивно-резинового макета не имеет фиксированного значения, она всегда равна ширине *vieport* браузера.

Контейнер в *Bootstrap* бывает адаптивно-фиксированным или адаптивно-резиновым.

Первый (адаптивно-фиксированный контейнер) характеризуется тем, что он имеет постоянную ширину в пределах некоторого диапазона ширины *viewport* (области просмотра).

В таблице 3.1 приведено то, какую ширину имеет адаптивно-фиксированный контейнер при той или иной ширине области просмотра (*viewport*) браузера:

Таблица 3.1 — ширина адаптивно – фиксированного контейнера

Ширина viewport	**Ширина контейнера (*container*)**
<768px	Ширина контейнера равна ширине *viewport*
>= 768px и <992px	750px
>= 992px и <1200px	970px
<=1200px	1170px

Кроме установки ширины адаптивно-фиксированный контейнер *Bootstrap* ещё центрирует себя в горизонтальном направлении относительно краёв страницы. Выполняет это он с помощью *CSS* свойств ***margin-left:auto*** и ***margin-right:auto***. Ещё адаптивно-фиксированный контейнер задает внутренние отступы слева и справа по 15px (с помощью *CSS* свойств ***padding-left:15px*** и ***padding-right:15px***) для содержимого, которое в него помещено.

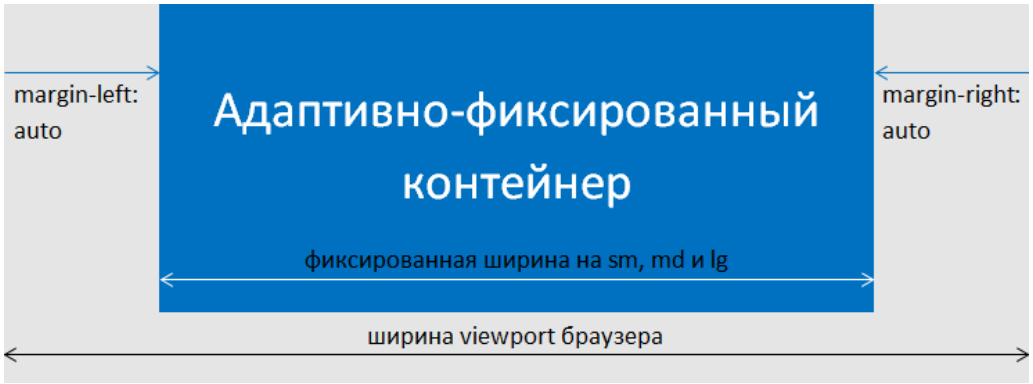

Рис 3.1 — Адаптивно-фиксированный контейнер

```
<!-- HTML-код адаптивно-фиксированного контейнера -->
<div class="container">
  ...
```

```
</div>
```

Адаптивно-резиновый контейнер отличается от адаптивно-фиксированного тем, что он занимает всю ширину (100%) окна браузера. Кроме этого, он также как и адаптивно-фиксированный контейнер задает внутренние отступы слева и справа по 15px для содержимого, которое в него помещено.

```
<!-- HTML-код адаптивно-резинового контейнера -->
<div class="container-fluid">
</div>
```

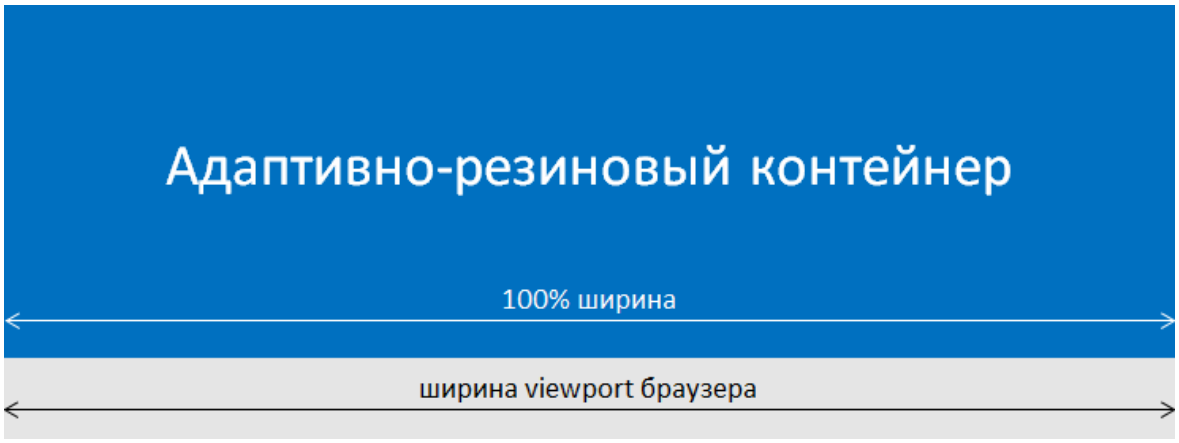

Рис 3.2 — Адаптивно – резиновый контейнер

Первый «строительный» элемент сетки *Bootstrap* – это обёрточный контейнер. Он определяет ширину макета на различных *viewport*, а также выполняет его центрирование (только адаптивно-фиксированный контейнер) относительно левого и правого края рабочей области вкладки или окна браузера.

Следующий строительный элемент – это ряд(блок *div* с классом *row*). Ряд - это специальный блок, который применяется только для оборачивания других строительных элементов (адаптивных блоков). Его основное назначение – это нейтрализация положительного внутреннего отступа (*15px* слева и справа) обёрточного контейнера или адаптивного блока.

Bootstrap - ряд, расположенный внутри адаптивно-фиксированного контейнера

Рис. 3.3 — иллюстрация работы класса row

Пример формирования внутренних отступов:

container (+*15px*) -> *row* (-*15px*) -> *col* (+*15px*) -> контент

container (+*15px*) -> *row* (-*15px*) -> *col* (+*15px*) -> *row* (-*15px*) -> *col* (+*15px*) -> контент

**col* - это адаптивный блок.

Пример показывает, что независимо от того в каком адаптивном блоке находится контент, он всегда будет иметь правильный отступ от края (т.е. 15px слева и справа).

Следующий «строительный» элемент сетки *Bootstrap* – это адаптивный блок(*div* с классом *col-?-?*).

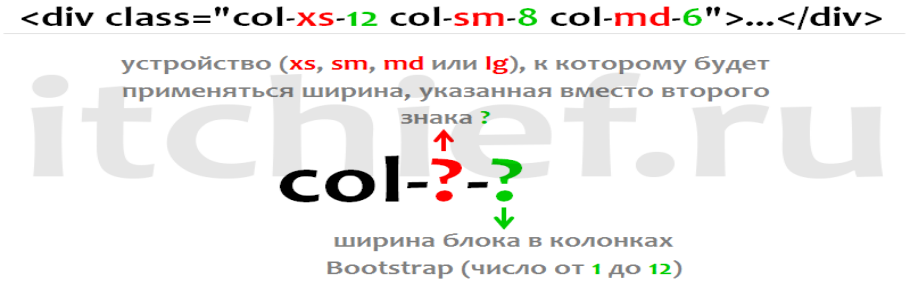

Рис. 3.4 — Адаптивный блок *Bootstrap*

Адаптивные блоки - это элементы сетки *Bootstrap*, которым установлен один или несколько классов *col-?-?*. Данные блоки являются основными «строительными» кирпичиками, именно они и формируют необходимую структуру.

Ширина адаптивному блоку задаётся в связке с типом устройства. Это означает, что адаптивный блок на разных устройствах может иметь разную ширину. Именно из-за этого данный блок и называется адаптивным.

Установка ширины адаптивному блоку, которую он должен иметь на определённом устройстве, задаётся по умолчанию числом от 1 до 12

(количеством колонок *Bootstrap*). Данное число указывается вместо второго знака? в классе *col-?-?*.

Данное число (по умолчанию от 1 до 12) определяет какой процент от ширины родительского элемента должен иметь адаптивный блок.

Например, число 1 - устанавливает адаптивному блоку ширину, равную 8,3% (1/12) от ширины родительского блока. Число 12 - ширину, равную 100% (12/12) от ширины родительского блока.

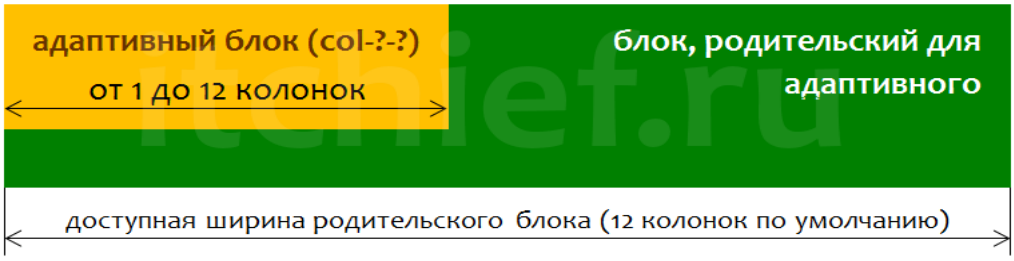

Рис. 3.4 — установка ширины адаптивного блока

Кроме указания ширины адаптивному блоку необходимо ещё указать и тип устройства (вместо 1 вопроса). Класс устройства будет определять то, на каком *viewport* будет действовать эта ширина. В *Bootstrap* 3 различают 4 основных класса. Это *xs* (ширина *viewport* >0) ,*sm* (ширина *viewport* >= 768px), *md* (ширина *viewport* >= 992px) и lg (ширина области просмотра браузера >=1200px).

Например, адаптивный блок с классом *col-xs*-12 *col-sm*-6 col-*md*-4 *col-lg*-3 будет иметь на устройстве xs ширину 100% (12/12), на *sm* - 50%(6/12), на *md* -33,3% (4/12), на *lg* - 25% (3/12).

Например, адаптивный блок с классом col-xs-6 col-lg-3 будет иметь на устройствах *xs*, *sm* и *md* ширину 50% (6/12), на *lg* - 25% (3/12).

Адаптивный блок, также как и контейнер, задаёт положительные отступы слева и справа по 15px для контента, который в него будет помещён.

Например, разобьём блок на 3 равные колонки, которые на *xs* будут отображаться вертикально, а на *sm* и выше горизонтально:

```
<div class="row">
  <div class="col-xs-12 col-sm-4">...</div>
  <div class="col-xs-12 col-sm-4">...</div>
  <div class="col-xs-12 col-sm-4">...</div>
</div>
```

Если необходимо рассчитать ширину колонок в пикселях, то это правильно делать на границах контрольных точек (т.е. когда одно действие класса сменяется другим).

Например, рассчитаем, доступную ширину под контент для первого и второго блока:

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

```
  <div class="row">
    <div class="col-xs-12 col-md-8">1 блок...</div>
    <div class="col-xs-12 col-md-4">2 блок...</div>
  </div>
</div>
```

3.2.6. Создание форм в *Bootstrap*

Платформа *Bootstrap* 3 содержит глобальные стили и классы, которые предназначены для оформления *HTML* форм и индивидуальных элементов управления[4].

Глобальные стили представляют собой определённые *CSS* правила, которые определяют внешний вид элементов управления на веб-странице. Эти стили элементы управления получают автоматически, и веб-разработчику их явно задавать не требуется.

В *Twitter Bootstrap* 3 основная задача для веб-разработчика в основном сводится в добавлении необходимых классов для элементов управления, форм и контейнеров.

Основные моменты при создании и оформлении формы представим в виде следующих этапов:

— Указать вид формы. В *Bootstrap* 3 различают следующие виды форм: вертикальная (без добавления класса), горизонтальная ***(.form-horizontal***) и в одну строку **(*.form-inline***);

— Добавить к необходимым текстовым элементам управления ***<input>,<textarea>, <select>*** класс ***.form-control***, чтобы установить им ширину, равную 100% (всю доступную ширину родительского элемента);

— Поместить каждую надпись ***(<label>)*** и элемент управления в контейнер *** ... *** с классом ***.form-group***. Это необходимо сделать, чтобы задать для элементов в форме оптимальные отступы.

