МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

ИНСТИТУТ ТРАНСПОРТНОЙ ТЕХНИКИ И СИСТЕМ УПРАВЛЕНИЯ (ИТТСУ)
Кафедра «Управление и защита информации»

А.И. САФРОНОВ, А.И. КОТОВА

ПРОЕКТИРОВАНИЕ ТИПОВОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ УПРАВЛЕНИЯ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ *WEB*-ПРОГРАММИРОВАНИЯ НА БАЗЕ ФРЕЙМВОРКА *VUE.JS*

Учебно-методическое пособие
для проведения аудиторных занятий по дисциплине
«Информационное обеспечение систем управления»

МОСКВА – 2019

МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

ИНСТИТУТ ТРАНСПОРТНОЙ ТЕХНИКИ И СИСТЕМ
УПРАВЛЕНИЯ (ИТТСУ)
Кафедра «Управление и защита информации»

А.И. САФРОНОВ, А.И. КОТОВА

ПРОЕКТИРОВАНИЕ ТИПОВОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ УПРАВЛЕНИЯ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ *WEB*-ПРОГРАММИРОВАНИЯ НА БАЗЕ ФРЕЙМВОРКА *VUE.JS*

Учебно-методическое пособие
для проведения аудиторных занятий по дисциплине
«Информационное обеспечение систем управления»
для бакалавров направления
27.03.04 «Управление в технических системах»

МОСКВА – 2019

УДК 004

С 21

Сафронов А.И., Котова А.И. Проектирование типовой информационной системы управления с использованием технологии *web*-программирования на базе фреймворка *Vue.js*: Учебно-методическое пособие для проведения аудиторных занятий по дисциплине «Информационное обеспечение систем управления». – М.: РУТ (МИИТ), 2019. – 97 с.

Учебно-методическое пособие составлено с целью предложения обучающимся альтернативного инструментария для выполнения курсового проекта по дисциплине «Информационное обеспечение систем управления». В нём изложены основы подключения, настройки и последующей нормальной работы фреймворка *Vue.js*, а также собраны возможные темы курсовых проектов по изучаемой дисциплине. Рассматриваются простейшие примеры реализации типовых вычислительных процессов: последовательного, разветвляющегося, циклического; примеры использования базовых директив фреймворка, а также реализация функционального списка как одного из компонентов абстрактной информационной системы управления по тематике «Отдел кадров».

Рецензент:

Заведующий кафедрой «Электропоезда и локомотивы», д.т.н., проф. Пудовиков О.Е.

Введение. Общие сведения о типовой информационной системе управления.

Дисциплина «Информационное обеспечение систем управления» (ИОСУ) призвана помочь обучающимся в освоении специфики информационных систем управления (ИСУ), которые в общем случае состоят из базы данных (*Data Base*) и оболочки (*Shell*). Оболочка представляет собой, прежде всего, графический пользовательский интерфейс, обеспечивающий навигационный способ доступа к данным, хранимым в базе данных. С другой стороны, оболочка – это средство обеспечения безопасности и контроля целостности данных, хранимых в базе.

В процессе изучения дисциплины «ИОСУ» предусмотрено создание обучающимися типовой ИСУ. Как правило, она реализуется в рамках курсового проектирования и/или во время практических занятий. Проектирование ИСУ нацелено на закрепление на практике обучающимися приобретённых теоретических знаний о технологии разработки, собственно, типовых ИСУ.

Исторически сложилось, что обучающимися кафедры «Управление и защита информации» (УиЗИ) курсовые проекты по дисциплине «ИОСУ» создавались согласно методике, изложенной в [1, 2]. Данная методика подразумевает составление базы данных *Paradox* с последующей организацией навигационного доступа к данным в среде (*IDE*) объектно-ориентированного программирования *Borland Delphi* посредством сопряжения этой среды с базой данных через *Borland Database Engine* (*BDE*).

Позднее, с переходом кафедры «Управление и защита информации» с языка *Pascal* (*Delphi*), преподаваемого в качестве основы, на язык *Visual C#,* пришлось пересмотреть сопрягаемые между собой программные компоненты.

Выбор СУБД, обозреваемой в рамках курса «ИОСУ», остановился на *Microsoft Office Access*, поскольку ряд

автоматизированных систем управления технологическими процессами (АСУ ТП), созданных и сданных кафедрой «УиЗИ» в промышленную эксплуатацию, основывался и по сей день основывается именно на этой СУБД. Примером такой системы является «АРМ Графиста» на ГУП «Московский метрополитен».

Принятое решение подкреплено так же и следующей идеей: «Выпускаемые кафедрой кадры должны обладать достаточным уровнем знаний, умений и навыков, необходимых для обслуживания систем, созданных и сопровождаемых сотрудниками кафедры».

Вместе с тем, работать с СУБД *Microsoft Office Access* возможно повсеместно, что, безусловно, повышает мобильность создаваемых обучающимися проектов: над ними можно работать в домашних условиях, в компьютеризированных аудиториях кафедры, на ноутбуке в общественном транспорте, на иных рабочих местах, где предустановлен пакет офисных приложений *Microsoft Office*.

За последние годы ИСУ, выполняемые обучающимися в ходе работ по курсовому проектированию в рамках дисциплины «ИОСУ», создавались как сочетание файлов локально-распределённой СУБД *Microsoft Office Access* и оболочки, написанной обучающимися в среде *Microsoft Visual Studio* на языке *Visual C#*.

Жизнь не стоит на месте, а вместе с ней не стоят на месте и информационные технологии (*IT*). Таким образом, при анализе рынка было определено следующее: большинство передовых *IT*-фирм, занятых разработкой программного обеспечения, активно используют Интернет-технологии и стремятся максимально уйти от локальных клиентских приложений в сторону браузерных веб-приложений.

Безусловно при работе с обучающимися важно учитывать повседневность *IT*-рынка и помогать им развивать знания, умения и навыки в этом направлении. Но для того, чтобы сделать направление веб-программирования основным на кафедре, необходимо менять и учебно-методическую базу, преподавая обучающимся с начальных курсов основы языков

веб-программирования. В одночасье это реализовать невозможно, поскольку за изменением базы программирования последует довольно длительный переходный процесс.

В связи с вышесказанным, целью данного учебно-методического пособия является предоставление обучающимся альтернативного способа разработки ИСУ в рамках курсового проекта по дисциплине «ИОСУ». Целевая аудитория – наиболее сильные обучающиеся кафедры «УиЗИ», которые в кратчайшие сроки и самостоятельно готовы к освоению новых информационных технологий.

Материал, представленный в данном издании, не отменяет текущей схемы сопряжения *Visual C#* с СУБД *Microsoft Office Access* для составления курсового проекта по дисциплине «ИОСУ». Он лишь позволяет обучающимся расширить свой кругозор и предоставляет возможность выполнения курсового проекта при сопряжении одностраничного веб-приложения, основанного на фреймворке *Vue.js*, с файлами текстового формата, необходимыми для хранения данных в синтаксисе *JSON*.

На страницах учебно-методического пособия рассматриваются основные идеи и принципы составления баз данных и оболочек для обеспечения навигационного доступа к базам данных.

Издание призвано изложить понятным, простым языком некоторую унифицированную методику разработки обучающимися типовых ИСУ, представляющих собой браузерные одностраничные веб-приложения, достаточных для дальнейшего развития их под задачи и специфику предметных областей, рассматриваемых при написании выпускных квалификационных работ на кафедре «Управление и защита информации» по программе бакалавриата.

На курсовой проект ставится конкретная задача по составлению ИСУ под определённую предметную область, выдаваемую отдельно взятому обучающемуся в качестве варианта задачи автоматизации технологического процесса (см. Приложение 1).

1 Основы создания локального одностраничного веб-приложения

1.1 Основные определения и обоснование необходимости создания одностраничных веб-приложений

Под локальным одностраничным веб-приложением (*Local Single-Page Web-Application, LSPWA*) далее понимается прототип перспективной ИСУ, предназначенный для отладки и учебных целей. Прототип является программной оболочкой и в качестве базы данных использует один из стандартизованных текстовых форматов передачи данных (*JSON* и/или *XML*).

Введём определение [3]: «Фреймворк – это:

- программная платформа, определяющая структуру программной системы;
- программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта».

Иными словами – это программируемый каркас разрабатываемой ИСУ.

Обучающимся для изучения и использования предлагается фреймворк «*Vue.js*», управляемый посредством команд и операторов, записываемых на языке *JavaScript* [4].

JavaScript – мультипарадигменный язык программирования. Он поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией языка *ECMAScript* (стандарт *ECMA-262*). *JavaScript* обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам [5].

Мультипарадигменное программирование [6] – программирование с одновременным использованием множества парадигм (идей, подходов, шаблонов).

Основные подходы к организации мультипарадигменного программирования:

- создание нового языка программирования,
- расширение существующего языка программирования,
- встраиваемые интерпретаторы,
- расширяемые интерпретаторы,
- трансляция из одного языка в другой,
- сборка модулей, написанных на разных языках программирования,
- библиотечное расширение существующего языка программирования.

Парадигма программирования [7] – это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

Важно отметить, что парадигма программирования не определяется однозначно только языком программирования; практически все современные языки программирования в той или иной мере допускают использование различных парадигм.

Так, на языке Си (имеется в виду *C++*, а не *C#*), который не является объектно-ориентированным, можно работать в соответствии с принципами объектно-ориентированного программирования, хотя это и сопряжено с определёнными сложностями; функциональное программирование можно применять при работе на любом императивном языке, в котором имеются функции, и т.д.

Столь же важно отметить, что существующие парадигмы зачастую пересекаются друг с другом в деталях (например, модульное и объектно-ориентированное программирование), поэтому можно столкнуться с ситуациями, когда различные авторы употребляют названия из разных парадигм, говоря при этом, по сути, об одном и том же явлении.

Итак, фреймворк «*Vue.js*». В сети Интернет для заинтересованных лиц имеется переведённое на русский язык официальное руководство пользователя [8]. Без него разобраться с фрейморком невозможно, но и при отсутствии под

рукой работоспособных примеров задача освоения «*Vue.js*» тоже является проблематичной, потому авторы в текущем разделе (разделе 1) берутся за рассмотрение простейших и функциональных основ шаг-за-шагом со всеми необходимыми комментариями, помогающими в освоении фреймворка, а заодно и языка программирования *JavaScript*.

Примеров, работающих должным образом и решающих конкретные задачи, в сети Интернет на сегодня размещено довольно мало, и это обстоятельство обуславливает высокую информативность и доступность для обучающихся РУТ (МИИТ) данного учебно-методического пособия.

Энтузиастами в сети Интернет решены лишь частные задачи, которые, преимущественно, являются задачами разработки типовых веб-сайтов, но не задачами создания комплексных ИСУ для учебных и корпоративных нужд.

Авторы настоятельно рекомендуют ознакомиться с разделом, содержащим примеры, представленные на официальном сайте фреймворка [9]. Некоторые из представленных там образцов легли в основу решений, рассмотренных в последующих разделах данных учебно-методических указаний.

Одним из видимых преимуществ разработки веб-приложений на базе фреймворка «*Vue.js*» является возможность их написания в обыкновенном текстовом блокноте (*Notepad*) операционной системы *Microsoft Windows*.

Следующий параграф данного раздела нацелен на изложение методики, обучающей азам работы с фреймворком в обычном блокноте.

1.2 Подключение и настройка ядра фреймворка *Vue.js*

В качестве подхода к решению задачи подключения ядра фреймворка *Vue.js* рассматривается самый простой пример, представленный в его официальном руководстве [8] (Рисунок 1.2.3).

Но прежде необходимо определить понятия. Под ядром авторы понимают огромный сценарий *Vue.js*, реализующий весь полезный функционал фреймворка.

В предыдущем параграфе данного раздела отмечено, что код примера может быть размещён в одном единственном текстовом файле. Важно внести некоторое уточнение, что речь идёт не о простом текстовом файле, а о текстовом файле, содержащем, прежде всего, гипертекстовую разметку. Такие файлы хранятся уже не с расширением «**.txt*», а соответствующим гипертекстовой разметке (*Hyper Text Markup Language*) расширением «**.html*» для понимания этих файлов браузерами.

Стоит напомнить (или рассказать впервые), что гипертекстовая разметка заключается в теги «*HTML*»: открывающий (*<HTML>*) и закрывающий (*</HTML>*). С них и начинается разметка. Размеченный шаблон веб-страницы размещается внутри вложенных тегов «*BODY*» (так же открывающего и закрывающего). В качестве шаблона может выступать строка статического текста, например, «Добрый день!».

Данная структура (Рисунок 1.2.1) является простейшей статической веб-страницей (ещё не веб-приложением), отображаемой в браузере (Рисунок 1.2.2).

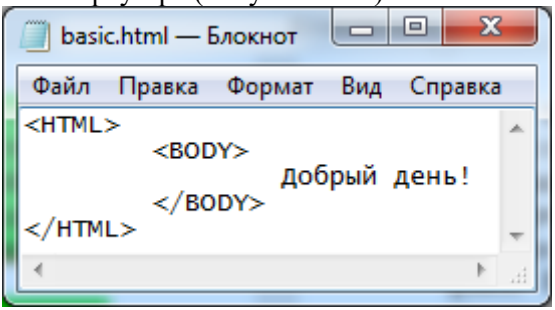


Рисунок 1.2.1 – Код простейшей веб-страницы, отображаемой в браузере

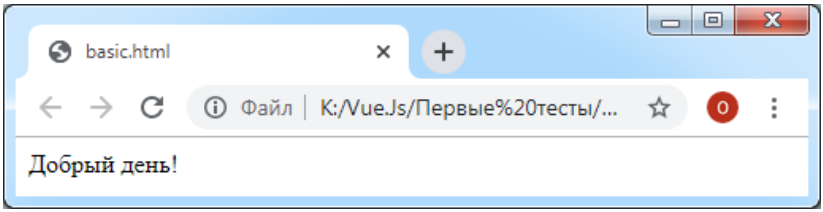


Рисунок 1.2.2 – Результат отображения простейшей веб-страницы в браузере

Нет необходимости отклоняться от курса, задаваемого официальной документацией *Vue.js*, потому написание кода впредь будет вестись в файлах с именами «*index.html*», созданных в блокноте *Notepad*.