3.2.7. Вертикальная форма (по умолчанию)

Вертикальная форма - это макет формы, в которой её элементы располагаются вертикально, т.е. один под другим. Данная форма создаётся без добавления класса к элементу формы (<form>). Метки и элементы управления формы необходимо размещать в блоке с классом ***.form-group***.

```
<form>
<div class="form-group">
<label for="inputEmail">Адрес email:</label>
<input type="email" class="form-control" id="inputEmail"
placeholder="Введите email">
  </div>
  <div class="form-group">
```

```
    <label for="inputPassword">Пароль:</label>
    <input type="password" class="form-control"
id="inputPassword" placeholder="Введите пароль">
  </div>
  <div class="checkbox">
    <label>
      <input type="checkbox"> Запомнить
    </label>
  </div>
  <button type="submit" class="btn btn-default">Войти</button>
</form>
```

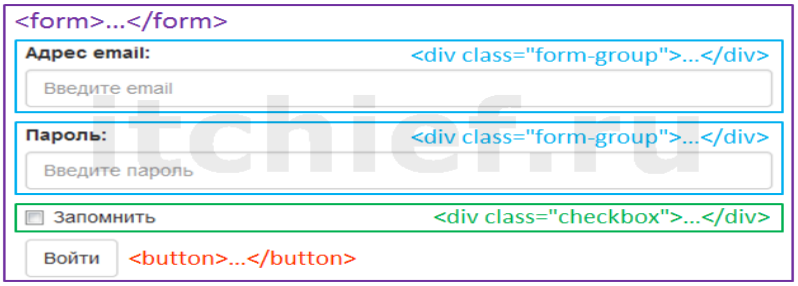

Рис. 3.5 - Создание формы

3.2.8. Горизонтальная форма (*.form-horizontal*)

Горизонтальная форма - это форма, в которой надписи и элементы управления находящиеся в одной группе (<***div class="form-group"***>***...</div***>) располагаются на одной строке.

Принцип создания горизонтальной формы:

— Добавить класс ***.form-horizontal*** к элементу ***form;***

— Поместить элементы формы, которые должны располагаться в одной строке, в контейнер <***div***>***...</div***> и добавить к нему класс ***.form-grou****p;*

— Кроме этого этим элементам, т.е. тем которые будут располагаться в одной строке, необходимо задать ширину с помощью классов системы сетки *Twitter Bootstrap* 3;

— Добавить класс ***.control-label*** к элементам ***label***.

```
<form class="form-horizontal">
  <div class="form-group">
    <label for="inputEmail" class="col-xs-2 control-
label">Адрес email:</label>
    <div class="col-xs-10">
      <input type="email" class="form-control" id="inputEmail"
placeholder="Введите email">
    </div>
  </div>
```

```
  <div class="form-group">
    <label for="inputPassword" class="col-xs-2 control-
label">Пароль:</label>
    <div class="col-xs-10">
      <input type="password" class="form-control"
id="inputPassword" placeholder="Введите пароль">
    </div>
  </div>
  <div class="form-group">
    <div class="col-xs-offset-2 col-xs-10">
      <div class="checkbox">
        <label><input type="checkbox"> Запомнить</label>
      </div>
    </div>
  </div>
  <div class="form-group">
    <div class="col-xs-offset-2 col-xs-10">
      <button type="submit" class="btn btn-
default">Войти</button>
    </div>
  </div>
</form>
```

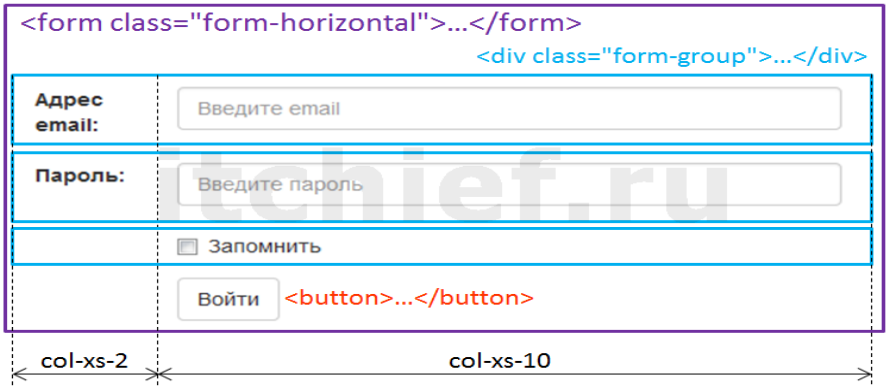

Рис. 3.6 - Создание формы

Создание формы, с расположением элементов в одну строку. Для создание формы с расположением элементов в одну строку, необходимо добавить *Bootstrap* класс ***.form-inline*** к элементу <***form***>. Такие формы можно создавать только для окон шириной не менее 768px.

```
<form class="form-inline">
  <div class="form-group">
    <label class="sr-only" for="inputEmail">Email</label>
    <input type="email" class="form-control" id="inputEmail"
placeholder="Email">
  </div>
  <div class="form-group">
    <label class="sr-only" for="inputPassword">Пароль</label>
```

```
    <input type="password" class="form-control"
id="inputPassword" placeholder="Пароль">
  </div>
  <div class="checkbox">
    <label><input type="checkbox"> Запомнить</label>
  </div>
  <button type="submit" class="btn btn-primary">Войти</button>
</form>
```

Пример отображения формы представлен на рис 3.7.

Рис. 3.7 — пример формы в одну строку

Если нужно поместить обычный текст рядом с элементом <***label***> в горизонтальной форме, то используется класс ***.form-control-static*** на элементе <***p***>.

```
<form class="form-horizontal">
  <div class="form-group">
    <label for="inputEmail" class="control-label col-xs-
2">Email</label>
    <div class="col-xs-10">
      <p class="form-control-static">admin@itchief.ru</p>
    </div>
  </div>
  <div class="form-group">
    <label for="inputPassword" class="control-label col-xs-
2">Пароль</label>
    <div class="col-xs-10">
      <input type="password" class="form-control"
id="inputPassword" placeholder="Пароль">
    </div>
  </div>
  <div class="form-group">
    <div class="col-xs-offset-2 col-xs-10">
      <div class="checkbox">
        <label><input type="checkbox"> Запомнить</label>
      </div>
    </div>
  </div>
  <div class="form-group">
    <div class="col-xs-offset-2 col-xs-10">
      <button type="submit" class="btn btn-
primary">Войти</button>
    </div>
  </div>
</form>
```

Пример отображения формы представлен на рис 3.8.

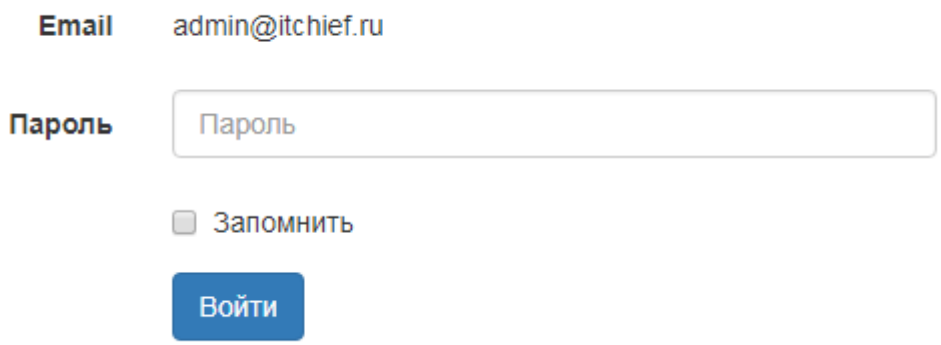

Рис. 3.8 — Пример отображения формы

3.2.9. Изменение высоты элементов <***input***> и <***select***>

С помощью классов *Bootstrap* ***.input-lg*** и ***.input-sm*** можно контролировать высоту элементов <***input***> и <***select***>**.**

```
<form>
  <div class="row">
    <div class="col-xs-6">
      <input type="text" class="form-control input-lg" place-
holder="Большой">
    </div>
    <div class="col-xs-6">
      <select class="form-control input-lg">
        <option>Большой - 1</option>
        <option>Большой - 2</option>
      </select>
    </div>
  </div>
  <br />
  <div class="row">
    <div class="col-xs-6">
      <input type="text" class="form-control"
placeholder="Средний">
    </div>
    <div class="col-xs-6">
      <select class="form-control">
        <option>Средний - 1</option>
        <option>Средний - 2</option>
      </select>
    </div>
  </div>
  <br />
  <div class="row">
    <div class="col-xs-6">
```

```
      <input type="text" class="form-control input-sm" place-
holder="Маленький">
    </div>
    <div class="col-xs-6">
      <select class="form-control input-sm">
        <option>Маленький - 1</option>
        <option>Маленький - 2</option>
      </select>
    </div>
  </div>
</form>
```

Пример отображения формы представлен на рис 3.9.

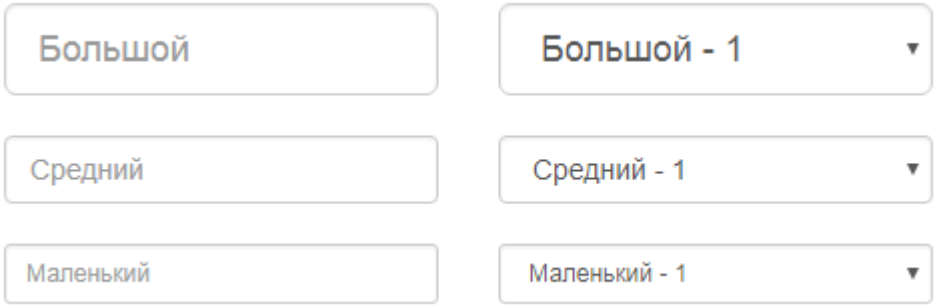

Рис. 3.9 — Изменение высоты элементов формы

3.2.10. Оформление таблиц с помощью *CSS* классов *Bootstrap*

Для придания таблицы базового оформления необходимо добавить класс ***.table*** к элементу *table*:

```
<table class="table">
...
</table>
Например:
<table class="table">
  <thead>
<tr>
<th>№ п/п</th>
<th>Имя</th>
<th>Фамилия</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
    <tr>
      <td>1</td>
      <td>Иван</td>
      <td>Чмель</td>
      <td>ivan@mail.ru</td>
    </tr>
```

```
    <tr>
      <td>2</td>
      <td>Петр</td>
      <td>Щербаков</td>
      <td>petr@mail.ru</td>
    </tr>
    <tr>
      <td>3</td>
      <td>Юрий</td>
      <td>Голов</td>
      <td>yuri@mail.ru</td>
    </tr>
  </tbody>
</table>
```

№ п/п	Имя	Фамилия	E-mail
1	Иван	Чмель	ivan@mail.ru
2	Петр	Щербаков	petr@mail.ru
3	Юрий	Голов	yuri@mail.ru

Рис. 3.10 — Оформление таблицы при помощи *Bootstrap*

Для выделения нечётных строк основной части таблицы (***<tbody>...</tbody>***) с помощью тёмного фона добавьте дополнительно класс ***.table-striped*** к классу ***.table***.

```
<table class="table table-striped">
 </table>
Например:
<table class="table table-striped">
  <thead>
    <tr>
      <th>№ п/п</th>
      <th>Имя</th>
      <th>Фамилия</th>
      <th>E-mail</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>1</td>
      <td>Иван</td>
      <td>Чмель</td>
      <td>ivan@mail.ru</td>
    </tr>
    <tr>
```

```
      <td>2</td>
      <td>Петр</td>
      <td>Щербаков</td>
      <td>petr@mail.ru</td>
    </tr>
    <tr>
      <td>3</td>
      <td>Юрий</td>
      <td>Голов</td>
      <td>yuri@mail.ru</td>
    </tr>
  </tbody>
</table>
```

№ п/п	Имя	Фамилия	E-mail
1	Иван	Чмель	ivan@mail.ru
2	Петр	Щербаков	petr@mail.ru
3	Юрий	Голов	yuri@mail.ru

Рис. 3.11 — пример оформления таблицы средствами *Bootstrap*

3.3. Ход выполнения работы

Используя макет сайта с проектируемого в 2 лабораторной работы, разработаем макет сайта с помощью фреймворка *Bootstrap*.

Создадим файл *index.html*, в котором будет размещена стандартная html разметка документа:

```
<!DOCTYPE html>
<html>
<head>
    <title>сайт посвященный машинам</title>
   <link rel="stylesheet" type="text/css" href="css/style.css">
   <meta charset="UTF-8">
</head>
<body>
</body>
</html>
```

Для использования возможностей фреймворка, необходимо скачать его с официального сайта (*http://bootstrap-3.ru/index.php*).После скачивания и разархивирования файла, необходимо перенести в папку **css** разрабатываемого проекта файлы *Bootstrap*. Далее необходимо подключить файл стилей в секции <***head***> файла *index.html*:

```
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
<link rel="stylesheet" href="css/bootstrap.css">
```

Для создания адаптивного дизайна средствами фреймворка *Bootstrap*, в тег <***body***> добавим следующий код:

```
<div class="container">
   <div class="row">
      <div class="col-lg-12" >
         шапка сайта
      </div>
      <div class="col-lg-12">
         <!--меню сайта-->
…
         <!—Конец меню сайта-->
      </div>
      <div class="col-lg-12" >
      <!--Область контента-->
…
      <!--Конец Области контента-->
      </div>
      <div class="col-lg-12">
         подвал
      </div>
   </div>
```

Обёрточный контейнер .***container***- это первый элемент, с которого начинается создание макета страницы или некоторой его самостоятельной части. Его основное назначение - это установить ширину разрабатываемого макета. Класс ***container*** предназначен для создания адаптивно-фиксированного макета. Обёрточный контейнер кроме установления ширины макету ещё выравнивает его по центру страницы и задаёт внутренние поля (***padding***) слева и справа по 15*px*. Следующим элементов который необходимо использовать при разработке макета являются ряд (***row***).

Ряд ***row***- это тоже контейнер, но для адаптивных блоков сетки *Bootstrap*.

В *Bootstrap* 3 его основная роль - это создать отрицательные отступы (***margin***) слева и справа по 15px.

Принцип использования элемента "*ряд*" очень прост, он всегда должен выступать родителем для адаптивных блоков. Т.е. если какой-то элемент (обёрточный контейнер или адаптивный блок) необходимо разметить с помощью адаптивных блоков, то перед тем как их создать сначала установите ряд, а уже в нём эти блоки.

Адаптивный блок - это элемент, который имеют адаптивную ширину. Т.е. его ширина на одном диапазоне *viewport* может иметь одно значение, а на другом - другое.

Установка поведения адаптивного блока осуществляется с помощью одного или нескольких классов ***col***. Для разработки макета применялся блок ***col-lg*-12**, данный элемент позволяет растянуть блок на всю ширину контейнера.

Для создания меню используем стили примененные во 2 лабораторной работе. Для этого создаем файл *style.css*, который будет находиться в папке ***css*** проекта и имеет следующий вид:

```
    #mainmenu {
    background: #1a77ff;
    position: relative;
    overflow: hidden;
    height: 40px;
    margin: 30px 0
#mainmenu ul li {
    position: relative;
    left: -50%;
    float: left;
    margin: 0 10px;
    height: 40px
}
#mainmenu ul li a {
    color: #fff;
    display: block;
    text-decoration: none;
    padding: 0 15px;
    line-height: 40px;
}
#mainmenu ul li a:hover {
    background-color: #666;
}
```

В файле *index.html*, в секции меню, прописываем следующий код, который создает ссылки на гипертекст:

```
<!--меню сайта-->
      <div id="mainmenu">
         <ul>
            <!-- Описание ссылок в меню и сами ссылки. -->
            <li><a href="#">Новости</a></li>
            <li><a href="#">Марки машин  </a></li>
            <li><a href="#">Контанты </a></li>
<li><a href="form.html">гостевая книга </a></li>
         </ul>
      </div>
```

Следующим этапом необходимо создать форму гостевой книги, имеющею 3 поля и кнопку для отправки данных. Для этого создаем файл с именем

form.html файла и в секцию <***body***> добавим следующий код реализующий форму:

```
<form >
     <div class="form-group">
       <label>Имя</label>
       <input type="text" class="form-control"
placeholder="Введите имя "name="name">
     </div>
     <div class="form-group">
       <label>EMAIL:</label>
       <input type="text" class="form-control"
placeholder="Введите Email" name="email">
     </div>
     <div class="form-group">
       <div class="form-group">
       <label>Сообщение</label>
          <textarea name="comment" class="form-control" place-
holder="Сообщение" name="text"></textarea>
      </div>

      <div class="form-group">
         <input type="submit" class="btn btn-info"
value="Отправить" />
      </div>
</form>
```

Парный тег <***form***> является контейнером для создания формы, в нем будет располагаться элементы формы. С помощью тега <***label***> создаем лейбу поля, т.е. уточняем таким образом, что это будет за поле. Для создания текстового поля используем тег <***input***> с атрибутом ***type="text".***Атрибут *placeholder* необходим ,чтобы подсказать пользователю, что необходимо вводить в данное поле.
Для отправки данных на сервер предназначена специальная кнопка ***Submit***. Её вид ничем не отличается от обычных кнопок, но при нажатии на нее происходит выполнение серверной программы, указанной атрибутом ***action*** тега <***form***>. Эта программа, называемая еще обработчиком формы, получает данные, введенные пользователем в полях формы, производит с ними необходимые манипуляции, после чего возвращает результат в виде *HTML*-документа.

3.4. Порядок выполнения лабораторной работы

3.4.1. Согласно варианту задания, выданный преподавателем во 2 лабораторной работе, необходимо сверстать адаптивный шаблон используя фреймворк *Bootstrap*.

3.4.2. Создать гостевую книгу состоящею из 3 полей: имя, имейл и сообщение. Создать кнопу *submit* для передачи данных на сервер. Используя классы фреймворка *Bootstrap* выполнить стилизацию формы.

3.4.3 Создать таблицу состоящую из 4 столбцов и к ней применить стили фреймворка *Bootstrap*.

3.5. Содержание отчета

3.5.1. Сформулировать цель работы.

3.5.2. Привести постановку задачи.

3.5.3. Привести краткие теоретические сведения о технологии Bootstrap, и описать основные элементы разметки.

3.5.4. Привести структуру разработанного сайта.

3.5.5. Привести тексты программ**.**

3.5.6. Составить таблицу, содержащею классы *Bootstrap* использованные для создания верстки и стилей документа (см. таблица 3.2)

Таблица 3.2 — классы и назначение

Имя класса	**Назначения класса**

3.5.7. Привести скриншоты выполнения программы.

3.5.8. Выводы.

3.6. Контрольные вопросы

3.6.1. Что такое *Bootstrap*?

3.6.2. Каким образом происходит подключение *Bootstrap* к *HTML* странице?

3.6.3. Назовите основные элементы сетки в *Bootstrap*

3.6.4. Что такое адаптивная верстка сайта? Каким образом создается в *Bootstrap*?

3.6.5. Что такое адаптивно резиновая верстка сайта? Каким образом создается в *Bootstrap*?

3.6.6. Что такое адаптивный блок? Пример адаптивного блока в *Bootstrap*.

3.6.7. Что такое форма? Каким образом создается?

3.6.8. Каким образом с помощью *Bootstrap* можно форме добавить стили?

3.6.9. Какие стили с помощью *Bootstrap* можно добавить таблице?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Харрис, Э. PHP/MySQL для начинающих / Э. Харрис. – М.: Кудиц Образ, 2007. – 384 с.
2. Веллинг, Л. Разработка веб-приложений с помощью PHP и MySQL / Л. Веллинг, Л. Томсон. – СПб. : Вильямс, 2010. – 848 с. – ISBN 978- 5-8459-1574-0.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, Javascript и CSS / Р. Никсон. – СПб. : Питер, 2013. – 560 с. – ISBN 978-5-496-00187-8.
4. Bootstrap в примерах. / Пер. с англ. Рагимов Р. Н. / Науч. ред. Киселев А. Н. – М.: ДМК Пресс, 2017. – 314 с.: ил.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт радиоэлектроники и информационной безопасности
Кафедра «Радиоэлектроника и телекоммуникации»

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

по дисциплине «Коммуникации в сети интернет и принципы построения сайтов»

для

студентов очной формы обучения по направлениям

общеуниверситетского пула

2 часть

«Создание динамических страниц на стороне сервера»

Севастополь
2020

УДК 621.396
Учебно-методическое пособие Коммуникации в сети Интернет и принципы построения сайтов 2 часть [Электронный ресурс] / СевГУ; сост. Д. И. Табакаев. — Севастополь : Изд-во СевГУ, 2020 — 34с.

Цель пособия: Целью учебно-методического пособия является оказание помощи студентам очной формы обучения при выполнении лабораторных работ по дисциплине «Коммуникации в сети Интернет и принципы построения сайтов».

Учебно-методическое пособие утверждено на заседании кафедры «Радиоэлектроника и телекоммуникации» от 23.03.20 г., протокол № 11.

Учебно-методическое пособие рассмотрено и рекомендовано к изданию на заседании методической комиссии Института радиоэлектроники и информационной безопасности от 24.04.20 протокол №9.

Рецензент д. т. н., профессор **И.Л. Афонин**
Редактор: к. т. н., доцент **В. Г. Слёзкин**

Ответственный за выпуск:
заведующий кафедрой «Радиоэлектроника и телекоммуникации»
д. т. н., профессор **И. Л. Афонин**

Издательский номер: №128/2020

Оглавление

ВВЕДЕНИЕ

В последние два десятилетия Интернет прочно вошел в нашу жизнь. В наше время он охватывает почти все сферы жизни: культура, искусство, бизнес. Наряду с обычными музеями, библиотеками, магазинами все большую популярность во всем мире завоевывают аналогичные Интернет-ресурсы. Даже бизнес становится виртуальным: создаются *Web*-представительства фирм, электронные магазины и торговые площадки, электронные банки, которые дают возможность осуществлять типичные для этих сфер операции с большей скоростью. В связи с этим профессия *Web*-дизайнера и *Web*-программиста стала востребованной и у нас в России. Интернет-технологии создания указанных *web*-ресурсов требует владения такими языками для написания серверных скриптов, как *Perl, PHP, ASP*, владения языком *XML*, а также умения создавать динамические страницы и сайты, используя скрипты *VBScript* и *JavaScript*, которые позволяют осуществлять вычисления, работу с датой и временем, изменение эле-ментов страницы по желанию пользователя.

На следующем этапе необходимо научиться создавать простые *дин*амические страницы с помощью *PHP*. При передаче информации через Интернет используются как раз *HTML*-страницы – файлы *.htm и *.html, поскольку они содержат в себе гипертекст, имеют размер меньше, чем текстовые или иные файлы и для их просмотра ну-жен только браузер.

Данное учебно-методическое пособие призвано помочь в освоении базовых знаний и получении практических навыков в области серверного программирования на языке высокого уровня *PHP*. Представляемые две лабораторные работы выполняются студентами направлений общеуниверситетского полу в рамках рабочей программы по дисциплине «Коммуникации в сети Интернет и принципы построения сайтов». Все работы выполняются и тестируются в рамках локальной вычислительной сети с использованием учебного веб-сервера. Перед выполнением каждой работы обязательным является ознакомление с учебно-методическим пособием.

4. ЛАБОРАТОРНАЯ РАБОТА №4

СОЗДАНИЕ ДИНАМИЧЕСКИХ СТРАНИЦ СРЕДСТВАМИ *PHP*

4.1. Цель работы

Освоить навыки построения динамических Веб-сайтов с использованием языки программирования *PHP*.