Это ни в коем случае не вольность, и даже не авторская прихоть. Это правило. При обращении к большинству веб-сайтов по их доменному или *IP*-адресу первой открывается по умолчанию страница с именем «*index.html*», размещённая в корневом каталоге (если не предусмотрено иного).

Итак, к упомянутому примеру из официального руководства (Рисунок 1.2.3). На сайте фреймворка в разделе официальной документации [8] код, как правило, представлен следующими фрагментами:

- результат (*Result*);
- гипертекстовая разметка (*HTML*);
- скрипт, программный сценарий (*Script*);
- настройка стилевого оформления (*CSS*).

На Рисунке 1.2.3 отсутствует лишь блок стилевого оформления. Вместо него сразу продемонстрирован результат. Результат представлен надписью: «Привет, *Vue!*» без каких-либо отметок в правом верхнем углу блока, в то время как у иных блоков, содержащих код, в правом верхнем углу выполнена отметка с наименованием парадигмы программирования (*HTML*, *JS*, *CSS*), согласно которой обрабатывается представленный фрагмент кода.

Рисунок 1.2.3 – Иллюстрация типового примера из официальной документации к фреймворку *Vue.js*

Ставится задача повторить рассмотренный пример на локальном персональном компьютере без явного обращения к Интернет-ресурсам.

Если внимательно и шаг-за-шагом читать руководство [8], то можно остановиться на очень важном и значимом абзаце про то, как именно подключается к локальному проекту ядро «*Vue.js*». Этот фрагмент представлен на Рисунке 1.2.4: «А можно и просто создать *index.html* файл на диске и подключить *Vue:*» [8]. Так в дальнейшем и необходимо поступать при создании локальных одностраничных веб-приложений.

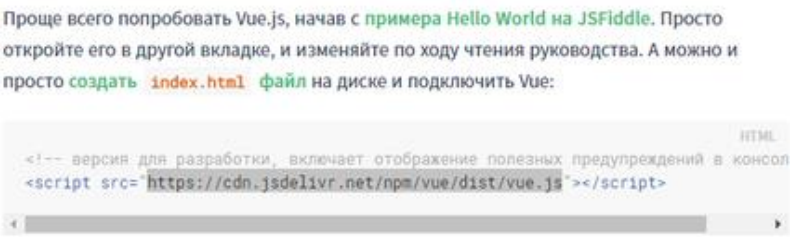


Рисунок 1.2.4 – Ссылка на наиболее свежие версии *Vue.js* из официальной документации к фреймворку

В текстовом файле «*index.html*» обязательно внутри тега <*body*>, ответственного за наполнение и разметку шаблона *html*-страницы, в самом начале должна быть записана следующая строка кода:

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script> (1)

Читается фрагмент данного оператора разметки «*script src*» как «скрипт сурс» (*Script Source*), то есть источник поставляемого веб-приложению сценария. За ним следует адрес размещения сценария либо в сети (1), либо локально в файле (2), заключённый в кавычки. Эта строка кода в дальнейшем для локального проекта веб-приложения сократится до:

<script src="Vue.js"></script> (2)

Именно эта конструкция и обеспечит развязку проекта с какими-либо Интернет-ресурсами. Его можно будет запускать и при отсутствии соединения с Интернет.

Рассматривается перенос примера (Рисунок 1.2.3) на локальный компьютер шаг-за-шагом.

1. На любом из локальных дисков (например, на *C:*) создать папку с именем «*Vue Example*».
2. В ней создать новый текстовый файл «*index.txt*».
3. В нём записать шаблон, представленный на Рисунке 1.2.5, а также далее по тексту (3).
4. Сохранить изменения.

5. Заменить расширение «*txt*» на «*html*».

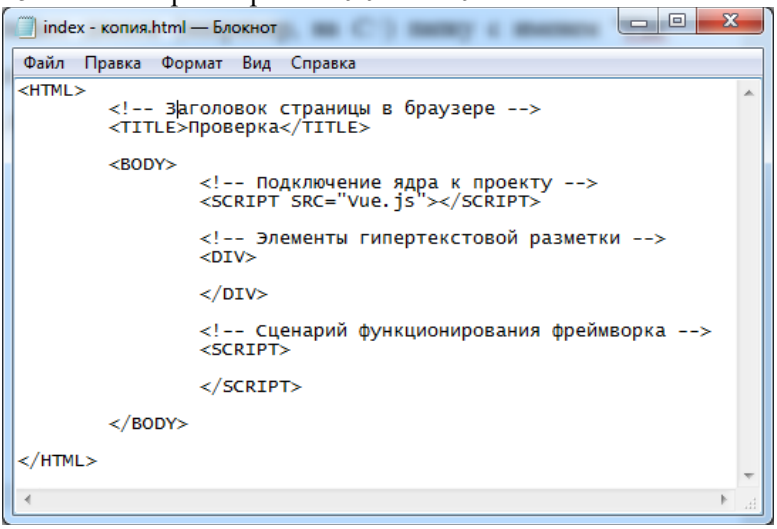


Рисунок 1.2.5 – Заготовка для разработки одностраничного веб-приложения в блокноте (*Notepad*)

```
<HTML>
	<!-- Заголовок страницы в браузере -->
	<TITLE>Проверка</TITLE>
	<BODY>
		<!-- Подключение ядра к проекту -->
		<SCRIPT SRC="Vue.js"></SCRIPT>
		<!-- Элементы гипертекстовой разметки -->
		<DIV>
		</DIV>
		<!-- Сценарий функционирования фреймворка -->
		<SCRIPT>
		</SCRIPT>
	</BODY>
</HTML>
```
(3)

Этот файл («*index.html*») можно скопировать для создания в дальнейшем с нуля и других проектов. Хорошо видно, что тег <*body*> содержит три основных раздела:

- подключение ядра;
- гипертекстовую разметку (*html*), представленную в шаблоне пустым тегом <*div*> (от английского «*division*» - раздел/блок);

- сценарии работы одностраничного веб-приложения (*script*).

Но сразу в таком исполнении одностраничное приложение работать не будет. **Это не более, чем удобная заготовка.**

Для понимания физики процессов необходимо вернуться к (1), то есть заменить текст в шаблоне

с

<SCRIPT SRC="Vue.js"></SCRIPT>

на

<SCRIPT SRC="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>.

Вместе с тем и текст

с

<DIV></DIV>

на

<DIV ID="app"> {{message}} </DIV>.

Далее конструкцию *<SCRIPT></SCRIPT>* на (4).

```
<SCRIPT>
    var app = new Vue(
    {
        el: '#app',
        data:
        {
                message: 'Hello, Vue!'
        }
    })
</SCRIPT>
```
(4)

По аналогии с примером, представленным на Рисунке 1.2.3, должно получиться нижеследующее (5):

```
<HTML>
    <TITLE>Проверка</TITLE>
    <BODY>
        <SCRIPT SRC="https://cdn.jsdelivr.net/npm/vue/dist/vue.js">
        </SCRIPT>
        <DIV ID="app"><p>{{ message }}</p></DIV>
        <SCRIPT>
                var app = new Vue(
                {
                        el: '#app',
                        data: { message: 'Hello, Vue!' }
                })
        </SCRIPT>
    </BODY>
</HTML>
```
(5)

Тот же код, записанный в блокноте, представлен на Рисунке 1.2.6.

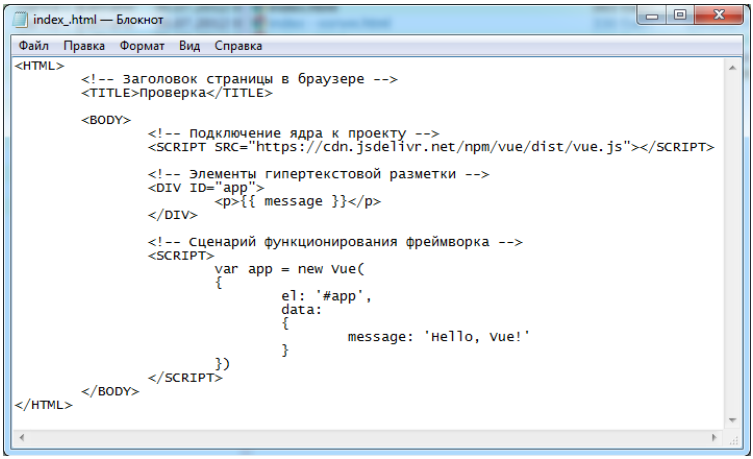


Рисунок 1.2.6 – Модифицированная заготовка для разработки одностраничного веб-приложения на базе фреймворка *Vue.js*

Итак, получилась ситуация, когда в папке «*Vue Example*», размещённой на локальном диске (*C:*), содержится один единственный *html*-файл (Рисунок 1.2.7) с содержимым, представленным на Рисунке 1.2.6.

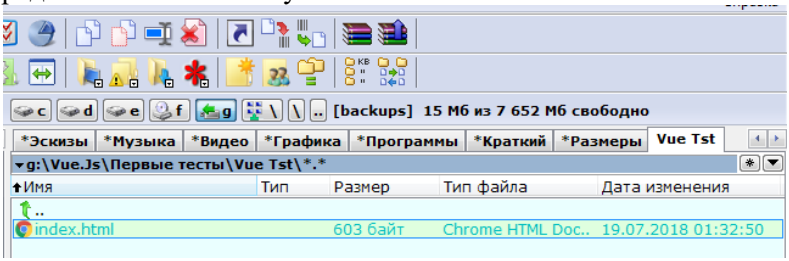


Рисунок 1.2.7 – Просмотр директории, содержащей файл «*index.html*», в окне файлового менеджера *Total Commander*

Если открыть этот файл через браузер, например, «*Google Chrome*», то должен быть получен результат, аналогичный представленному на Рисунке 1.2.8.

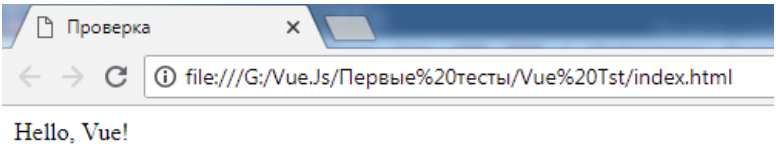


Рисунок 1.2.8 – Результат работы примера из официального руководства *Vue.js* на локальном компьютере с подключением к фреймворку через Интернет

Можно работать с фреймворком так и только так, однако «сегодня веб-ресурс жив, а завтра – мёртв», потому разумно желание разработчика иметь некоторою гарантию того, что его одностраничное веб-приложение не откажет в работе даже в условиях, когда веб-ресурс, поставщик сценария фреймворка *Vue.js*, окажется недоступным.

Под ситуацию можно сформулировать следующее правило, которое должно со временем войти в привычку: «Понравилось что-то в сети – в тот же день сохрани это на жёсткий диск».

Точно так же следует поступить и с фреймворком «*Vue.js*». По итогам тестирования не оспаривается факт, что данный фреймворк всецело удовлетворяет потребностям на разработку одностраничных веб-приложений, и принимается решение о его сохранении на жёсткий диск для последующего локального использования.

Для этого необходимо перейти по ссылке источника (*https://cdn.jsdelivr.net/npm/vue/dist/vue.js*), который был указан как «скрипт сурс», через браузерное окно.

Источник содержит код ядра фреймворка *Vue.js* (Рисунок 1.2.9).

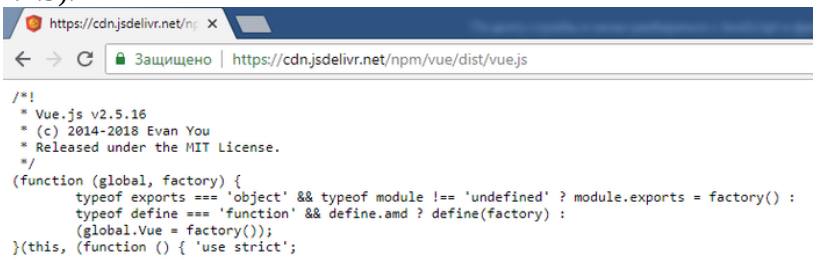


Рисунок 1.2.9 – Фрагмент текстового содержимого ссылки *https://cdn.jsdelivr.net/npm/vue/dist/vue.js*

Код необходимо полностью скопировать, предварительно выделив сочетанием клавиш «*Ctrl*» + «*A*» (Выделить всё), и вставить в новый блокнот (Рисунок 1.2.10), через который сохранить под именем «*Vue.js*» в той же папке («*Vue Example*»), где ранее был создан файл «*index.html*» (Рисунок 1.2.11).

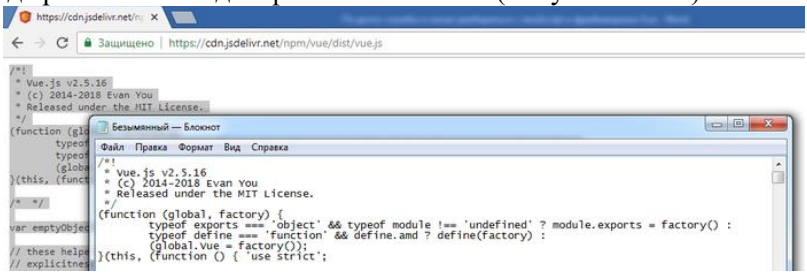


Рисунок 1.2.10 – Копия текста ядра *Vue.js*, размещённая в блокноте

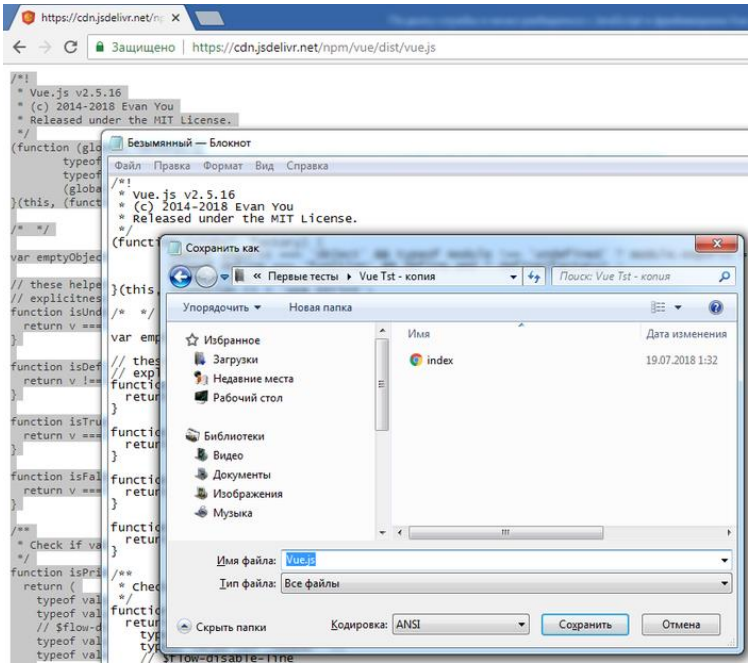


Рисунок 1.2.11 – Сохранение ядра «*Vue.js*» в директорию с ранее созданным файлом «*index.html*»

Теперь в директории содержатся два файла (Рисунок 1.2.12), проект стал существенно более тяжёлым, но вместе с тем получил и определённые гарантии по работоспособности в условиях отсутствия Интернет-соединения с удалённым поставщиком сценария *Vue.js*.

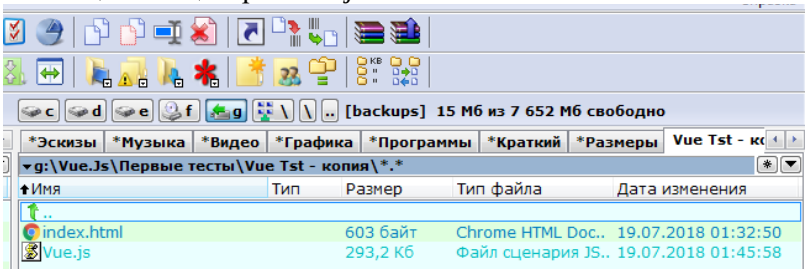


Рисунок 1.2.12 – Просмотр директории, содержащей файлы «*index.html*» и «*Vue.js*», в окне файлового менеджера «*Total Commander*»

Далее в коде «*index.html*» необходимо заменить «скрипт сурс» с (1) на (2), как это показано на Рисунке 1.2.13, и созданное по примеру из официального руководства приложение должно остаться работоспособным.

```
<BODY>
        <!-- Это подключение ядра к проекту -->
        <SCRIPT SRC="Vue.js"></SCRIPT>


        <!-- Это элементы гипертекстовой разметки
                Спасибо блогосфере за то, что обучила меня HTML'ю -->
```

Рисунок 1.2.13 – Изменения источника ядра *Vue.js* в коде с удалённого на локальный

Работа подтверждается представленным на Рисунке 1.2.14 результатом тестирования одностраничного веб-приложения.

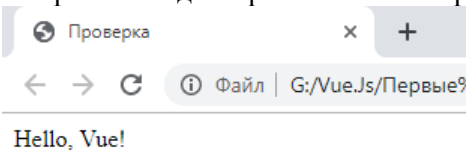


Рисунок 1.2.14 – Результат тестирования локального веб-приложения

В рассмотренном примере из официальной документации чёткому восприятию и пониманию механизмов сопряжения

гипертекстовой страницы с фреймворком *Vue.js* препятствует наличие одинаковых имён переменных в структуре приложения (Рисунок 1.2.15).

```
<!-- Сценарий функционирования фреймворка -->
<SCRIPT>

        var app = new Vue(
        {
                el: '#app',
                data:
                {
                        message: 'Hello, Vue!'
                }
        })
</SCRIPT>
```

Рисунок 1.2.15 – Дублирование имён переменной сценария и идентификатора объекта/элемента фреймворка

Поставлен эксперимент: изменено имя переменной сценария, ответственной за хранение объекта *Vue.js*. **Было**: «*var app = new Vue(…)*»; **стало**: «*var appVue = new Vue(…)*» (Рисунок 1.2.16).

```
<DIV ID="app">
        <p>{{ message }}</p>
</DIV>


<!-- Сценарий функционирования фреймворка -->
<SCRIPT>

        var appVue = new Vue(
        {
                el: '#app',
                data:
                {
                        message: 'Hello, Vue!'
                }
        })
```

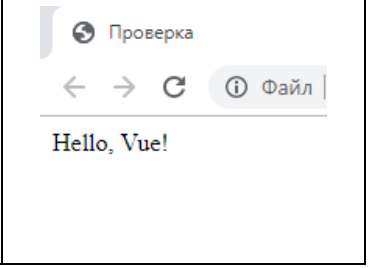


Рисунок 1.2.16 – Результат эксперимента по замене имени переменной объекта *Vue.js* в сценарии

Приложение сохранило работоспособность. Это свидетельствует о том, что сцепление *HTML* блока *<DIV>* происходит не по имени переменной, хранящей ссылку на созданные объект *Vue.js*, а по имени элемента, которое записывается в разделе «*el:*» фреймворка (Рисунок 1.2.17).

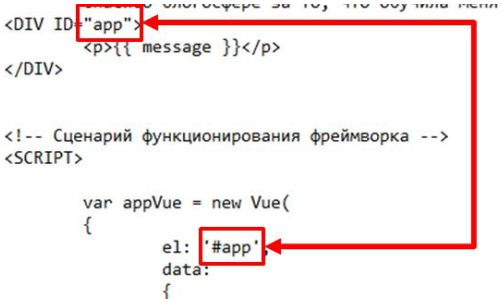


Рисунок 1.2.17 – Указание связующего имени для сценария и гипертекстового шаблона

Этот факт так же следует проверить: выполнена замена в блоке <*DIV*> идентификатора сопряжения с фреймворком. **Было**: «<*DIV ID="app"*>»; **стало**: «<*DIV ID="appVue"*>» (Рисунок 1.2.18).

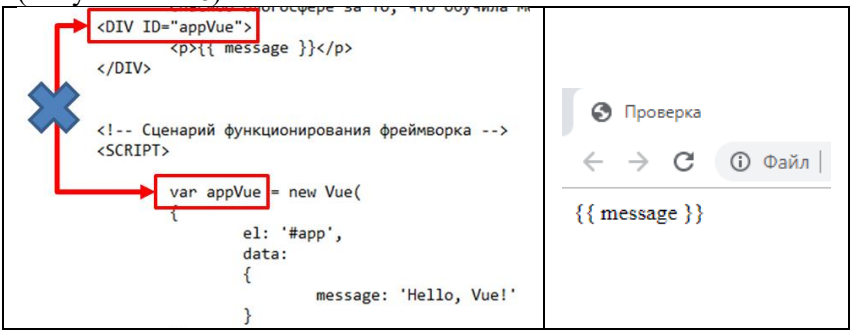


Рисунок 1.2.18 – Результат эксперимента по замене имени связки блока *div* с имени элемента фреймворка *Vue.js* на имя переменной, хранящей ссылку на объект *Vue.js*

Приложение утратило работоспособность.

В данном разделе и его параграфах рассмотрена реализация последовательного (линейного) вычислительного процесса, в котором операторы выполняются строго друг за другом в указанном алгоритмическом порядке последовательно.

2 Подготовка операционной системы к удобному программированию одностраничных веб-приложений на базе фреймворка *Vue.js*

На этапе, пока изучение фреймворка *Vue.js* в локальном режиме не продвинулось достаточно далеко, следует отойти в сторону от возможностей самого фреймворка и рассмотреть полезное и удобное для разработчиков дополнительное программное обеспечение (помимо специализированной среды разработки).

В данном разделе речь идёт, прежде всего, об установке под управление операционной системы *Microsoft Windows* блокнота с расширенной функциональностью «*Notepad++*».

Следует принять во внимание тот факт, что блокнот «*Notepad++*» не является, вообще говоря, программным обеспечением, которое следует использовать абсолютно для всех случаев жизни, потому его необходимо «подвязывать» в рамках операционной системы только к конкретному файловому менеджеру, а не для всех текстовых файлов операционной системы, предустановленной на персональном компьютере.

В качестве удобного файлового менеджера последовательно рассматриваются «*Total Commander*» и «*Double Commander*» (его бесплатный аналог).

Для начала о «*Total Commander*». В нём для редактирования текстовых файлов по нажатии на горячую клавишу «*F4*» встроен обыкновенный блокнот *Microsoft Windows* («*Notepad.exe*»).

При этом расширенная, администраторская сборка «*Total Commander*» содержит привязку по этой клавише более удобного блокнота, – блокнота второго поколения («..*Soft**Notepad2**Notepad.exe*»). Собственно, это обстоятельство и послужило идеей для расширения программного обеспечения рассматриваемого файлового менеджера до «*Notepad++*», поскольку «*Notepad++*» обладает куда большим функционалом, нежели «*Notepad2.0*».

Если открыть диалоговое окно конфигурации администраторской версии файлового менеджера «*Total Commander*», то там прописан следующий путь: «***%commander_path%\Soft\Notepad2\Notepad.exe***». Хорошо видно, что блокнот второго поколения – это плагин (подключаемая надстройка, специальная подключаемая подпрограмма, адаптированная под «*Total Commander*»).

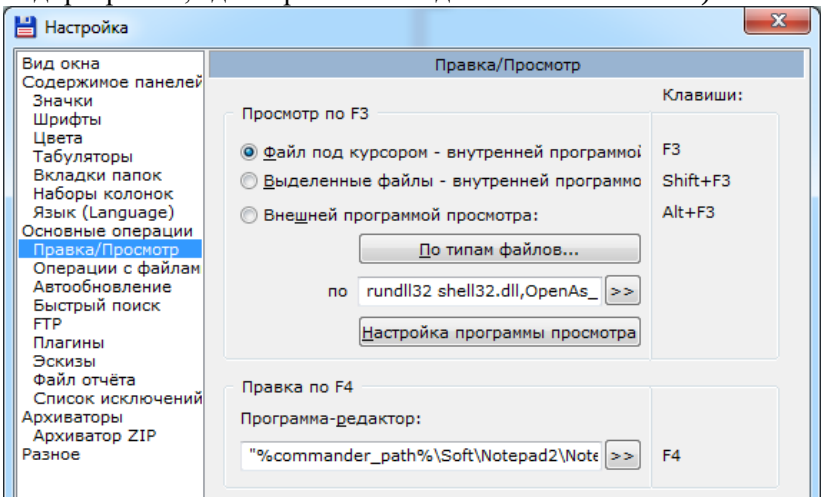


Рисунок 2.1 – Диалоговое окно конфигурации файлового менеджера «*Total Commander*». Раздел «Правка/просмотр»

В блокноте второго поколения гипертекстовые теги, их параметры, а также комментарии расцвечены для удобства их восприятия разработчиком. Эта функция называется маркировкой синтаксиса. Блокнот второго поколения предоставляет возможность маркировать не только *HTML*, но и *Java*, *SQL*, *C#*, *Delphi* и синтаксисы многих других языков, известных на сегодняшний день. Значимым удобством является тот факт, что строки кода в блокноте пронумерованы – это существенно упрощает процедуру отладки программного обеспечения.

Отладка браузерного веб-приложения крайне важный и непростой этап разработки, позволяющий учесть все детали и витиеватую логику работы программного обеспечения. Ведётся отладка в собственной консоли браузера, где отлов ошибок

выполняется по номерам строк, в которых произошла та или иная ошибка. Таким образом подобный инструмент как блокнот с нумерованными строками в арсенале любого веб-программиста просто незаменим. Сравнение блокнотов выполнено на Рисунках 2.2 и 2.3

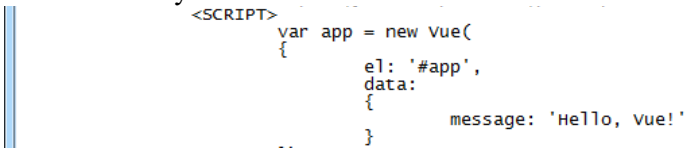


Рисунок 2.2 – Фрагмент кода, просматриваемого через встроенный блокнот *Microsoft Windows*

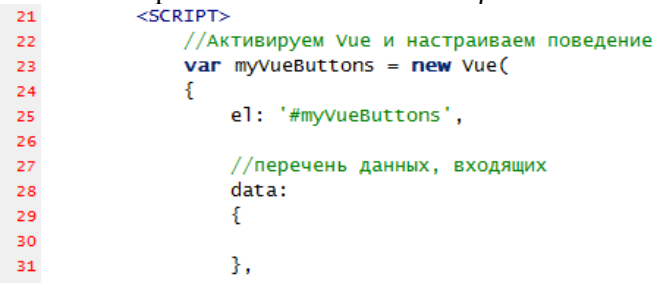


Рисунок 2.3 – Фрагмент кода, просматриваемого через блокнот второго поколения администраторской версии файлового менеджера «*Total Commander*»

Далее раздел методических указаний посвящён рассмотрению рекомендаций по установке упомянутых программных продуктов. Их можно достать из надёжных источников и без лишних проблем (вирусов).