4.2. Теоретические сведения

4.2.1. Описание языка *PHP*

PHP – это широко используемый язык сценариев общего назначения с открытым исходным кодом.

PHP это язык программирования, специально разработанный для написания *web*-приложений (сценариев), исполняющихся на *Web*-сервере.

Аббревиатура *PHP* означает "*Hypertext Preprocessor* (*Препроцессор Гипертекста*)". Синтаксис языка берет начало из *C, Java* и *Perl*. *PHP* достаточно прост для изучения. Преимуществом *PHP* является предоставление *web*-разработчикам возможности быстрого создания динамически генерируемых *web*-страниц[1].

Важным преимуществом языка *PHP* перед такими языками, как языков *Perl* и *C* заключается в возможности создания *HTML* документов с внедренными командами *PHP*.

Значительным отличием *PHP* от какого-либо кода, выполняющегося на стороне клиента, например, *JavaScript*, является то, что *PHP*-скрипты выполняются на стороне сервера. Можно сконфигурировать свой сервер таким образом, чтобы *HTML*-файлы обрабатывались процессором *PHP*, так что клиенты даже не смогут узнать, получают ли они обычный *HTML*-файл или результат выполнения скрипта.

PHP позволяет создавать качественные *Web*-приложения за очень короткие сроки, получая продукты, легко модифицируемые и поддерживаемые в будущем.

PHP прост для освоения, и вместе с тем способен удовлетворить запросы профессиональных программистов.

PHP-код может находиться в любой части документа и может быть включен в документ одним из двух способов:

<?php **PHP-код** **?>**	или	**<?** **PHP-код** **?>**

Основные правила написания программы на *PHP*:

- в именах переменных и функций большие и малые буквы *различаются*;
- в конце каждого оператора (команды) указывается точка с запятой (;);
- специального символа переноса оператора на другую строчку нет – переносить можно с любой позиции в которой можно указать пробел (слова не разрываются при переносе оператора).

4.2.2. Статические и динамические сайты

Статические сайты состоят из неизменяемых страниц. Это значит, что сайт имеет один и тот же внешний вид, а также одно и то же наполнение для всех посетителей. При запросе такого сайта в браузере сервер сразу предоставляет готовый *HTML*-документ в исходном виде, в котором он и был создан. Кроме *HTML*, в коде таких страниц используется разве что *CSS* и *JavaScript*, что обеспечивает их легкость и быструю загрузку.

Чаще всего статическими бывают сайты с минимальным количеством страниц или с контентом, который не нужно регулярно обновлять, а именно **сайты-визитки**, **каталоги продукции**, **справочники технической документации**. Однако с помощью сторонних инструментов существует возможность добавить на такие страницы отдельные динамические элементы (комментарии, личный кабинет для пользователей, поиск).

Динамические сайты, в свою очередь, имеют изменяемые страницы, адаптирующиеся под конкретного пользователя. Такие страницы не размещены на сервере в готовом виде, а собираются заново по каждому новому запросу. Сначала сервер находит нужный документ и отправляет его интерпретатору, который выполняет код из *HTML*-документа и сверяется с файлами и базой данных. После этого документ возвращается на сервер и затем отображается в браузере[1]. Для интерпретации страниц на серверной стороне используются языки программирования *Java, PHP, ASP* и другие.

Самыми яркими примерами динамических сайтов являются страницы, созданные на основе систем управления контентом (*CMS*). Среди них чаще всего встречаются **интернет-магазины**, а также **форумы**, **страницы с отзывами** и другие ресурсы с возможностью размещения контента посетителями.

4.2.3. Функции вывода

В *PHP* существует несколько способов вывода информации на страницу. Операторы **echo** и **print** выводят значение аргумента:

— **echo <строка>**; **print (<строка>);**Работают одинаково, но **print** всегда возвращает 1, в то время как ***echo*** ничего не возвращает;

— **var_dump** (<переменная>); - выводит переменную вместе с типом.

Замечания:

Строковое выражение может быть взято как в одинарные, так и в двойные кавычки.

Если используются двойные кавычки, то вместо имен переменных будут выводиться их значения.

Пример вывода данных с помощью операторов **echo** и **print:**

```
echo " Hello world";
print("Привет мир!");
print "print() можно использовать и без скобок.";
```

Пример вывода данных с помощью **var_dump**:

```
Var_dump($name);
```

4.2.4. Методы передачи данных *GET* и *POST*

Все переменные, приходящие в скрипт от браузера, в данном случаи через *URL*, или с сервера являются внешними переменными.

Скрипт - любая программа написанная, для обработки переменных. Так как язык *PHP* серверный язык программирования, следовательно, все скрипты находятся на сервере, т.е. на сайте в папке и никуда не отправляются (например, браузеру). Скрипт выполняет запрос браузера на представление страницы, он просто собирает, формирует страницу *HTML* кода и отправляет её в готовом виде пользователю, который запросил её с помощью своего браузера[2].

Существуют два способа, с помощью которых клиенты-браузеры могут отправлять информацию на веб-сервер — это метод *GET* и метод *POST*.

Прежде чем браузер отправит информацию, он кодирует ее, используя схему, называемую кодировкой *URL*-адресов. В этой схеме пары имя / значение объединяются знаком равно, а разные пары разделяются амперсандом: **name1=value1&name2=value2&name3=value3**.

GET - это название запроса, который отправляется на сервер скрипту с помощью браузера открыто, через *URL*, адресную строку.

Если в адресной строке увидели знак амперсанды (**&**) и знак вопрос (**?**), можно считать, что этот узел работает на *PHP*, и ему в данный момент отправлены переменные и их значения.

Метод *GET* создает длинную строку, которая отображает в логах сервера и в адресной строке браузера.

Метод *GET* предназначен для отправки только до 1024 символов.

метод *GET*, не используется если нужно отправить на сервер пароль или другую конфиденциальную информацию.

GET не может использоваться для отправки на сервер двоичных данных, таких как изображения или текстовые документы.

Доступ к данным, отправленным через метод *GET*, можно получить с помощью переменной среды *QUERY_STRING*.

PHP предоставляет ассоциативный массив **$_GET** для доступа ко всей информации, отправляемой с использованием метода GET.

```
$_GET['имя_ переменной'];
```

POST — используется для передачи больших объемов данных или когда нужно скрыть значения передаваемых параметров. Переданные сценарию параметры не отображаются в окне браузера

Метод *POST* передает информацию через *HTTP*-заголовки. Информация кодируется, как описано в случае метода *GET*, и помещается в заголовок *QUERY_STRING* **.**

Метод **POST** не имеет ограничений по объему отправляемых данных.

Метод **POST** может использоваться для отправки *ASCII*, а также двоичных данных.

Данные, отправленные методом *POST*, проходят через *HTTP*-заголовок, поэтому их безопасность зависит от протокола *HTTP*. Используя *Secure HTTP*, вы можете обеспечить защиту информации.

PHP предоставляет ассоциативный массив **$_POST** для доступа ко всей информации отправляемой с помощью метода *POST*. Пример доступа к данным через метод POST.

```
$_POST['имя_ переменной'];
```

4.2.5. Сообщения об ошибках в *PHP*

Перед исполнением кода интерпретатор *PHP* проверяет скрипт на наличие ошибок различного уровня. Если *PHP* обнаруживает ошибки, соответствующие установленным уровням, то, в зависимости от настроек конфигурации, *PHP* генерирует соответствующие сообщения, которые записываются в переменные и (или) выводятся в браузер пользователя. В таблице 4.1 представлены уровни ошибок[2]:

Таблица 4.1 — уровни ошибок в *PHP*

Константа	Описание
E_ERROR	Фатальные ошибки времени выполнения. Указывает на ошибки, которые не могут быть устранены, такие как проблемы выделения памяти. Выполнение скрипта останавливается.

Константа	Описание
E_WARNING	Предупреждения времени выполнения (нефатальные ошибки). Выполнение скрипта не останавливается.
E_PARSE	Ошибки разбора при компиляции. Ошибки разбора должны генерироваться только разборщиком.
E_NOTICE	Уведомления времени выполнения. При работе скрипта возникло нечто, что может указывать на ошибку, но может также появиться и при нормальном выполнении скрипта.
E_CORE_ERROR	Фатальные ошибки, возникающие при начальном старте *PHP*. Напоминает **E_ERROR**, но генерируется ядром *PHP*.
E_CORE_WARNING	Предупреждения (нефатальные ошибки), возникающие при начальном старте *PHP*. Напоминает **E_WARNING**, но генерируется ядром *PHP*.
E_COMPILE_ERROR	Фатальные ошибки компиляции. Напоминает **E_ERROR**, но генерируется машиной Zend Scripting Engine.
E_COMPILE_WARNING	Предупреждения времени компиляции (нефатальные ошибки). Напоминает **E_WARNING**, но генерируется машиной Zend Scripting Engine.
E_USER_ERROR	Генерируемое пользователем сообщение об ошибке. Напоминает **E_ERROR**, но генерируется в PHP-коде путём использования PHP-функции **trigger_error()**.
E_USER_WARNING	Генерируемое пользователем предупреждение. Напоминает **E_WARNING**, но генерируется в PHP-коде путём использования PHP-функции **trigger_error()**.
E_USER_NOTICE	Генерируемое пользователем уведомление. Напоминает **E_NOTICE**, но генерируется в PHP-коде путём использования PHP-функции **trigger_error()**.

Константа	Описание
E_ALL	Все ошибки и предупреждения, если поддерживаются, за исключением уровня **E_STRICT**.
E_STRICT	Примечания во время выполнения. Включите, чтобы *PHP*, предлагал замены вашему коду, который будет гарантировать лучшую функциональную совместимость и совместимость вашего кода с предыдущими версиями PHP.

4.2.6. Коды состояния *HTTP*

Код состояния *HTTP* (англ. *HTTP status code*) — часть первой строки ответа сервера при запросах по протоколу *HTTP*. Он представляет собой целое число из трёх десятичных цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа. Клиент может не знать все коды состояния, но он обязан отреагировать в соответствии с классом кода[3]. В настоящее время выделено пять классов кодов состояния:

— Информационные (100-105);
— Успешные (200-226);
— Перенаправление (300-307);
— Ошибка клиента (400-499);
— Ошибка сервера (500-510).

В таблице 4.2 представлены основные ошибки клиента и сервера.

Таблица 4.2 — Код состояния *HTTP* клиента и сервера.

Client error responses		
400	**Bad Request**	"Плохой запрос". Этот ответ означает, что сервер не понимает запрос из-за неверного синтаксиса.
401	**Unauthorized**	"Неавторизовано". Для получения запрашиваемого ответа нужна аутентификация. Статус похож на статус 403, но,в этом случае, аутентификация возможна.
402	**Payment Required**	"Необходима оплата". Этот код ответа зарезервирован для будущего использования. Первоначальная цель для создания этого когда была в использовании его для цифровых платежных систем(на данный момент не используется).

403	**Forbidden**	"Запрещено". У клиента нет прав доступа к содержимому, поэтому сервер отказывается дать надлежащий ответ.
404	**Not Found**	"Не найден". Сервер не может найти запрашиваемый ресурс. Код этого ответа, наверно, самый известный из-за частоты его появления в вебе.
405	**Method Not Allowed**	"Метод не разрешен". Сервер знает о запрашиваемом методе, но он был деактивирован и не может быть использован. Два обязательных метода, GET и HEAD, никогда не должны быть деактивированы и не должны возвращать этот код ошибки.
	Server error responses	
500	**Internal Server Error**	"Внутренняя ошибка сервера". Сервер столкнулся с ситуацией, которую он не знает как обработать.
501	**Not Implemented**	"Не выполнено". Метод запроса не поддерживается сервером и не может быть обработан. Единственные методы, которые сервера должны поддерживать (и, соответственно, не должны возвращать этот код) - GET и HEAD.
502	**Bad Gateway**	"Плохой шлюз". Эта ошибка означает что сервер, во время работы в качестве шлюза для получения ответа, нужного для обработки запроса, получил недействительный(недопустимый) ответ.
503	**Service Unavailable**	"Сервис недоступен". Сервер не готов обрабатывать запрос. Зачастую причинами являются отключение сервера или то, что он перегружен. Обратите внимание, что вместе с этим ответом удобная для пользователей(user-friendly) страница должна отправлять объяснение проблемы. Этот ответ должен использоваться для временных условий и Retry-After: HTTP-заголовок должен, если возможно, содержать предполагаемое время до восстановления сервиса. Веб-мастер также должен позаботиться о заголовках, связанных с кэшем, которые отправляются вместе с этим ответом, так как эти ответы, связанные с временными условиями, обычно не должны кэшироваться.

504	**Gateway Timeout**	Этот ответ об ошибке предоставляется, когда сервер действует как шлюз и не может получить ответ вовремя.
505	**HTTP Version Not Supported**	"HTTP-версия не поддерживается". HTTP-версия, используемая в запросе, не поддерживается сервером.

4.2.7. Подключение файла в *PHP*

Чтобы сделать программный код более удобочитаемым, можно поместить определения функций и/или классов в отдельный файл. Возможность подключения файлов в *PHP* обеспечивают четыре языковые инструкции:

— **Include;**
— **Require;**
— **include_once;**
— **require_once;**

Синтаксис подключения файлов через инструкции выглядит следующим образом:

```
Имя_инструкции 'имя_файла.php';
```

Например:

```
include 'test.php';
```

Все четыре инструкции могут принимать в качестве параметра имя локального файла. Инструкция **include** и **require** очень схожи по действию и отличаются лишь реакцией на невозможность получения запрошенного ресурса. Например, в случае недоступности ресурса **include** и **include_once** выводят предупреждение и пытаются продолжить исполнение программы. **require** и **require_once** при недоступности запрошенного ресурса останавливают обработку данной страницы.

Инструкция **include** позволяет подключать и присоединять к *PHP*-сценарию другие сценарии. При запуске программы интерпретатор просто заменит инструкцию на содержимое подключаемого файла.

Инструкция **include** присоединяет содержимое подключаемого файла, благодаря этому программа получает доступ к другим переменным, функциям, классам и т.д.

Файлы подключаются исходя из указанного пути к файлу, если путь не указан, будет использоваться путь, который указан в директиве **include_path** (в конфигурационном файле **php.ini**). Если файл не найден по указанному пути в **include_path**, инструкция **include** попытается проверить текущую рабочую директорию, в которой находится скрипт

подключающий файл, если конструкция **include** не сможет найти файл, будет выдано предупреждение (**warning**).

Если путь указан - не важно какой: абсолютный или относительный (относительно текущей директории, где расположен включающий сценарий) - директива **include_path** будет проигнорирована.

Поведение **include_once** идентично инструкции **include**, с той лишь разницей, что, если код из файла уже был один раз подключен, он не будет подключен и выполнен повторно. Это В результате будет получено сообщение об ошибке, так как функции переопределять нельзя. Чтобы избежать ошибок подобного рода, следует использовать инструкцию **include_once.**

Инструкции **require и require_once** работают идентично **include** и **include_once** за исключением лишь одной особенности. Если подключаемый файл не будет найден, выполнение скрипта будет остановлено, в то время как **include** и **include_once** выводят предупреждение и продолжают выполнение скрипта.

4.2.8. Суперглобальные переменные

При передачи данных формы на сервер формируются суперглобальные переменные — ассоциативные массивы, содержащие передаваемые значения:

- **$_GET[]** – ассоциативный массив, который содержит все значения, передаваемые в сценарий с помощью метода формы *GET*.
- **$_POST[]** – ассоциативный массив, который содержит все значения, передаваемые в сценарий с помощью метода формы *POST*.
- **$_REQUEST[]** – ассоциативный массив, который содержит все значение, передаваемые в сценарий с помощью методов *POST* и *GET*.

4.3. Ход выполнения работы

Используя разметку и стили разработанного сайта в 3 лабораторной работе расмотрим создания динамического сайта на *PHP*.Структурная схема разробатываего сайта представлены на рисунке 4.1.

Шапка страницы
Меню сайта

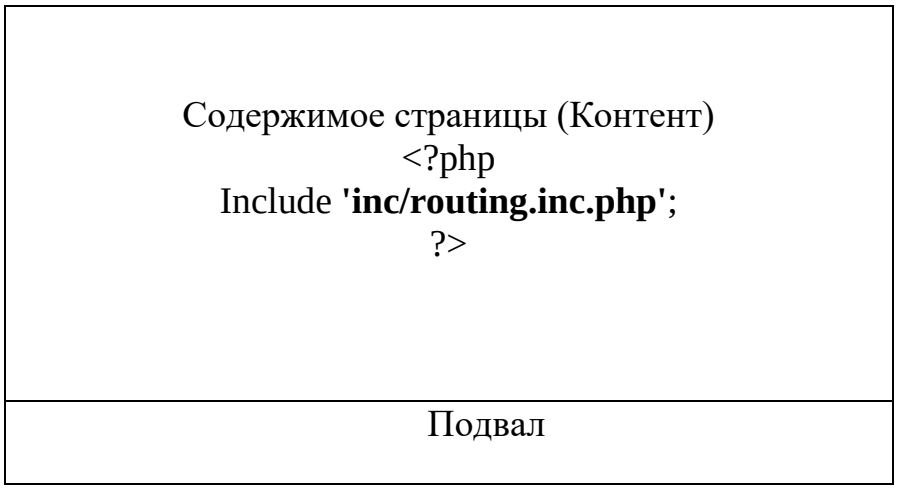

Рис. 4.1 — структура создаваемого сайта

Файловая структура разрабатываемого сайта представлена на рисунке 4.2.

Создадим файл *index.php***,** в который добавим следующий код**:**

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, ini-
tial-scale=1.0">
  <title>Bootstrap сайт</title>
  <link rel="stylesheet" href="css/bootstrap.css">
  <link rel="stylesheet" href="css/font-awesome.min.css">
  <link rel="stylesheet" href="css/style.css" />
   </head>

   <body>
<div class="container">
   <div class="row">
      <div class="col-lg-12" >
         шапка сайта
      </div>
      <div class="col-lg-12 no-margin">
         <!--меню сайта-->
         <div id="mainmenu">
         <ul>
            <!-- Описание ссылок в меню и сами ссылки. -->
            <li><a href='index.php'>Домой</a></li>
            <li><a
            href='index.php?id=contact'>Контакты</a></li>
            <li><a href='index.php?id=about'>О нас</a></li>
            <li><a href='index.php?id=gbook'>Гостевая
книга</a></li>
         </ul>
```

```
      </div>

      </div>
      <div class="col-lg-12"  >
      <!--Область контента-->
      <?php
            include 'routing.php';
                  ?>

      <!--Конец Области контента-->
      </div>
      <div class="col-lg-12">
         подвал
      </div>

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

Файл *index.php* – файл который является точкой входа на сайт, то есть через него будет происходить отображение всех данных и переход на другие ссылки. Для осуществления перехода на другие ссылки в атрибуте **href** тега **<a>** указывается, что через файл *index.php* будут передаваться данные с помощью пары **имя = значение (**см. пункт 4.2.4.**)**. Пример кода:

```
<a href='index.php?id=contact'>Контакты</a>
```

Знак вопроса означает, что передается данные с именем **id,** где значение данной перемены равно **contact.**
В области основного контента с помощью инструкции **include** подключается файл с именем *routing.php***.**

```
include 'routing.php';
```

Файл *routing.php* за динамическое подключение файлов и отображения в области контента данных
В корне проекта создадим файл *routing.php* и запишем следующий код:

```
<?php
$id = null;
// Получение id страницы с помощью метода GET и фильтрация
данных
$id = strtolower(strip_tags(trim($_GET['id'])));
//Проверка на содержимое переменной $id
switch($id){
   case 'contact': include 'inc/contact.inc.php';  break;
   case 'about':   include 'inc/about.inc.php';    break;
   case 'gbook':   include 'inc/gbook.inc.php';    break;
```

```
   default:        include 'inc/index.inc.php';
}
}
?>
```

С помощью суперглобального массива данных **$_GET**, происходит получение данных *id* страницы.

Для корректного получения данных через суперглобальный массив **$_GET**, необходимо отфильтровать полученные данные, то есть отделить мусор от данных которые может вести пользователь, например, пробелы, *html* – теги и т.д. для это используются следующие функции:

- Функция **strtolower()** преобразует строку в нижний регистр данные полученные через **$_GET**;
- Функция **strip_tags()** - убирает *html* теги из строки *PHP;*
- Функция **trim()** – убирает пробелы в начале и в конце строки.

Для организации маршрутизации файлов используется оператор множественного выбора **switch**.

Switch является наиболее удобным средством для организации мультиветвления. Синтаксис оператора ветвления следующий:

```
switch(Вырожение) // выражение которое будем проверять
{
   case value1: // константное выражение 1
      statements; // блок операторов
   break;
   case value2: // константное выражение 2
      statements;
   break;
   default:
      statements;
}
```

На вход оператора **switch** передается переменная **$id,** содержащая значение **id** страницы на которую пользователь хочет перейти. С помощью оператора **case**, происходит проверка, какое значение содержится в переменной **$id**. Если пользователь перешел на страницу **Контакты (id = contact)** в область контента файла *index.php* происходит динамическое подключение через инструкцию **include** содержимого файла *contact.inc.php***.**

И так происходит для каждой страницы. Оператор **default** необходим если не один из **case** операторов не выполнится и программе нужно сообщить, что ей дальше делать.

Следующим этапом, будет создание страниц области контента. Для этого в корне проекта создадим папку **inc**, а в ней создадим следующие файлы:

- *index.inc.php* – файл контента главной страницы сайта;

— *contact.inc.php* – файл содержат контактную информацию;
— *about.inc.php* – файл содержит информацию о компании;
— *gbook.inc.php* – гостевая книга.

В файл *gbook.inc.php*, добавим код для создания гостевой книги:

```
<!--ФОРМА ГОСТЕВОЙ КНИГИ-->
<div class="col-lg-10">
   <form>
        <div class="form-group">

          <label>Имя</label>
          <input type="text" class="form-control"  placehold-
er="Введите имя "name="name">
        </div>
        <div class="form-group">
          <label>EMAIL:</label>
          <input type="text" class="form-control"  placehold-
er="Введите Email" name="email">
        </div>
        <div class="form-group">
          <div class="form-group">
          <label>Сообщение</label>
             <textarea name="comment" class="form-control"
placeholder="Сообщение" name="text"></textarea>
         </div>

         <div class="form-group">
            <input type="submit" class="btn btn-info"
value="Отправить" />
         </div>
   </form>
</div>
```

Для того, чтобы запустить скрипт на *PHP* необходим веб-сервер. Веб-сервер (*сервер*, *server*, *web-server*) - отвечает за обработку запросов клиентов к веб-сайту и исполнение скриптов, таких как *CGI*, *JSP*, *ASP* и *PHP*, отвечающих за организацию запросов к базам данных и приложениям электронной коммерции (например Apache). Для размещения сайта в Интернет или на персональном компьютере необходим веб-сервер с поддержкой как минимум сервиса Apatche.

В качестве веб сервера возьмем *XAMPP*, скачать его можно с официального сайта по ссылке **https://www.apachefriends.org/ru/download.html**. После успешной установки веб - сервера на персональном компьютере на диске C:\ необходимо найти папку с именем *xampp*, в ней будет папке с именем *htdocs*. В этой папке будут храниться все *PHP* приложения. Следующим этапом будет запуск сервера *Apatche*, который будет обеспечивать работоспособность *php* скриптов. На рабочем столе после установки

появится файл с именем *XAMPP ControlPanel*. После нажатия на данный файл откроется окно(Рис 4.2).