Стоит отметить, что часть программного обеспечения останется условно открытой (не зарегистрированной и, соответственно, не оплаченной). Однако, самое страшное, чем придётся пожертвовать в этом случае – это временем и вниманием: при каждом открытии файлового менеджера потребуется нажимать на экранную кнопку с правильным номером. Речь здесь идёт только о специфике работы с «*Total Commander*».

Как правило «*Total Commander*» не входит в базовый перечень программ, которыми обязательно должен обладать программист.

С высокой вероятностью, если «*Total Commander*» и предустановлен в специализированной сборке *Microsoft Windows* (как, например, в аудиториях кафедры «УиЗИ»), то он вряд ли расширенный, то есть администраторский. Обычно это – базовый «*Total Commander*» и вполне достаточно иметь в наличии именно такую его версию. «*Total Commander*» можно скачать с сайта разработчика ***https://www.ghisler.com/***, хотя, безусловно, удобнее сразу переходить на страницу загрузки приложений: ***https://www.ghisler.com/download.htm***.

На момент составления методических указаний актуальной версией оказался «*Total Commander 9.21a*». Скачивать, к слову, всегда необходимо новейшую из имеющихся сборок и версий.

Из рассмотрения в методических указаниях намеренно опущен этап установки файлового менеджера, поскольку его установка является достаточно простой и типовой процедурой (как установка большинства программных продуктов, проводимая через пошаговый интерфейс). Дальнейшее рассмотрение ведётся с этапа запуска файлового менеджера.

Как ранее было замечено, программное обеспечение не зарегистрировано, но вполне работоспособно и полнофункционально. Таковым и именно в таком виде можно пользоваться на кафедре. В рассмотренном на Рисунке 2.4 примере для продолжения работы следует нажать на кнопку «2» (нужная указана в строке над кнопками «1», «2», «3», заключёнными между символиками платёжных систем «*MasterCard*» и «*Visa*»). В зарегистрированной версии «*Total Commander*» это окно выдаваться при каждом запуске не будет.

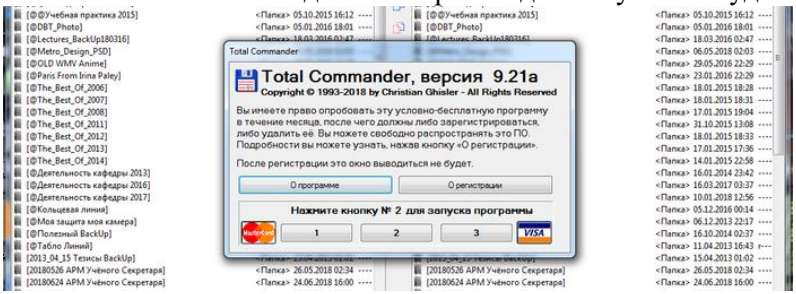


Рисунок 2.4 – Первый запуск файлового менеджера «*Total Commander*» без регистрации и оплаты

По нажатии на клавишу «*F4*» во время расположения курсора (рамки) менеджера на любом текстовом файле будет открыт обычный блокнот с текстовым содержимым файла (Рисунок 2.5).

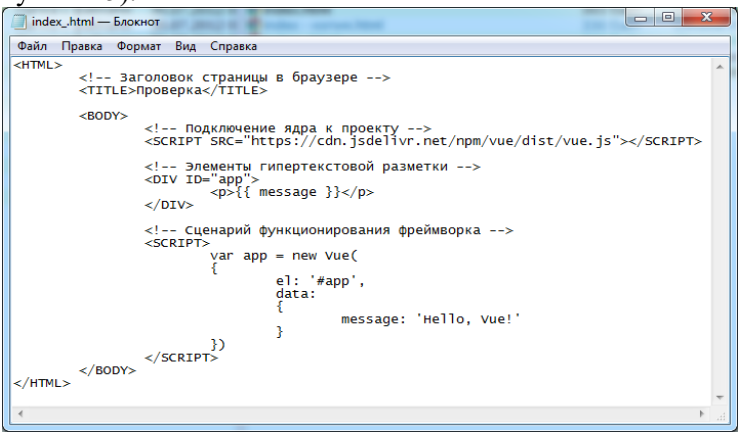


Рисунок 2.5 – Содержимое файла, просматриваемое через обычный блокнот

На следующем шаге необходимо скачать «*Notepad++*». У программного обеспечения тоже есть свой стабильный и безопасный сайт: ***https://notepad-plus-plus.org***. И, опять же, удобно переходить сразу на прямую страницу загрузки приложения из Интернет: ***https://notepad-plus-plus.org/download/v7.5.8.html***. Следует обратить внимание на то, что последняя актуальна только для загрузки версии 7.5.8.

На момент составления методических указаний актуальной и наиболее свежей была версия 7.5.8. Дальнейшая работа по изучению «*Vue.js*» связана именно с этой версией «*Notepad++*». Установка данного программного обеспечения такая же типовая, проводимая через пошаговый интерфейс без лишних сложностей. Она так же опущена из рассмотрения в данных методических указаниях. Итогом установки является открытие окна программного обеспечения со вкладкой, содержащей основные исправления конкретной версии «*Notepad++*», как это показано на Рисунке 2.6.

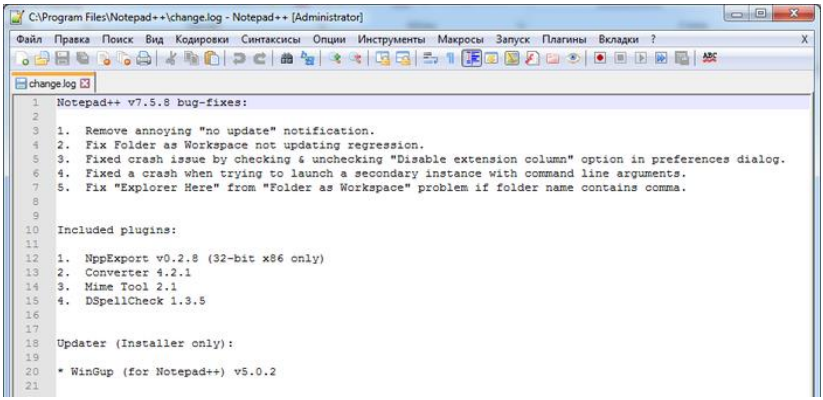


Рисунок 2.6 – Первый запуск расширенного блокнота «*Notepad++*»

Дальнейшее рассмотрение вновь связано с файловым менеджером «*Total Commander*». В его главном меню необходимо выбрать пункт «Конфигурация» и в нём – пункт подменю «Настройка...» (он первый в списке, что и показано на Рисунке 2.7).

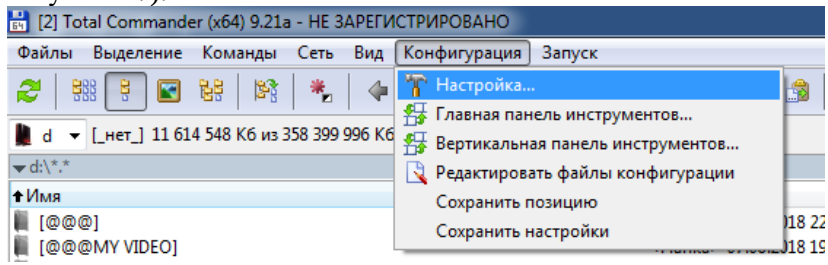


Рисунок 2.7 – Переход к основным настройкам через главное меню файлового менеджера «*Total Commander*»

В раскрывшемся диалоговом окне необходимо найти и выбрать из списка, расположенного слева, группу настроек «Правка/просмотр» (Рисунок 2.8).

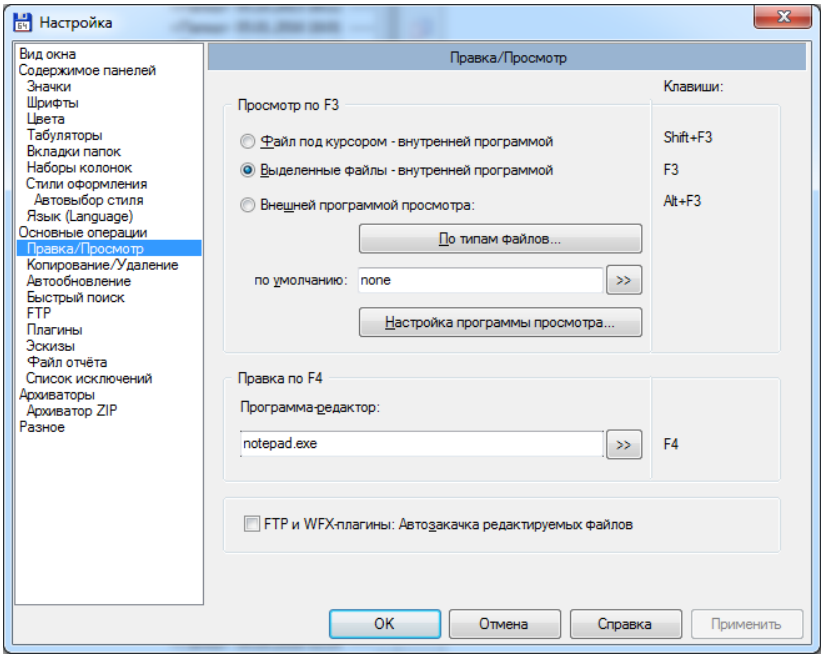


Рисунок 2.8 – Группа настроек «Правка/просмотр» в разделе настроек файлового менеджера «*Total Commander*»

Интерес в этой группе представляет перечень компонентов, объединённых под общим заголовком «Правка по *F4*», внутри которого указан подзаголовок «Программа-редактор:». Под ним размещено текстовое поле, в котором указывается исполняемый файл программы, вызываемой по умолчанию по нажатии пользователем на клавишу «*F4*» клавиатуры.

Стандартный «*notepad.exe*» необходимо заменить на «*notepad++.exe*». Сделать это необходимо не вручную, набором строки с клавиатуры, а через специальный диалог выбора файла. Для вызова этого диалога необходимо нажать на кнопку с изображённой на ней двойной стрелкой «>>», расположенную справа от поля для ввода текста.

В диалоговом окне выбора файла следует указать полный путь к исполняемому файлу «*notepad++.exe*» после чего подтвердить выбор нажатием на кнопку «Открыть» (Рисунок 2.9).

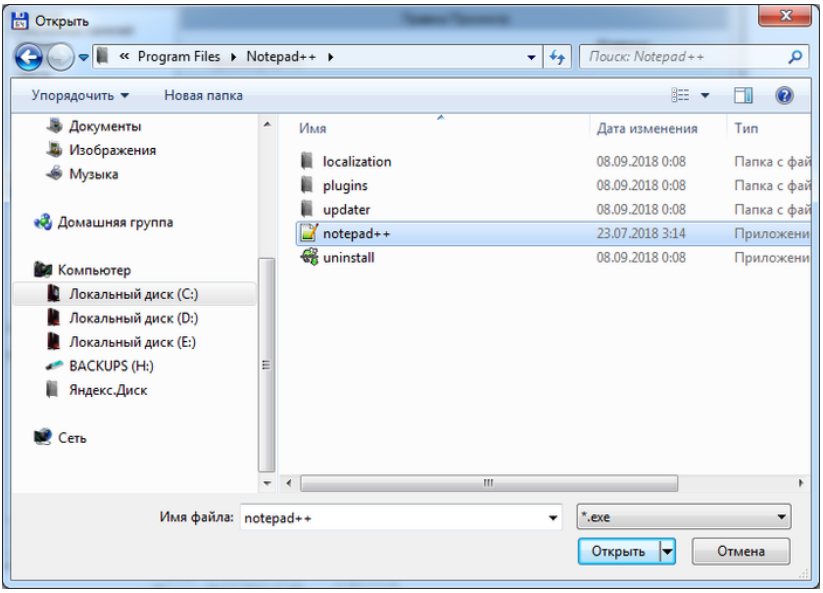


Рисунок 2.9 – Диалоговое окно открытия файла для выбора программы «*notepad++.exe*»

Строка «*notepad.exe*» автоматически изменится на путь к «*notepad++.exe*», который в рассматриваемом случае записан полностью как «*C:\Program Files\Notepad++\notepad++.exe*» «*%1*» (Рисунок 2.10).