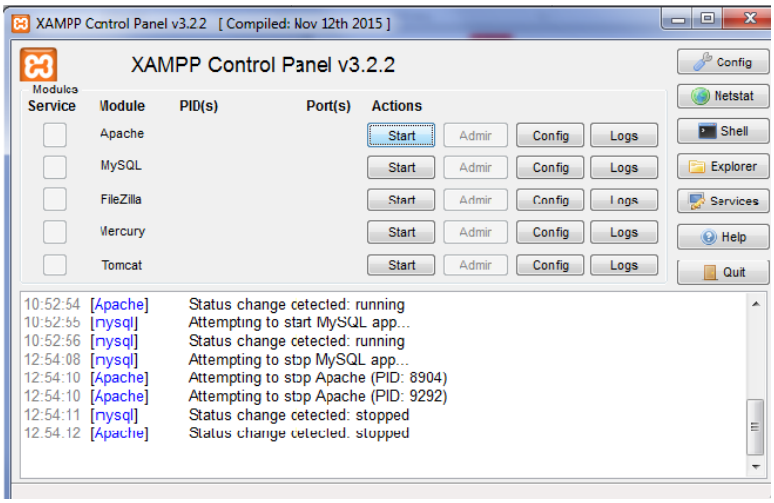

Рис. 4.2 — *XAMPP* контроль панель

Кнопкой старт *Start* запускам *Apache*.Если сервер запустился, появляется номер порта, который используется для работы *Apache* (Рис. 4.3).

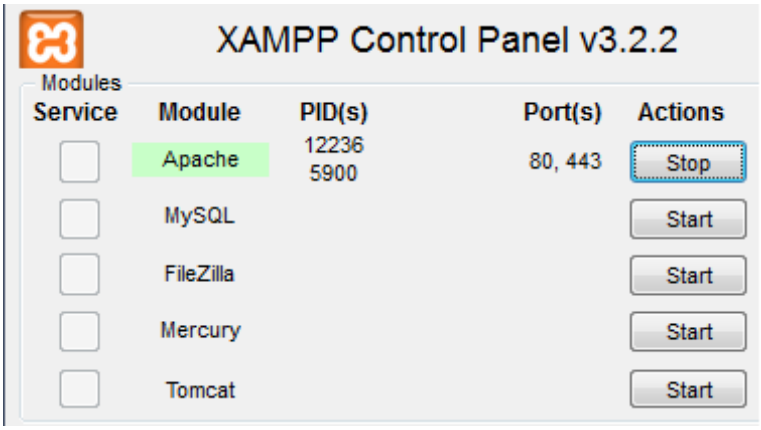

Рис. 4.3 — Запуск *Apache*

После чего открываем браузер и в адресной строке вводим:

```
localhost/имя_проекта/index.php
```

4.4. Порядок выполнения

4.4.1. Создать файловую структуру согласно рисунку 4.2.

4.4.2. Создать файл *index.php* в корне проекта. В него сохранить разработанную в 3 лабораторной работе верстку сайта.

4.4.3. Создать файл *routing.php* для маршрутизации страниц.

4.4.4. Создать следующие страницы: *index.inc.php ,contact.inc.php about.inc.php, gbook.inc.php* и заполнить их контентом.

4.4.5. Оформить отчет согласно требованиям.

4.5. Содержание отчета

4.5.1. Сформулировать цель работы.

4.5.2. Сформулировать постановку задачи и индивидуальное задание.

4.5.3. Привести краткие теоретические сведения о технологии *PHP*.

4.5.4. Привести структуру разработанного сайта.

4.5.5. Привести тексты программ**.**

4.5.6. Привести скриншоты выполнения программы.

4.5.7. Выводы.

4.6. Контрольные вопросы

4.6.1. Что такое *PHP*?

4.6.2. Что такое динамические страницы?

4.6.3. Синтаксис язык *PHP*.

4.6.4. Функции вывода данных.

4.6.5. Какие коды состояния *HTPP* существуют?

4.6.6. Какие уровни ошибок существуют в *PHP*?

4.6.7. Операторы ветвления программы *if, switch*. Синтаксис, описание работы операторов.

4.6.8. Каким образом происходит получение *id* страницы, на которую пользователь переходит?

4.6.9. Что такое веб-сервер, для чего нужен? Что входит в состав веб-сервера? Как запустить скрипт на *PHP*?

4.6.10. Характеристика метода передачи данных *GET*.Пример обработки формы *GET* методом.

4.6.11. Характеристика метода передачи данных *POST*.Пример обработки формы *POST* методом.

5. ЛАБОРАТОРНАЯ РАБОТА №5
ИСПОЛЬЗОВАНИЕ БАЗ ДАННЫХ *MYSQL*.

5.1. Цель работы

Ознакомиться с принципами создания базы данных через графическую оболочку *phpmyadmin*, созданием запросов для сохранения и выборки данных из базы данных.

5.2. Теоретические сведения

5.1.1.Основные понятия о базах данных

База данных – набор сведений, хранящихся некоторым упорядоченным способом. Можно сравнить базу данных со шкафом, в котором хранятся документы. Иными словами, база данных – это хранилище данных. Сами по себе базы данных не представляли бы интереса, если бы не было систем управления базами данных (*СУБД*). *Система управления базами данных* – это совокупность языковых и программных средств, которая осуществляет доступ к данным, позволяет их создавать, менять и удалять, обеспечивает безопасность данных и т.д. В общем *СУБД* – это система, позволяющая создавать базы данных и манипулировать сведениями из них[4]. А осуществляет этот доступ к данным *СУБД* посредством специального языка – *SQL*.

SQL – язык структурированных запросов, основной задачей которого является предоставление простого способа считывания и записи информации в базу данных.

Простейшая схема работы с базой данных выглядит примерно так:

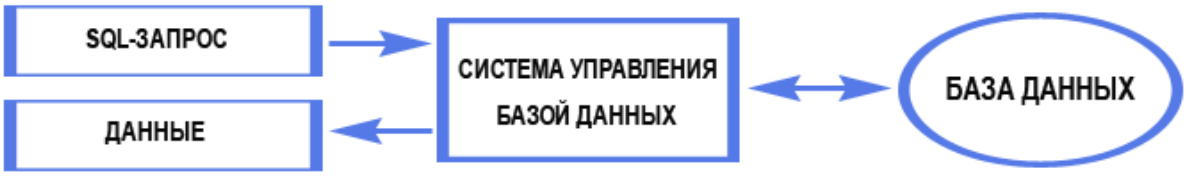

Рис 5.1 — схема работы с базой данных

5.1.2. Запросы к базе данных

Структурированный язык запросов *SQL* позволяет производить различные операции с базами данных: создавать таблицы, помещать, обновлять и удалять из них данные, производить запросы из таблиц и.т.д

Несмотря на то, что последний стандарт *SQL* принят в 1992 году, на сегодняшний день нет ни одной *СУБД*, где бы он полностью выполнялся. Более того, в различных базах данных часть операций осуществляется по-разному. Будем придерживаться диалекта *SQL* характерного для *СУБД MySQL* поэтому не все запросы могут выполняться для других баз данных[4].

SELECT – запрос, который выбирает уже существующие данные из БД. Для выбора можно указывать определённые параметры выбора. Например, суть запроса русским языком звучит так – ВЫБРАТЬ(**SELECT**) такие-то колонки(имена колонок в базе данных) ИЗ(**FROM**) такой-то таблицы(указывается имя таблицы).

```
SELECT * FROM имя_таблицы;
```

INSERT – запрос, который позволяет ПЕРВОНАЧАЛЬНО вставить запись в *БД*. То есть создаёт НОВУЮ запись (строчку) в *БД*. Делает новую запись(insert into) в таблице *users*, в поле *name* вставляет Сергей, а в поле *age* вставляет 25. Таким образом, в таблицу дописывается новая строки с данными значениями. Если колонок больше, то они оставшиеся останутся либо пустыми, либо с установленными по умолчанию значениями.

```
INSERT INTO users (name , age) VALUES('Сергей', '25' )
```

CRUD (сокр. от англ. *create*, *read*, *update*, *delete* — «создать, прочесть, обновить, удалить») — термин, обозначающий четыре базовые функции, используемые при работе с персистентными хранилищами данных:

- создание;
- чтение;
- редактирование;
- удаление.

В таблице 5.1 представлены операции с базой данных

Таблица 5.1

Операция	Оператор в языке SQL	Операция в протоколе HTTP
Создание (create)	INSERT	POST
Чтение (read)	SELECT	GET
Редактирование (update)	UPDATE	PUT
Удаление (delete)	DELETE	DELETE

5.2.3. Взаимодействие с базами данных средствами *PDO*

Для обеспечения взаимодействия базы данных с *PHP* существует библиотека *PDO*.

«*PDO – PHP Data Objects* – это прослойка, которая предлагает универсальный способ работы с несколькими базами данных.

Рис 5.2 – Схема работы PDO

Для соединения с базой данных через PDO, необходимо создать экземпляр класса *PDO*:

```
$DBH = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);
```

где -

$**host** – имя хоста (для локальной машины прописываем localhost).

$**dbname** – имя базы данных.

$**user** – имя пользователя (по умолчанию root).

$**pass** – пароль для подключения к БД(по умолчанию пусто).

Для создания запроса сохранения используется 2-х ступеньчитая система запроса, главной задачей которой является избежание создания sql-инъекции.

Рис 5.3 — Структура создания запроса

Prepared statement — это заранее скомпилированное *SQL*-выражение, которое может быть многократно выполнено путем отправки серверу лишь различных наборов данных. Дополнительным преимуществом является невозможность провести *SQL*-инъекцию через данные, используемые в *placeholder'ax*.

execute — Запускает подготовленный запрос на выполнение.

```
$STH = $DBH->prepare("INSERT INTO lab5 (name, email, text)
values (?, ?, ?)");
$result = $STH ->execute( array($name, $email, $ text) );
```

5.2.4. Создание пользовательской функции в *PHP*

В любом языке программирования существуют подпрограммы. В языке С они называются функциями, в ассемблере - подпрограммами, а в *Pascal* существуют даже два вида подпрограмм: процедуры и функции.

Подпрограмма - это специальным образом, оформленный фрагмент программы, к которому можно обратиться из любого места внутри программы. Подпрограммы существенно упрощают жизнь программистам, улучшая читабельность исходного кода, а также сокращая его, поскольку отдельные фрагменты кода не нужно писать несколько раз

В *PHP* такими подпрограммами являются пользовательские функции.

Помимо встроенных функций *PHP*, часто возникает необходимость создания пользовательских функций, выполняющих определенные задачи.

Пользовательская функция может быть объявлена в любой части программы (скрипта), до места ее первого использования. И не нужно никакого предварительного объявления, как в других языках программирования, в частности, в *C*. Преимущества применяемого в *PHP* подхода в следующем.

Дойдя до определения пользовательской функции, транслятор проверит корректность определения и выполнит трансляцию определения функции во внутреннее представление, но транслировать сам код он не будет. И это правильно - зачем транслировать код, который, возможно, вообще не будет использован. Синтаксис объявления функций следующий:

```
Function
Имя_функции(аргумент1[=значение1],.,аргумент1[=значение1])
{
// тело функции
}
Пример объявления функции:
     function filter($name)
   {
        $result = htmlspecialchars( strip_tags( trim($name)));
        return $result;
    }
```

Объявление функции начинается служебным словом **function**, затем следует имя функции, после имени функции - список аргументов в скобках. Тело функции заключается в фигурные скобки и может содержать любое количество операторов.

Требования, предъявляемые к именам функций:

- Имена функций могут содержать русские буквы, но давать функциям имена, состоящие из русских букв, не рекомендуется;
- Имена функций не должны содержать пробелов;
- Имя каждой пользовательской функции должно быть уникальным. При этом, необходимо помнить, что регистр при объявлении

функций и обращении к ним не учитывается. То есть, например, функции **funct()** и **FUNCT()** имеют одинаковые имена;

— Функциям можно давать такие же имена, как и переменным, только без знака **$** в начале имен.

Типы значений, возвращаемые пользовательскими функциями, могут быть любыми. Для передачи результата работы пользовательских функций в основную программу (скрипт) используется конструкция **return**. Если функция ничего не возвращает, конструкцию **return** не указывают. Конструкция **return** может возвращать все, что угодно, в том числе и массивы.

5.3. Ход выполнение работы

Чтобы обмениваться данными с сервером *MySQL* (сохранять, изменять, удалять, получать данные), необходима база данных. Создать базу данных можно из консоли *MySQL*, а также из визуального интерфейса *phpMyAdmin*. Для работа с СУБД нужно запустить службу *MySQL* с помощью *XAMPP CONTROL Panel* :

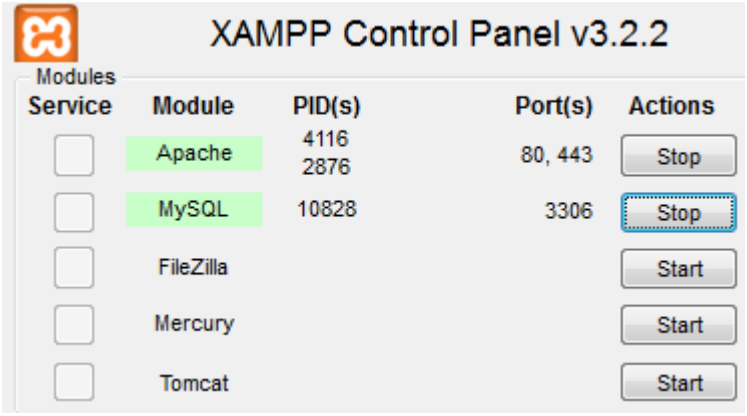

Рис. 5.4 — Запуск *MySQL* через *XAMPP*

Для входа *phpMyAdmin* необходимо в адресной строке браузера вести следующее:

```
localhost/phpmyadmin
```

После чего откроется интерфейс СУБД представленный на рисунке 5.5

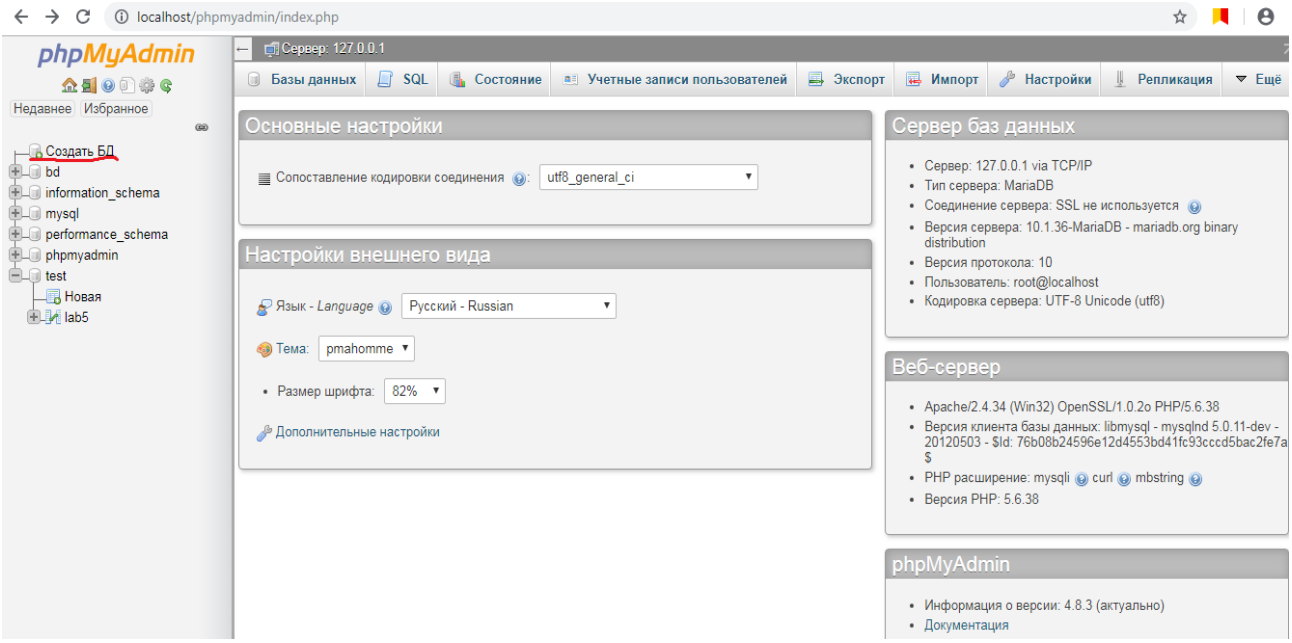

Рис. 5.5 — Интерфейс *phpMyAdmin*

Перейдем на вкладку Базы данных. Под меткой *Создать* базу данных введем какое-нибудь имя для новой бд, например, lab5 и нажмем на кнопку "Создать".

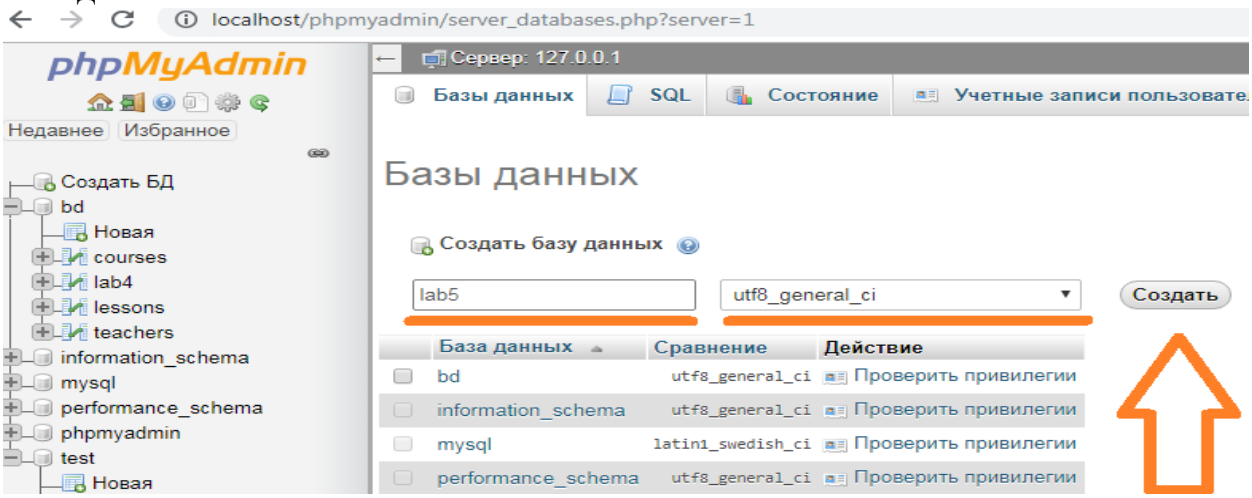

Рис 5.6 – Создание таблицы lab5

После этого мы получим сообщение об успешном создании новой бд, и она будет добавлена в списки баз данных.

Новая база данных пока пуста и не содержит ничего. Добавим в нее таблицу, которая будет хранить данные. Для этого нажмем на название базы данных и попадем на вкладку "*Структура*", где нам будут предложены опции новой таблицы. В поле "Имя" введем название новой таблицы. Пусть, таблица будет хранить данные формы гостевой книги,

поэтому введем название ***"lab5",*** а в качестве количества столбцов введем цифру 4(Рисунок 5.6):

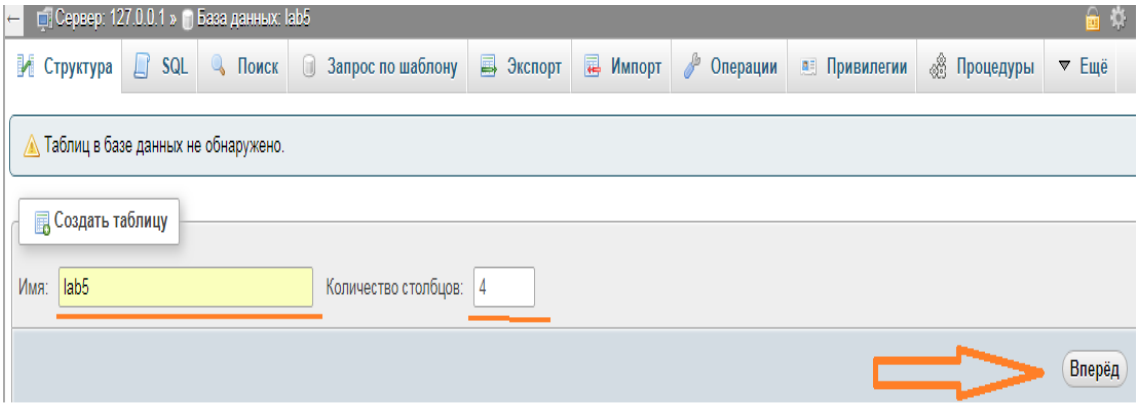

Рис 5.7– Создание таблицы базы данных

Для создания таблицы нажмем на кнопку "*Вперед*". После этого появится набор ячеек для установки параметров столбцов. Укажем последовательно для имен столбцов следующие: **id_lab, name, email, text**. В качестве типа нужно указать для столбцов **id_lab** тип *INT*, а для столбцов **name** и **email** - тип *VARCHAR*, для поля **text** тип данных *TEXT*. Для столбцов **name** и **email** в поле "Длина/Значения" укажем число 256 - оно будет указывать максимальную длину строки в символах. Также для столбца **id_lab** укажем в поле "Индекс" *PRIMARY* а в поле "*A_I*" (*AutoIncrement*) поставим галочку:

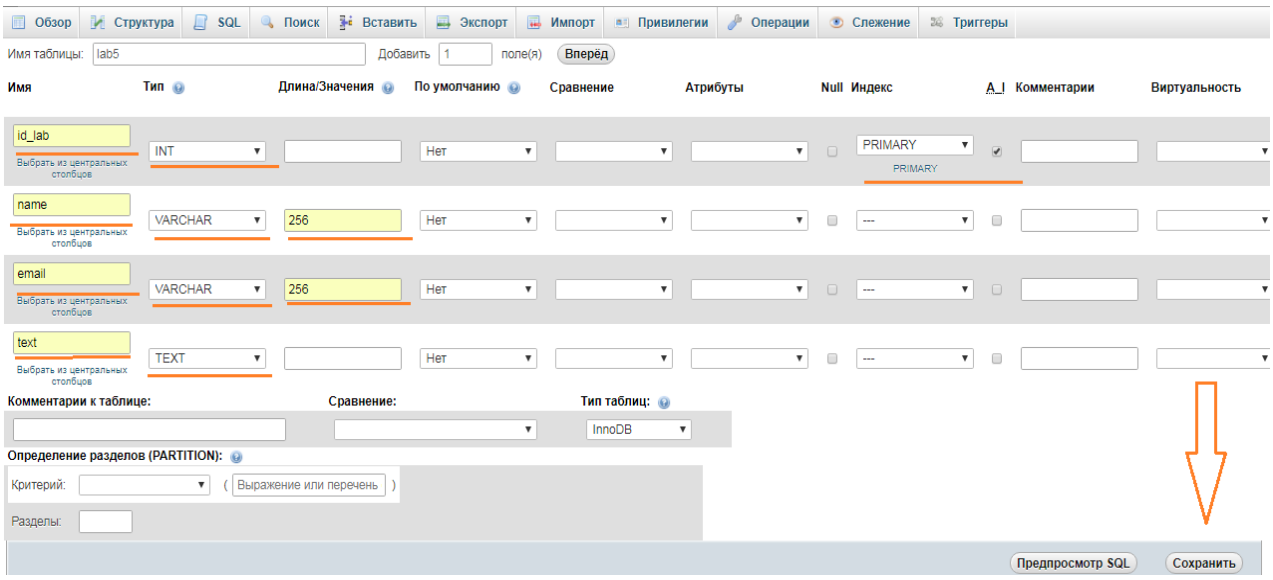

Рис 5.7 – Окно создания таблицы базы данных.