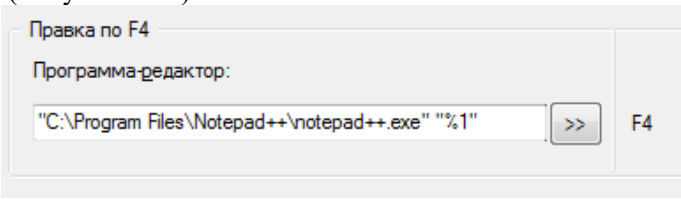


Рисунок 2.10 – Результат изменения программы для просмотра и редактирования текстовых файлов, заданной в файловом менеджере «*Total Commander*» по умолчанию

В дальнейшем по нажатии на клавишу «*F4*» будет вызываться «*Notepad++*». Он выглядит куда более функциональным, по сравнению с блокнотом второго поколения «*Notepad2.0*». Следует отметить, что фрагменты кода в «*Notepad++*» можно сворачивать и, тем самым, не

просматривать лишнее в процессе отладки. Вместе с тем в «*Notepad++*» присутствуют вертикальные отступы, которые делают (и/или заставляют составителя программы делать) код более читаемым и удобным для восприятия. Коды в блокнотах второго поколения и «*Notepad++*» сопоставлены на Рисунках 2.11 и 2.12.

```
21          <SCRIPT>
22              //Активируем Vue и настраиваем поведение
23              var myVueButtons = new Vue(
24              {
25                  el: '#myVueButtons',
26
27                  //перечень данных, входящих
28                  data:
29                  {
30
31                  },
```

Рисунок 2.11 – Фрагмент кода в блокноте второго поколения «*Notepad2.0*»

Рисунок 2.12 – Фрагмент кода в блокноте «*Notepad++*»

Для сторонников «честного программного обеспечения» существует бесплатный аналог файлового менеджера «*Total Commander*» под названием «*Double Commander*». Сайтом данного программного продукта является: ***https://doublecmd.sourceforge.io/***, а непосредственно его загрузку можно выполнить со страницы: ***https://sourceforge.net/p/doublecmd/wiki/Download/***.

На момент составления методических указаний наиболее свежей являлась версия: ***doublecmd-0.9.6.x86_64-win64.exe***.

В начальный момент после установки и запуска приложения пользователям, привыкшим работать в «*Total Commander*», интерфейс «*Double Commander*» может показаться

непривычным и громоздким (Рисунок 2.13), но должный вид в нём достаточно гибко и быстро настраивается.

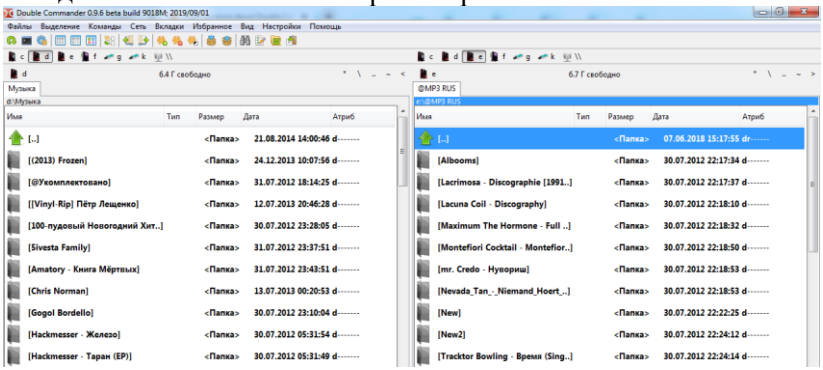


Рисунок 2.13 – Исходное представление файлового менеджера «*Double Commander*»

Для приведения файлового менеджера в порядок необходимо перейти через главное меню в пункт «Настройки» и оттуда через подменю в «Параметры…» (Рисунок 2.14).

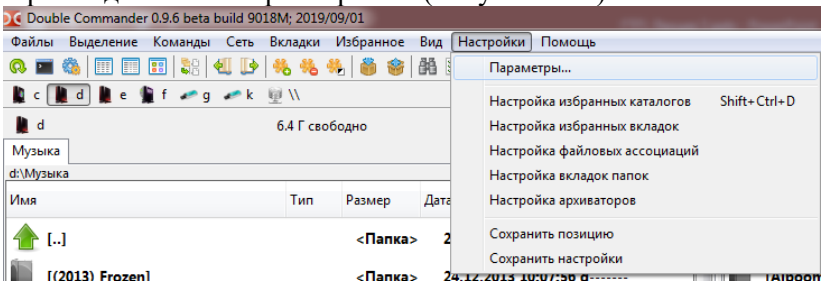


Рисунок 2.14 – Переход через меню файлового менеджера «*Double Commander*» к настройке его основных параметров

Прежде всего непривычным является размер значков (иконок). Их следует переключить с размера 32x32 на размер 16x16, для чего в списке, расположенном слева, выбрать группу настроек «Значки», а в самой группе найти блок «Размер значков» и напротив надписи «Панель файлов:» из комбинированного списка выбрать размер 16x16 как показано на Рисунке 2.15. Настройки подтвердить нажатием на кнопку «*OK*» с зелёной галочкой, стоящей слева.

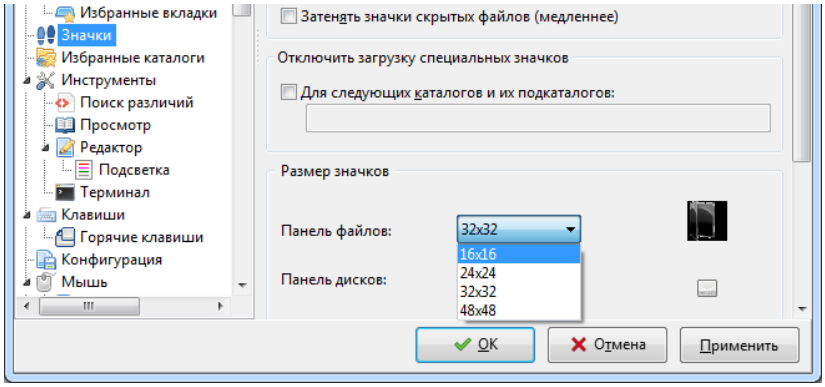


Рисунок 2.15 – Изменение стандартного размера иконок, стоящих напротив имён файлов

При последующем сопоставлении двух файловых менеджеров можно убедиться, что они довольно сильно становятся похожи друг на друга (Рисунки 2.16 и 2.17).

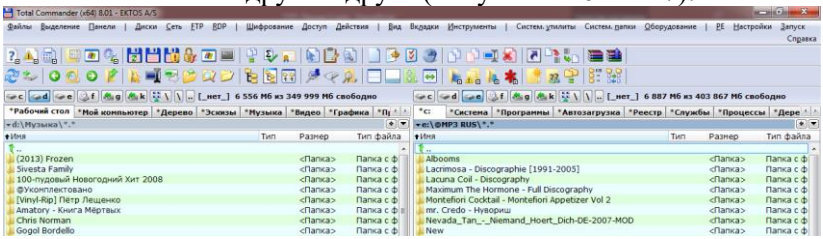


Рисунок 2.16 – Отображение структуры каталогов в файловом менеджере «*Total Commander*»

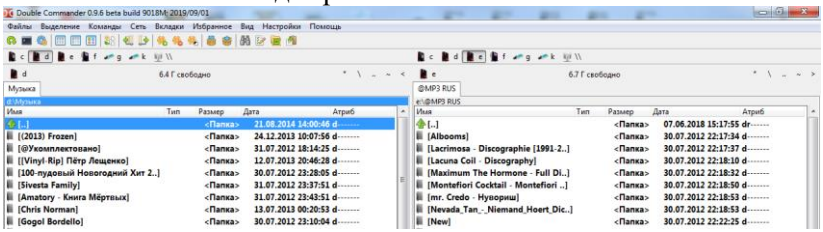


Рисунок 2.17 – Отображение структуры каталогов в файловом менеджере «*Double Commander*»

В «*Double Commander*» работа клавиши «*F4*» аналогична её работе в «*Total Commander*», но встроенным в нём является более продвинутый блокнот-редактор, нежели даже «*Notepad2.0*». Однако, всё равно этот блокнот является менее

функциональным и удобным по сравнению с «*Notepad++*» (Рисунок 2.18).

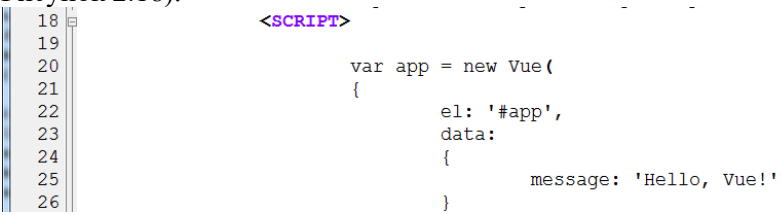


Рисунок 2.18 – Встроенный блокнот файлового менеджера «*Double Commander*»

Во встроенном блокноте присутствует возможность прямого перехода к настройкам редактора через его собственное меню «Файл» (Рисунок 2.19). Таким образом, довольно быстро можно заменить его на какую-либо иную внешнюю программу, то есть в рассматриваемом случае на «*Notepad++*».

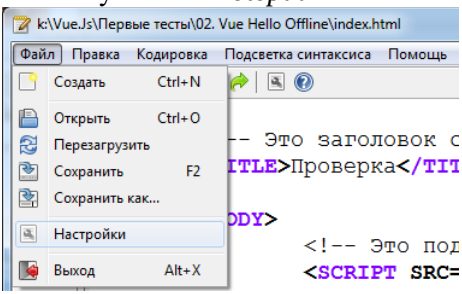


Рисунок 2.19 – Переход к настройкам встроенного блокнота файлового менеджера «*Double Commander*» через его меню «Файл»

Настройки встроенного редактора автоматически выставляют выбор пункта «Редактор» в списке-перечне (размещённом слева) возможных настроек файлового менеджера «*Double Commander*». По умолчанию выбор сторонней программы отключён, но эта функция может быть активирована пользователем посредством выставления галочки-флажка напротив пункта «Использовать внешнюю программу» (Рисунок 2.20).

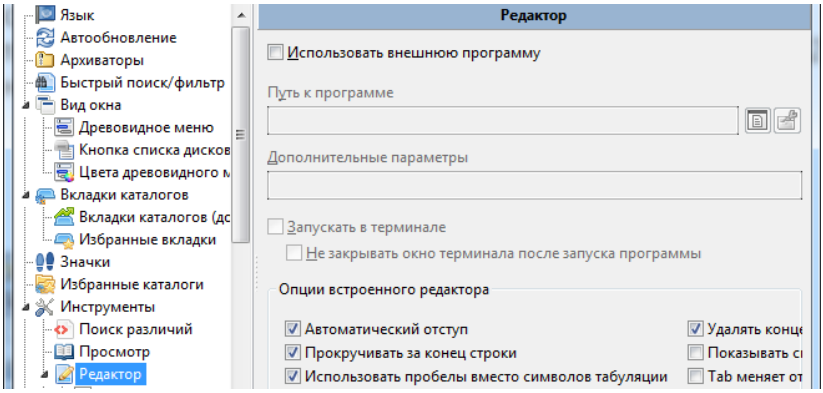


Рисунок 2.20 – Исходное состояние группы настроек файлового менеджера «*Double Commander*», относящихся ко встроенному текстовому редактору

После выявления всех отмеченных возможностей «*Double Commander*» можно смело считать достойной молодой заменой старому-доброму файловому менеджеру «*Total Commander*», ведь «*Notepad++*» подключается к нему аналогичным образом (Рисунок 2.21).

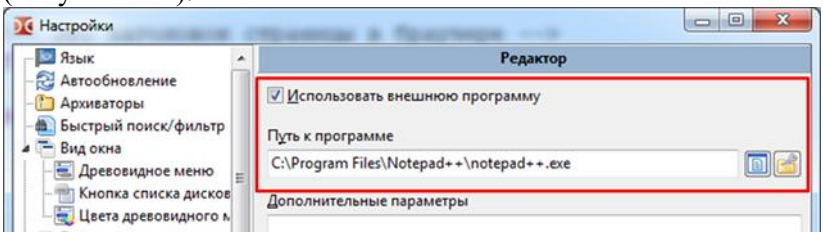


Рисунок 2.21 – Подключение к файловому менеджеру «*Double Commander*» расширенного блокнота «*Notepad++*»

На данном этапе все удобства для локальной работы с фреймворком *Vue.js* настроены и в последующих разделах методических указаний они будут активно использованы при рассмотрении более сложных конструкций, нежели типы вычислительных процессов.

3 Разветвляющийся вычислительный процесс на базе фреймворка *Vue.js*

3.1 Условная отрисовка шаблонов гипертекстовой разметки на базе условного оператора (директивы *v-if, v-else*)

На сайте официальной документации *Vue.js* [8] составлен целый раздел, посвящённый условной отрисовке гипертекстовых блоков. Но следует остановиться и подробнее рассмотреть наиболее показательный из представленных там примеров (Рисунок 3.1.1).

```HTML
<template v-if="loginType === 'username'">
  <label>Имя пользователя</label>
  <input placeholder="Введите имя пользователя">
</template>
<template v-else>
  <label>Email</label>
  <input placeholder="Введите адрес email">
</template>
```

Рисунок 3.1.1 – Пример условной отрисовки шаблонов из официальной документации

В нём в гипертекстовой части размечены два шаблона, заключённые между парой тегов «*template*» (шаблон). В любой момент времени будет отображён один и только один из двух шаблонов (Рисунок 3.1.2). Оба шаблона одновременно отображены не будут никогда в связи с тем, что условие строго будет либо выполняться, либо нет. Именно так работает разветвляющийся вычислительный процесс, когда в зависимости от выполнения или не выполнения условий выполняются операторы только соответствующей ветви алгоритма. Опять же, важно, что одной и только одной ветви.

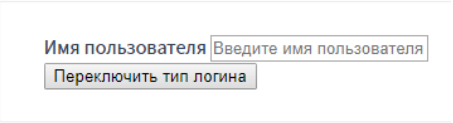


Рисунок 3.1.2 – Ожидаемый результат работы примера с условной отрисовкой

И если изучать материал из официальной документации непоследовательно, то может показаться, что для получения представленного результата вполне достаточно размещённого в примере кода, но это далеко не так. Далее по тексту методических указаний выполняется проверка данного тезиса.