База данных создана и готова для использования.

В папке **inc** созданной в предыдущей лабораторной работе, создаем файл *bd.php* и добавим в него следующий код:

```
<?php
//имя базы данных, к которой будет подключение
      $dbName = 'test' ;
    /*
     * имя хоста для подключения к БД
     *
     *  */
$dbHost = 'localhost' ;
    /*
     *Имя пользователя для подключения к базе данных
     * */
      $dbUsername = 'root';
    /*
     * пароль для подключения к БД*/
      $dbUserPassword = '';
/*
 *переменная хранящая соединение с базой данных
 * */
      $cont  = null;

$cont =  new PDO( "mysql:host=".$dbHost.";"."dbname=".$dbName,
$dbUsername, $dbUserPassword);
    /*
     * Функция для сохранения данных в БД
     *  $name - имя пользователя, который оставил свой
комментарий
     * $email - электронный адрес
     * $text - комментарий

     *
     *  */
    function save($name, $email, $text)
    {
    global $cont;
    $sql = $cont->prepare("INSERT INTO lab5 (name, email,
text) values (?, ?, ?)");
            $result = $sql->execute( array($name, $email,
$text) );
    }

    /*
    Функция для фильтрации данных полученных через POST запрос
    $name - переменная для фильтрации данных
     *
     * */
    function filter($name)
    {
```

```
    $result = htmlspecialchars( strip_tags( trim($name)));
    return $result;
}

/*Функция для получения всех данных из БД
 *

 * */
function getAll()
{
 global $cont;
 $$result = $cont->query("SELECT * from lab5");
 return $result;
}
```

В данном файле, реализуется подключение к базе данных через библиотеку *PDO* и обеспечивает сохранение данных и выборку для отображения данных на странице сайта.
Для начала необходимо осуществить подключение к базе данных. Для этого создается экземпляр класса *PDO*. В экземпляре класса **$cont** будет храниться подключение к базе данных. Что бы обратиться к методам библиотеки *PDO*, необходимо использовать оператора ->.
Для обеспечения возможности обратиться к экземпляру класса **$cont**(переменная, которая хранит соединение с базой данных) в пользовательских функция, необходимо прописать строчку:

```
global $cont;
```

В файле bd.php реализовано 3 пользовательских функции:

1. **Save()** – пользовательская функция для сохранения с формы гостевой книги данных, введённых пользователем . На входе функции 3 параметра:
   - **$name** - имя пользователя, который оставил свой комментарий в форме гостевой книги.
   - **$email** – электронный адрес пользователя.
   - **$text** – комментарий пользователя.

   С помощью 2-х ступенчатой системы запросов выполняется сохранение данных в базу данных.
   Так как в запрос передаётся переменные **($name, $email, $text)**, то этот запрос в обязательном порядке должен выполняться только через подготовленные выражения. Подготовленные выражения - это обычный *SQL* запрос, в котором вместо переменной ставится специальный маркер - плейсхолдер. *PDO* поддерживает позиционные плейсхолдеры **(?**), для которых важен порядок

передаваемых переменных, и именованные **(:name),** для которых порядок не важен.

```
$sql = $cont->prepare("INSERT INTO lab5 (name, email,
text) values (?, ?, ?)");
```

Чтобы выполнить такой запрос, сначала его надо подготовить с помощью метода **prepare()**. Она также возвращает *PDO statement*, но ещё без данных. Чтобы их получить, надо исполнить этот запрос, предварительно передав в него переменные. Чаще всего можно просто выполнить метод **execute()**, передав ему массив с переменными:

```
$sql->execute( array($name, $email, $text) );
```

2. **Filtet()** – пользовательская функция, обеспечивающая фильтрация данных полученных через метод POST. Для очистки данных используются функции:
   - **htmlspecialchars()** - полезна при отображении данных, введенных пользователем, которые могут содержать нежелательные HTML тэги в форме гостевой книги;
   - Функция **strip_tags() -**Удаляет теги HTML и PHP из строки;
   - Функция **trim()** – убирает пробелы в начале и в конце php строки;
3. **getALL()** – пользовательская функция для получения всех данных из базы данных. Для выполнения выборки данных используется метод **query()**. В круглых скобках указывается, что необходимо выбрать. В данном случае запрос означает: выбрать(**Select**) все поля(*****) из таблицы (**lab5** – таблица ,которая была создана с помощью *phpmyadmin*).Результат данного запроса сохраняется в переменной **$q**. Для получения данных в виде массива используется метод **fetchAll()**.Функция **getALL** возвращает результат запроса в виде массива.

В файле *gbook.inc.php* в строчке, где происходит объявление формы с помощью тега **<form>** нужно добавить атрибуты:

```
action ="form.php" method ="POST"
```

В Атрибуте **action** тега **<form>** указывается имя скрипта, который будет обрабатывать форму и передавать данные на сервер для сохранения в базе данных. Атрибут **method** предназначен для указания серверу, каким методом будет передавать данные. Для того, чтобы данные были переданы в защищённом в виде используем метод *POST*. Для осуществления считывания содержимого с поля, необходимо каждому полю через атрибут name присвоить имя. Для обработки кнопки необходимо указать имя с

помощью атрибута name. Полный код формы гостевой книги, выглядит следующим образом:

```
<form action =”form.php” method =”POST” >
     <div class="form-group">
       <label>Имя</label>
       <input type="text" class="form-control"
placeholder="Введите имя "name="name">
     </div>
     <div class="form-group">
       <label>EMAIL:</label>
       <input type="text" class="form-control"
placeholder="Введите Email" name="email">
     </div>
     <div class="form-group">
       <div class="form-group">
       <label>Сообщение</label>
          <textarea class="form-control" placehold-
er="Сообщение" name="text"></textarea>
      </div>

      <div class="form-group">
         <input type="submit" class="btn btn-info"
value="Отправить" />
```

Для то что бы осуществить отправку данных из формы на сервер необходимо создать кнопку:

```
<input type="submit" class="btn btn-info" name = “submit” val-
ue="Отправить" />
```

В атрибуте тега **type** указываем значение **submit**, для отправки данных формы на сервер. Атрибут **name** необходим для указания имени кнопки, по данному имени к ней можно будет обратиться.В атрибут **value** указываем с каким именем будет отображаться кнопка на странице сайта.

Что бы получить данные из формы необходимо для каждого поля, с которого будем получать информацию указать атрибут тега **name**. В данном атрибуте указываем имя данного поля. Непосредственное получение данных с формы происходит через суперглобальный массив **$_POST**.Что бы получить через **$_POST** данные с поля, например, с поля **name="email"**, необходимо в качестве параметра в **$_POST** указать имя поля.

```
$_POST['email'];
```

В файле *gbook.inc.php*, после кода формы добавим следующий код, который будет обеспечивать отображение данных из базы данных.

```
<table border="1"    align="center" class="table" ">
<tr>
```

```
      <td>Имя пользователя</td>
      <td>Емейл</td>
      <td>Комментаций</td>
</tr>
    <?php
    include 'bd.php';
    foreach(getALL() as $row)
    {
             echo "<tr><td>" . $row["name"]
          . "</td><td>" . $row["email"]
          . "</td><td>" . $row["text"]
          . "</td></tr>";
      }
?>
 </table>
```

Для выборки данных на вход цикла **foreach** нужно подать функцию **getALL().**

На каждой итерации значение текущего элемента присваивается переменной **$row** и внутренний указатель массива увеличивается на единицу (таким образом, на следующей итерации цикла работа будет происходить со следующим элементом).

Теперь создадим файл скрипт обработчик формы. Для это в папке **inc** создадим файл с именем *form.php* и добавим в него следующий код:

```
<?php
/*
 * Скрипт релазующий по нажатию кнопки submit передачу данных
с формы в базу данных
 * */
// подлючение файла bd.php релазиющего взаимодействие с базой
данных
include_once "bd.php";
// Проверка,была ли нажата кнопка с именем submit
    if(isset($_POST['submit']))
    {
        // фильтация данных полученных через метод POST
        $name  = filter($_POST['name']);
        $email = filter($_POST['email']);
        $text  = filter($_POST['text']);

        //Запись данных в базу данных
        save($name, $email, $text);

        //переход на страницу после записи данных в базу
данных
        header("Location: ../index.php?id=gbook");
    }
```

Алгоритм обработки формы гостевой книги следующий:

1. Подключение с помощью инструкции **include** файла с именем *bd.php***.**

```
include_once "bd.php";
```

2. Проверяем нажата ли кнопка с именем **submit**, для этого с помощью функции **isset** проверяем существует ли переход с помощью суперглобального массива **$_POST** параметра submit.

```
if(isset($_POST['submit']))
```

3. Если кнопка с именем submit нажата, то с помощью пользовательской функции **filter(),** фильтруем данные полеченные с формы с помощью суперглобального массива **$_POST** и записываем полученные данные в переменные.

```
$name  = filter($_POST['name']);
$email = filter($_POST['email']);
$text  = filter($_POST['text']);
```

4. После того, как данные были отфильтрованные от мусора с помощью функции **filter()**, необходимо их передать их в качестве параметров функции **save()** для сохранения данных в базе данных.

```
save($name, $email, $text);
```

5. Далее, когда данные были сохранены, переходим на страницу гостевой книги. На данной страничке далее будет отображены данные которые были введены через форму.

```
header("Location: ../index.php?id=gbook");
```

5.4. Порядок выполнения

5.4.1. Создать с помощью графической оболочки *phpmyadmin* базу данных.

5.4.2. Создать файл *bd.php* и записать в него код реализующий подключение к базе данных с помощью библиотеки *PDO* , сохранение данных с помощью библиотеки *PDO* ,выборку данных с помощью библиотеки *PDO* и фильтрацию данных.

5.4.3. Создать файл *form.php* реализующий обработку данных с формы и передачу данных на сервер.

5.4.4. Осуществить с помощью библиотеки *PDO* выборку данных из базы данных и отобразить в виде таблицы.

5.4.5. Подготовить отчет по работе.

5.5. Оформление отчета

5.5.1. Сформулировать цель работы.

5.5.2. Сформулировать постановку задачи и индивидуальное задание.

5.5.3. Привести краткие теоретические сведения.
5.5.4. Привести тексты программ**.**
5.5.5. Привести скриншоты выполнения программы.
5.5.6. Выводы.

5.6. Контрольные вопросы:

5.6.1. Что такое база данных?
5.6.2. Что такое *СУБД*?
5.6.3. Что такое язык запросов *SQL*? Для чего он нужен? Пример.
5.6.4. Этапы создания базы данных через *phpmyadmin*.
5.6.5. Что такое *CRUD*? Пример *CRUD*.
5.6.6. Что такое *PDO*? Краткая характеристика. Подключение к базе через PDO.
5.6.7. Пример кода создания формы гостевой книги.
5.6.8. Каким образом происходит добавление данных в базу данных средствами *PDO*?
5.6.9. Каким образом происходит выборка данных из базы данных средствами *PDO?*
5.6.10. Каким образом происходит фильтрация данных полученных из формы?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Харрис, Э. PHP/MySQL для начинающих / Э. Харрис. – М.: Кудиц Образ, 2007. – 384 с.
2. Зандстра, Мэт. PHP. Объекты, шаблоны и методики программирования / Мэт Зандстра. – СПб. : Вильямс, 2011. – 560 с. – ISBN 978- 5-8459-1689-1.
3. Веллинг, Л. Разработка веб-приложений с помощью PHP и MySQL / Л. Веллинг, Л. Томсон. – СПб. : Вильямс, 2010. – 848 с. – ISBN 978- 5-8459-1574-0.
4. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, Javascript и CSS / Р. Никсон. – СПб. : Питер, 2013. – 560 с. – ISBN 978-5-496-00187-8.