Итак, в блокноте прописан согласно примеру (Рисунок 3.1.1) код страницы без каких-либо сценариев, но при наличии ссылки-подключения локального сценария фреймворка *Vue.js* (Рисунок 3.1.3).

```
<HTML>
	<!-- Это заголовок страницы в браузере -->
	<TITLE>Проверка</TITLE>

	<BODY>
		<!-- Это подключение ядра к проекту -->
		<SCRIPT SRC="Vue.js"></SCRIPT>

		<template v-if="loginType === 'username'">
			<label>Имя пользователя</label>
			<input placeholder="Введите имя пользователя">
		</template>

		<template v-else>
			<label>Email</label>
			<input placeholder="Введите адрес email">
		</template>

	</BODY>

</HTML>
```

Рисунок 3.1.3 – Набранный в блокноте код примера условной отрисовки

На выходе получена пустая страница (Рисунок 3.1.4).

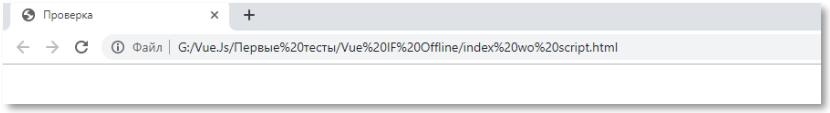


Рисунок 3.1.4 – Показано, что представленного фрагмента *HTML*-кода недостаточно для нормального функционирования примера условной отрисовки из официальной документации

В попытках оживления изучаемой структуры условной отрисовки в код добавляется новый объект *Vue.js* и его базовое наполнение согласно материалам, представленным в разделе 1 данных методических указаний (Рисунок 3.1.5).

```
<template v-else>
        <label>Email</label>
        <input placeholder="Введите адрес email">
</template>

<!-- Сценарий функционирования фреймворка -->
<SCRIPT>
        var appVue = new Vue(
        {
                el: '#app',
                data:
                {
                        message: '12345'
                }
        })
</SCRIPT>
</BODY>
```

Рисунок 3.1.5 – Добавление объекта *Vue.js* к рассматриваемому примеру условной отрисовки

На выходе снова получается пустая страница (Рисунок 3.1.6), поскольку объект фреймворка *Vue.js* хоть и создан, но оказался ни с чем не связанным.

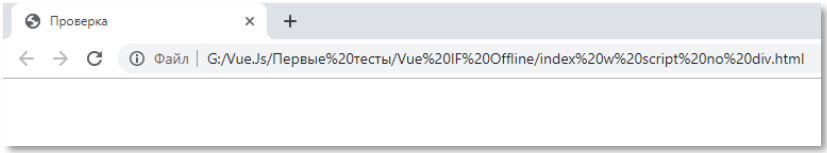


Рисунок 3.1.6 – Результат запуска приложения с условной отрисовкой при записанном сценарии создания объекта фреймворка *Vue.js*

Далее выполняется добавление недостающего компонента – связки. Код, содержащий два шаблона из рассматриваемого примера (Рисунок 3.1.1), оборачивается в блок <*DIV*>, связанный с объектом *Vue.js* по идентификатору элемента «*app*» (Рисунок 3.1.7).

```
<!-- Это подключение ядра к проекту -->
<SCRIPT SRC="Vue.js"></SCRIPT>

<DIV ID="app">
        <template v-if="loginType === 'username'">
                <label>Имя пользователя</label>
                <input placeholder="Введите имя пользователя">
        </template>

        <template v-else>
                <label>Email</label>
                <input placeholder="Введите адрес email">
        </template>
</DIV>

<!-- Сценарий функционирования фреймворка -->
<SCRIPT>
```

Рисунок 3.1.7 – Обёртывание шаблонов в тег *div*, связанный с фреймворком *Vue.js*

Содержимое на странице, наконец, отобразилось (Рисунок 3.1.8) и этот факт – прямое свидетельство того, что теги «*template*» являются элементами фреймворка *Vue.js*, а не элементами гипертекстовой разметки *HTML*. Без фреймворка они не работают. Вместе с тем получен иной показательный результат – отобразился шаблон из блока ложного результата условного оператора, полученный согласно директиве «*v-else*» (Рисунок 3.1.8). Дело в том, что в написанном коде, как минимум, отсутствует переменная «*loginType*», которая, как максимум, проверяется на соответствие строковой константе «*username*». Значение «*loginType*» в отсутствие этой переменной равняется пустой строке (««»). Пустая строка, что естественно, – это не «*username*», а значит условие не выполняется и вычислительный процесс идёт согласно директиве *v-else*, а не *v-if*.

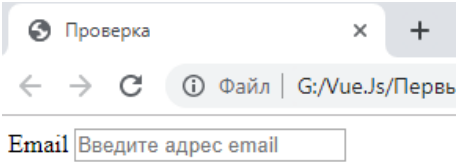


Рисунок 3.1.8 – Результат отображения шаблона согласно директиве *v-else*

Таким образом на следующем шаге в рассматриваемом одностраничном приложении следует выстроить управление с использованием заявленной переменной. К объекту *Vue.js* в раздел исходных данных добавляется управляющая переменная «*loginType*» с присвоенным ей по умолчанию значением «*username*» (Рисунок 3.1.9).

```
<!-- Сценарий функционирования фреймворка -->
<SCRIPT>

        var appVue = new Vue(
        {
                el: '#app',
                data:
                {
                        loginType: 'username'
                }
        })
</SCRIPT>
```

Рисунок 3.1.9 – Добавление в объект *Vue.js* переменной с интересующим значением, заданным по умолчанию

На Рисунке 3.1.10 получен ожидаемый результат. На страницу выведен шаблон, полученный согласно директиве *v-if*. Что и требовалось реализовать.

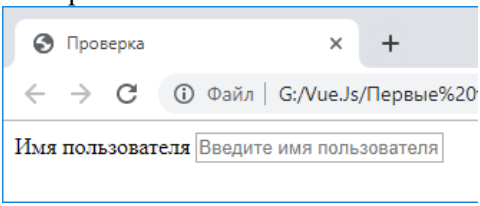


Рисунок 3.1.10 – Результат отображения шаблона согласно директиве *v-if*

3.2 Связь элементов гипертекстовой разметки с событиями (директива *v-on*)

Данный параграф призван изложить методику, согласно которой можно в точности повторить результат, продемонстрированный на Рисунке 3.1.2. Созданное в рамках параграфа 3.1 одностраничное приложение должно получить развитие.

Для начала необходимо добавить на графический пользовательский интерфейс кнопку без какого-либо функционала, подключённого к ней (Рисунок 3.2.1).

```
15          <template v-else>
16              <label>Email</label><br>
17              <input placeholder="Введите адрес email"><br>
18          </template>
19
20          <button>Изменить режим ввода</button>
21      </DIV>
22
23      <!-- Сценарий функционирования фреймворка -->
24      <SCRIPT>
25          var appVue = new Vue(
26          {
27              el: '#app',
28              data:
29              {
30                  loginType: 'username'
31              }
32          })
33      </SCRIPT>
```

Рисунок 3.2.1 – Добавление к примеру кода для отображения кнопки

На Рисунке 3.2.2 показано, что кнопка отобразилась, согласно заявленному плану работ.

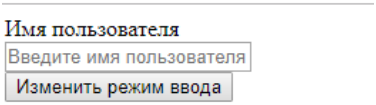


Рисунок 3.2.2 – Отображение кнопки «Изменить режим ввода»

Кнопка должна присутствовать на экране вне зависимости от принятого решения, поскольку она размещена вне кода какого-либо из шаблонов.

Для назначения события на кнопку необходимо воспользоваться директивой фрейморка *Vue.js* «*v-on:*» с указанием далее наименования подключаемого события «*click*» (нажатие кнопки мыши по экранной кнопке). Вслед за этой конструкцией записывается знак «равно» и в кавычках указывается наименование метода, который должен быть вызван по нажатии на кнопку мыши (в данном случае, как показано на Рисунке 3.2.3, это метод «*changeReg*»).

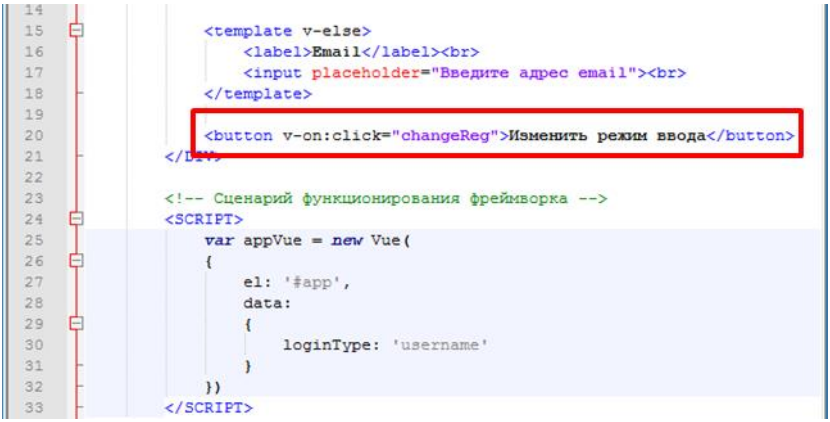


Рисунок 3.2.3 – Подключение обработки события к экранной кнопке по нажатии на неё кнопкой мыши согласно правилам фреймворка *Vue.js*

Речь зашла о программном методе (процедуре), потому в объекте фреймворка *Vue.js* должен появиться раздел, позволяющий создавать в нём новые программные методы. Раздел может быть организован вслед за разделом объявления исходных данных (*data*). Начинается раздел методов со служебного слова «*methods*». Важно не забыть поставить запятую «,» после завершения раздела для объявления исходных данных (Рисунок 3.2.4).

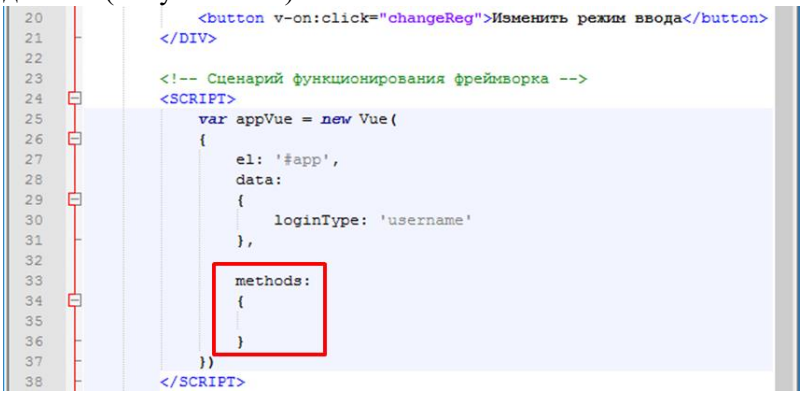


Рисунок 3.2.4 – Подключение к объекту фреймворка *Vue.js* раздела пользовательских методов (процедур)

В разделе объявления методов для тестирования срабатывания нажатия на экранную кнопку создаём тело упомянутого ранее метода «*changeReg*», который будет сбрасывать значение переменной «*loginType*» к пустой строке («»). **Внимание!** Без указания ссылки на объект фреймворка *Vue.js* через служебное слово *this* обращение к переменной «*loginType*» произведено не будет.

```
20          <button v-on:click="changeReg">Изменить режим ввода</button>
21      </DIV>
22
23      <!-- Сценарий функционирования фреймворка -->
24      <SCRIPT>
25          var appVue = new Vue(
26          {
27              el: '#app',
28              data:
29              {
30                  loginType: 'username'
31              },
32
33              methods:
34              {
35                  changeReg: function()
36                  {
37                      this.loginType = ''
38                  }
39              }
40          })
41      </SCRIPT>
```

Рисунок 3.2.5 – Создание метода «*changeReg*», реализующего сброс значения, хранимого в переменной «*loginType*»

На Рисунке 3.2.6 выполнена проверка реакции графического пользовательского интерфейса на нажатие кнопки мыши по экранной кнопке. Выполнен тест обработки события.

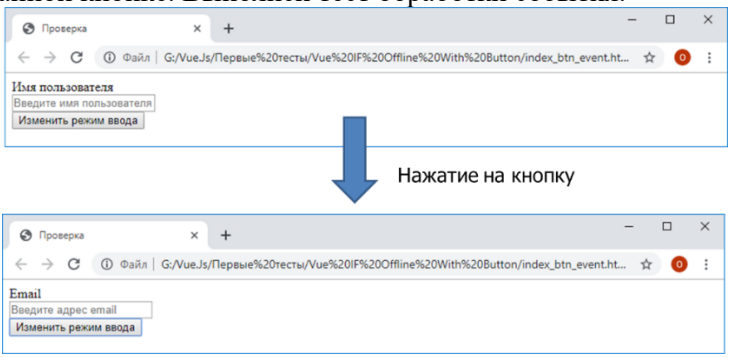


Рисунок 3.2.6 – Тест обработки события, связанного с нажатием на кнопку мыши

Для краткости записи директиву «*v-on:*» можно заменить символом «@». Результат работы одностраничного приложения при этом остаётся неизменным.

```
20          <button @click="changeReg">Изменить режим ввода</button>
21      </DIV>
22
23      <!-- Сценарий функционирования фреймворка -->
24      <SCRIPT>
25          var appVue = new Vue(
26          {
27              el: '#app',
28              data:
29              {
30                  loginType: 'username'
31              },
```

Рисунок 3.2.7 – Изменение директивы связи интерфейсного элемента управления с событием на «ленивую» форму записи

В рассмотренном примере кнопка работает только на сброс значения переменной. Качественной функциональности одностраничного приложения это, безусловно не обеспечивает. Необходимо разветвить вычислительный процесс таким образом, чтобы каждое последующее нажатие на кнопку меняло состояние одностраничного приложения (Рисунок 3.2.8).

```
32
33              methods:
34              {
35                  changeReg: function()
36                  {
37                      if (this.loginType == 'username')
38                      {
39                          this.loginType = ''
40                      }
41                      else
42                      {
43                          this.loginType = 'username'
44                      }
45                  }
46              }
47          })
48      </SCRIPT>
```

Рисунок 3.2.8 – Изменение метода «*changeReg*» под последовательную смену имеющихся состояний (каждое нажатие переключает отображаемые шаблоны согласно директивам *v-if* и *v-else*)

Элементы меняют своё состояние за счёт расположения в теле различных шаблонов. Сама кнопка своего состояния не меняет, хотя для порядка должна это делать.

Поскольку кнопка размещена на *html*-странице как статичный компонент и, к тому же, компонент не снабжённый

уникальным идентификатором (или связкой с переменной через директиву *v-model*), то до неё необходимо достучаться через *DOM*. Первым приближением к этому может служить добавление ссылки на событие в качестве параметра вызываемого метода. Для этого нужно дописать переменную «*e*» (первая буква служебной директивы *$event* – событие) в заглавие метода «*changeReg*».

3.3 Настройка интерфейсных элементов управления через *DOM*

DOM (от англ. *Document Object Model* – «объектная модель документа») – это не зависящий от платформы и языка программный интерфейс, позволяющий программам и сценариям получить доступ к содержимому:

- *HTML-*,
- *XHTML-*,
- *XML-*

документов, а также изменять содержимое, структуру и оформление таких документов [10].

Модель *DOM* не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью *DOM* может быть представлен в виде дерева узлов, каждый узел которого представляет собой:

- элемент,
- атрибут,
- текстовый,
- графический,
- любой другой объект.

Узлы связаны между собой отношениями «родительский-дочерний». Изначально различные браузеры имели собственные модели документов (*DOM*), несовместимые с остальными. Для обеспечения взаимной и обратной совместимости специалисты международного консорциума *W3C* классифицировали эту модель по уровням, для каждого из которых была создана своя

спецификация. Все эти спецификации объединены в общую группу, носящую название «*W3C DOM*».

Исходя из вышесказанного статические компоненты записываются в рамках одностраничного приложения как узлы *DOM*, что означает, что ими тоже можно управлять без подключения через фреймворк.

Далее представлен подход по управлению статичным интерфейсным элементом (кнопкой) через восходящее рассуждение, то есть от произошедшего события к связанному с этим событием интерфейсному элементу управления.

В предыдущем параграфе отмечен подход по направлению ссылки на событие в метод через переменную «*e*». Это реализовано на Рисунке 3.3.1.

```
33                  methods:
34                  {
35                      changeReg: function(e)
36                      {
37                          if (this.loginType == 'username')
38                          {
39                              this.loginType = ''
40                          }
41                          else
42                          {
43                              this.loginType = 'username'
44                          }
45                      }
46                  }
47              })
48          </SCRIPT>
```

Рисунок 3.3.1 – Дополнение метода «*changeReg*» ссылкой на событие «*e*»

Для просмотра узлов, содержащихся в «*e*» следует воспользоваться браузерной консолью (Рисунок 3.3.3). Умение обращения с браузерной консолью существенно упрощает понимание механизмов *Vue.js*, а также ускоряет отладку в ходе разработки одностраничных веб-приложений. Но прежде, чем начать работу с консолью необходимо дополнить тело метода строкой вида *console.log(e)* (журнализация/аудит события в консоли). Именно она и будет поставщиком структуры события согласно *DOM* в консоль (Рисунок 3.3.3).

```
35                      changeReg: function(e)
36                      {
37                          console.log(e)
38                          if (this.loginType == 'username')
```

Рисунок 3.3.2 – Добавление строки аудита для передачи структуры события в консоль браузера

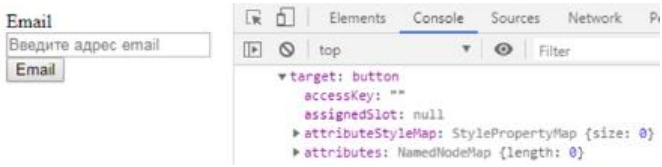
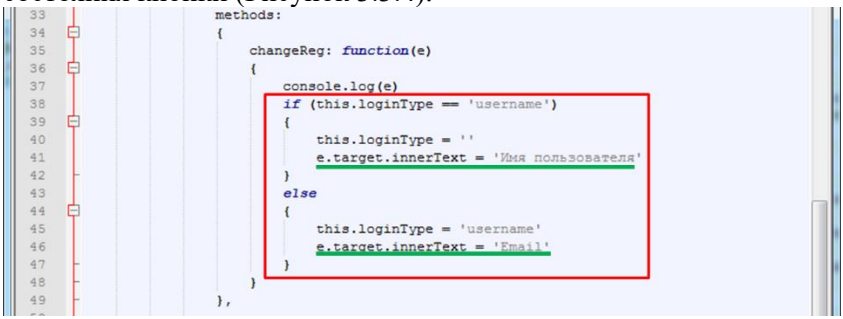


Рисунок 3.3.3 – Просмотр структуры события «*e*» через консоль

Из консоли можно определить, что элемент управления «кнопка» является дочерним узлом события, записанным под именем «*target*» (целевой объект события). Через целевой объект можно найти его поля, соответствующие надписи, нанесённой на кнопку, например, «*innerText*» (по опознавательной надписи «*Email*», как показано на Рисунке 3.3.6). Эту надпись можно поменять, присвоив ей другое строковое значение.

Таким образом, дополнив ранее составленную кодовую конструкцию соответствующими строками управления, можно добиться изменения состояния, в том числе, и изменения состояния кнопки (Рисунок 3.3.4).



```
33          methods:
34          {
35              changeReg: function(e)
36              {
37                  console.log(e)
38                  if (this.loginType == 'username')
39                  {
40                      this.loginType = ''
41                      e.target.innerText = 'Имя пользователя'
42                  }
43                  else
44                  {
45                      this.loginType = 'username'
46                      e.target.innerText = 'Email'
47                  }
48              }
49          },
```

Рисунок 3.3.4 – Изменение состояния кнопки через восходящее рассуждение по *DOM* события, связанного с этой кнопкой

Далее на Рисунке 3.3.5 показано поведение интерфейсных элементов управления одностраничного веб-приложения в результате нажатия на кнопку «*Email*».

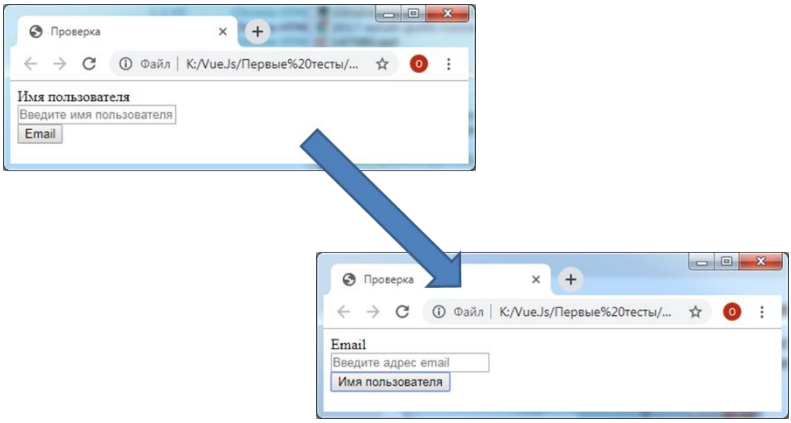


Рисунок 3.3.5 – Изменение состояния интерфейса одностраничного веб-приложения в результате нажатия на кнопку «*Email*»

Среди прочих и крайне полезных для управления полей стоит выделить «*disabled*» (заблокировано) и «*hidden*» (скрыто). Имеется занятное наблюдение, что они существуют в инверсной логике по отношению к *Microsoft Visual C#*, где аналогичные свойства называются «*enabled*» (доступно) и «*visible*» (видимо).

Рисунок 3.3.6 – Структура *DOM* события, связанного с кнопкой

Далее выполнена проверка работоспособности одного из указанных свойств. На Рисунках 3.3.7 и 3.3.8 выполняется блокировка кнопки после очередной смены её состояния.

```
35                  changeReg: function(e)
36                  {
37                      console.log(e)
38                      if (this.loginType == 'username')
39                      {
40                          this.loginType = ''
41                          e.target.innerText = 'Имя пользователя'
42                      }
43                      else
44                      {
45                          e.target.innerText = 'Email'
46                          e.target.disabled = true
47                          this.loginType = 'username'
48
49                      }
```

Рисунок 3.3.7 – Дополнение кода одностраничного приложения операцией блокировки кнопки после возврата ко вводу имени пользователя

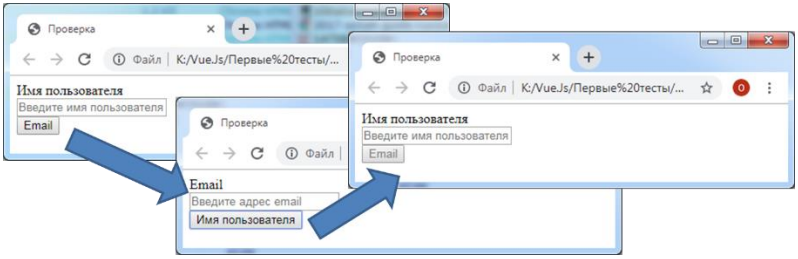


Рисунок 3.3.8 – Демонстрация смены состояний интерфейса при добавленной операции блокировки кнопки

Помимо рассмотренных директив *v-if* и *v-else*, необходимых для условной отрисовки шаблонов гипертекстовой разметки, существует и директива *v-else-if* для одностраничных веб-приложений с более, чем парой возможных состояний. Эта директива рассматривается отдельно в следующем параграфе.

3.4 Условная отрисовка шаблонов гипертекстовой разметки на базе оператора переключения (*v-else-if*, *switch*)

Созданное одностраничное приложение (Рисунок 3.3.5) подвергается ещё одной модификации. К нему добавляется третье состояние – третий шаблон для ввода пароля (Рисунок 3.4.1), для чего потребуется использование директивы «*v-else-if*».

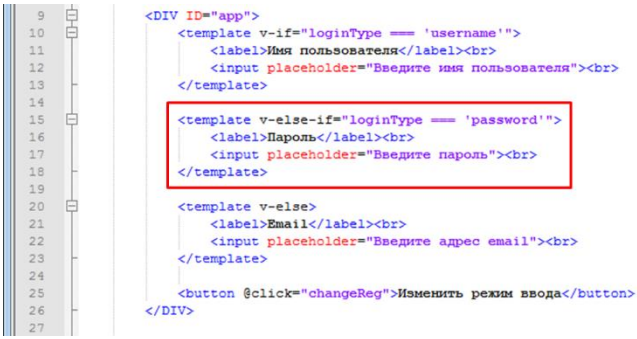


Рисунок 3.4.1 – Добавление шаблона для ввода пароля в ранее созданное одностраничное веб-приложение

Вместе с тем уместно использовать в сценарном блоке уже не условный оператор, а оператор переключения *switch* (Рисунок 3.4.2).

Рисунок 3.4.2 – Изменение содержимого метода «*changeReg*» под три различных состояния интерфейса

Результат работы подобной модификации одностраничного веб-приложения показан на Рисунке 3.4.3.

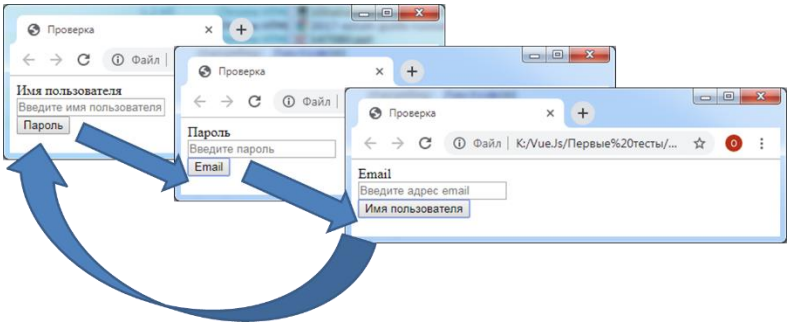


Рисунок 3.4.3 – Смена трёх возможных состояний интерфейсных элементов по нажатии на кнопку

Как правило, работать со статичными элементами управления муторно и крайне неудобно, потому в дальнейшем рационально использовать возможности фреймворка для придания реактивности элементам управления. Это достигается благодаря применению директивы *v-model*, которая в сочетании с директивой *v-bind* позволяет средствами фреймворка действовать в обход прямого взаимодействия с *DOM*.

3.5 Директива *v-model* для реактивной связки состояния переменных и элементов управления в обход *DOM*

Рассматриваемая в параграфе директива *v-model* связывает переменную, объявленную в разделе данных, с интерфейсным элементом управления.

Как правило, работа директивы рассматривается на примере увязки с текстовым полем для ввода информации:

<input v-model="message">

Для последующего развития взята известная заготовка, рассмотренная в качестве примера последовательного вычислительного процесса «*Hello, Vue!*». В неё перед строкой форматирования абзаца (* … *) добавляется конструкция (Рисунок 3.5.1):

<input v-model="message">.

Вместе с ней записывается тег отступа по строке <*br*> для визуального разделения элементов на странице.

В результате одностраничное приложение начинает содержать функцию вводимого с клавиатуры сообщения *message*, которое реактивно изменяется, повторяя пользовательский ввод (Рисунок 3.5.2).

```
7   <DIV ID="app">
8       <input v-model="message">
9       <br>
10      <p>{{ message }}</p>
11  </DIV>
12
13  <SCRIPT>
14
15      var app = new Vue(
16      {
17          el: '#app',
18          data:
19          {
20              message: 'Hello, Vue!'
21          }
22      })
23  </SCRIPT>
```

Рисунок 3.5.1 – Связывание содержимого переменной *message* со значением поля для ввода текста *input*

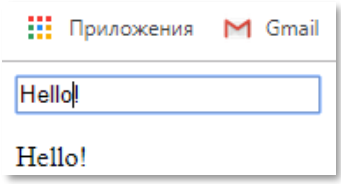


Рисунок 3.5.2 – Дублирование пользовательского ввода текста через *input* в абзац текста, размещённый ниже

Благодаря элементу *input* в сочетании с *v-model* управлять можно не только текстом. Многогранные возможности этого функционала могут быть качественно усвоены и применены только после рассмотрения всех возможных типов поля для ввода *input* в следующем параграфе.

3.6 Вариации интерфейсного элемента управления (*input*)

Поле для ввода *input* может быть визуально изменено и адаптировано под определённые типы данных благодаря использованию различных значений атрибута *type* (тип):

<input type="button | checkbox | file | hidden | image | password | radio | reset | submit | text">

Вертикальной чертой в записи отмечено логическое сложение, то есть в качестве типа может выступать: либо **«кнопка»**, либо **«флажок»**, либо **«форма открытия файла»**, либо **«скрытый режим»**, либо **«изображение»**, либо **«пароль»**, либо **«опция»**, либо **«сброс»**, либо **«подтверждение ввода с отправкой запроса»**, либо **«текст»**.

Без явного указания типа поле для ввода (*input*) является текстовым (Рисунки 3.6.1 и 3.6.2): **<input type="text">**.

```
5       <SCRIPT SRC="Vue.js"></SCRIPT>
6
7       <DIV ID="app">
8           <input type="text" v-model="message">
9           <br>
10          <p>{{ message }}</p>
11      </DIV>
12
13      <SCRIPT>
14
15          var app = new Vue(
16          {
17                  el: '#app',
18                  data:
19                  {
20                      message: 'Hello, Vue!'
21                  }
22          })
23      </SCRIPT>
```

Рисунок 3.6.1 – Указание типа *text* для поля *input*

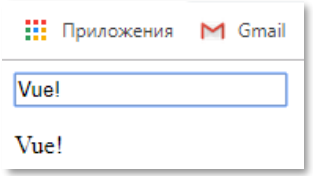


Рисунок 3.6.2 – Результат не отличается от использования *input* без указания конкретного типа

Поле для ввода пароля (***<input type="password">***) не позволяет отслеживать на экране вводимые в него символы. Данный пример демонстрирует верх некомпетентности с точки зрения информационной безопасности, но с образовательной точки зрения является весьма показательным (Рисунок 3.6.3).

Рисунок 3.6.3 – Указание типа *password* для поля *input*

Исходно скрытое поле (***<input type="hidden">***) не предоставляет абсолютно никакой функциональности (Рисунок 3.6.4) и разумно для использования только в сочетании с каким-либо сценарием. Например, ввести имя пользователя во время сеанса посещения одностраничного приложения и не позволять менять это имя пользователя до следующего обращения к одностраничному приложения (принудительного обновления страницы, расположенной по указанному адресу в браузере сочетанием клавиш «*Ctrl*» + «*F5*»).

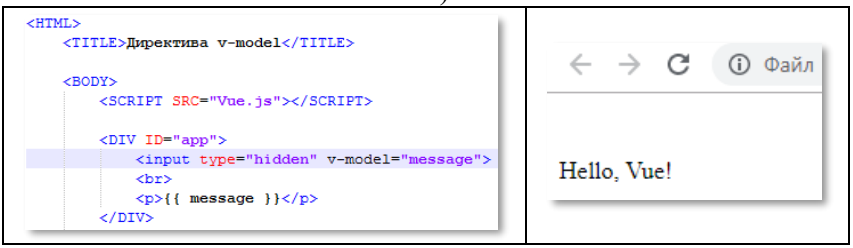


Рисунок 3.6.4 – Указание типа *hidden* для поля *input*

Для создания функционального изображения или иконки (***<input type="image" src="[адрес изображения]">***) уместно использование типа *image* для поля ввода информации (Рисунок 3.6.5).

```
<HTML>
    <TITLE>Директива v-model</TITLE>

    <BODY>
        <SCRIPT SRC="Vue.js"></SCRIPT>

        <DIV ID="app">
            <input type="image" src="vuejs-wide.png">
            <br>
            <p>{{ message }}</p>
        </DIV>
```

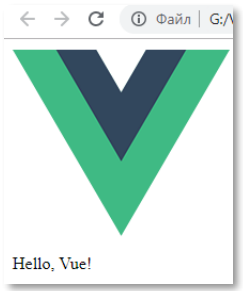


Рисунок 3.6.5 – Указание типа *image* для поля *input*

Далее представлен некоторый вспомогательный справочный материал, позволяющий оценить возможности поля для ввода. На Рисунок 3.6.6 представлены универсальные типы.

Тип	Описание	Вид
button	Кнопка.	Кнопка
checkbox	Флажки. Позволяют выбрать более одного варианта из предложенных.	Пиво Чай Кофе
file	Поле для ввода имени файла, который пересылается на сервер.	Выберите файл Файл не выбран
hidden	Скрытое поле. Оно никак не отображается на веб-странице.	
image	Поле с изображением. При нажатии на рисунок данные формы отправляются на сервер.	Отправить
password	Обычное текстовое поле, но отличается от него тем, что все символы показываются звездочками. Предназначено для того, чтобы никто не подглядел вводимый пароль.	
radio	Переключатели. Используются, когда следует выбрать один вариант из нескольких предложенных.	Пиво Чай Кофе
reset	Кнопка для возвращения данных формы в первоначальное значение.	Сбросить
submit	Кнопка для отправки данных формы на сервер.	Отправить
text	Текстовое поле. Предназначено для ввода символов с помощью клавиатуры.	

Рисунок 3.6.6 – Таблица стандартных типов поля для ввода (*input*)

На Рисунке 3.6.7 представлены типы, которые используются только в современных браузерах, поддерживающих пятую версию гипертекстовой разметки *HTML5.0*.

Тип	Описание
color	Виджет для выбора цвета.
date	Поле для выбора календарной даты.
datetime	Указание даты и времени.
datetime-local	Указание местной даты и времени.
email	Для адресов электронной почты.
number	Ввод чисел.
range	Ползунок для выбора чисел в указанном диапазоне.
search	Поле для поиска.
tel	Для телефонных номеров.
time	Для времени.
url	Для веб-адресов.
month	Выбор месяца.
week	Выбор недели.

Рисунок 3.6.7 – Таблица новых типов поля для ввода (*input*) согласно стандарту *HTML5.0*

Режим кнопки (***<input type="button">***) существенно упрощает решение ранее рассмотренной задачи с изменением надписи на кнопке по нажатии на неё (смена состояния). Кнопочный режим поля для ввода представлен на Рисунке 3.6.8.

```
<HTML>
    <TITLE>Директива v-model</TITLE>

    <BODY>
        <SCRIPT SRC="Vue.js"></SCRIPT>

        <DIV ID="app">
            <input type="button" v-model="message">
            <br>
            <p>{{ message }}</p>
        </DIV>
```

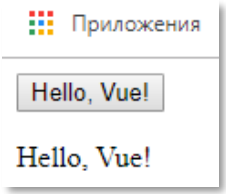
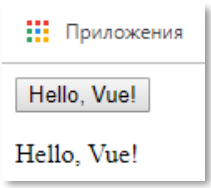


Рисунок 3.6.8 – Указание типа *button* для поля *input*

В данном случае не потребуется обращение к структуре *DOM*. Достаточно изменить только значение переменной *message* (Рисунок 3.6.9).

```
<DIV ID="app">
    <input type="button" @click="Chng" v-model="message">
    <br>
    <p>{{ message }}</p>
</DIV>
```



```
methods:
{
    Chng:function()
    {
        console.log(this.message);
        if (this.message == 'Hello, Vue!')
            this.message = 'Bye, Vue!';
        else
            this.message = 'Hello, Vue!';
    }
}
```

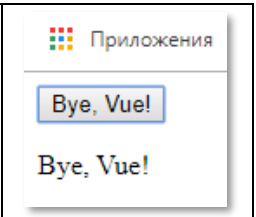


Рисунок 3.6.9 – Модификация примера, рассмотренного в параграфе 3.3

Управление логическим признаком (<***input type="checkbox"***>) тоже бывает чрезвычайно полезным шагом на пути к решению аналогичной задачи смены состояний по флажку, выставляемому на графическом пользовательском интерфейсе (Рисунок 3.6.10).

```
<DIV ID="app">
    <input type="checkbox" v-model="flgChng">
    <br>
    <p>{{ message }}</p>
</DIV>
```

```
data:
{
    message: 'Hello, Vue!',
    flgChng: false
},
updated:function()
{
    if (this.flgChng)
        this.message = 'Bye, Vue!';
    else
        this.message = 'Hello, Vue!';
}
```

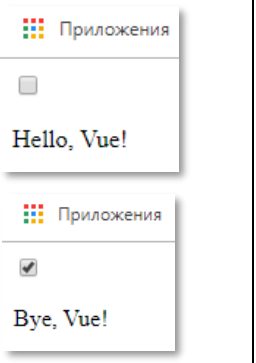


Рисунок 3.6.10 – Указание типа *checkbox* для поля *input*

Опции (<***input type="reset"***>) в сочетании с кнопкой сброса (<***input type="radio"***>) так же являются чрезвычайно полезными интерфейсными элементами управления (Рисунок 3.6.11).

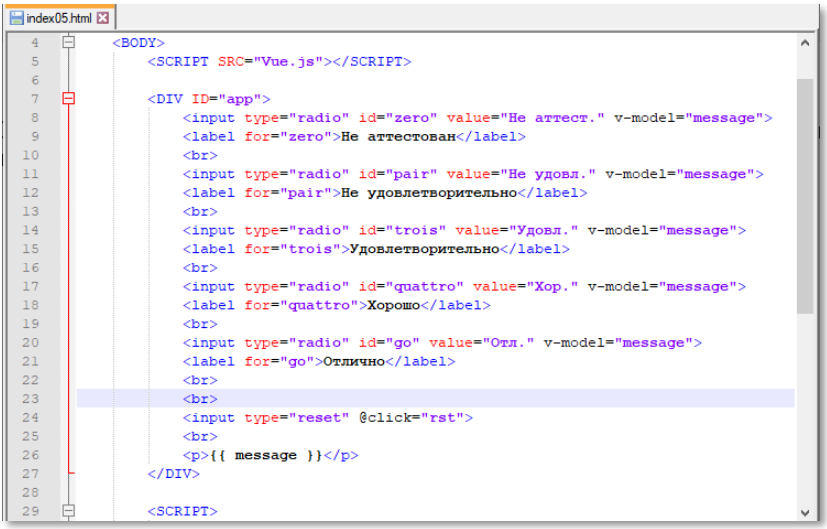


Рисунок 3.6.11 – Указание типов *radio* и *reset* для поля *input* (начало)

Рисунок 3.6.11 – Указание типов *radio* и *reset* для поля *input* (продолжение)

При изменении позиции точки среди возможных опций значение (*value*) выбранной опции передаётся через директиву

v-model в переменную *message*, выводимую под группой опций в абзаце текста (Рисунок 3.6.12).

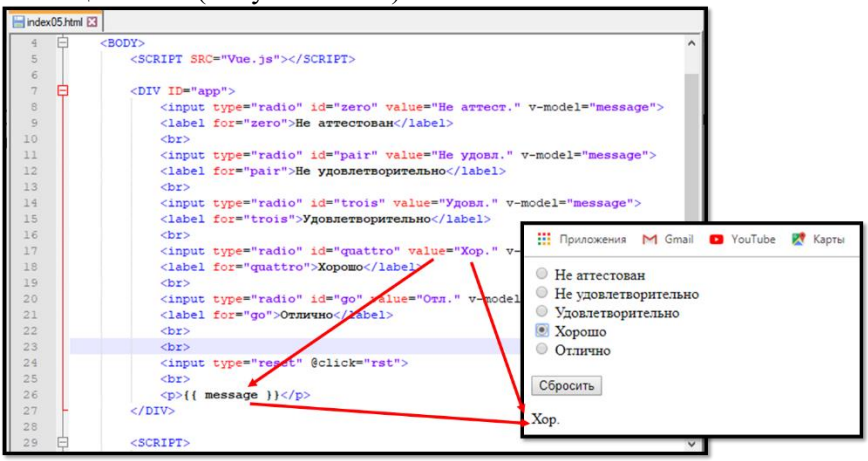


Рисунок 3.6.12 – Значение, закрепляемое за опцией, может отличаться от текста, стоящего напротив этой опции

На Рисунке 3.6.13 показано поведение одностраничного приложения после нажатия на кнопку «Сбросить». Управление передаётся методу *rst*, в котором сбрасывается к пустой строке («») «*message*», фреймворк не может обнаружить ни одного совпадения с пустой строкой среди значений (*values*) опций и потому убирает точку с интерфейса (Рисунок 3.6.14).

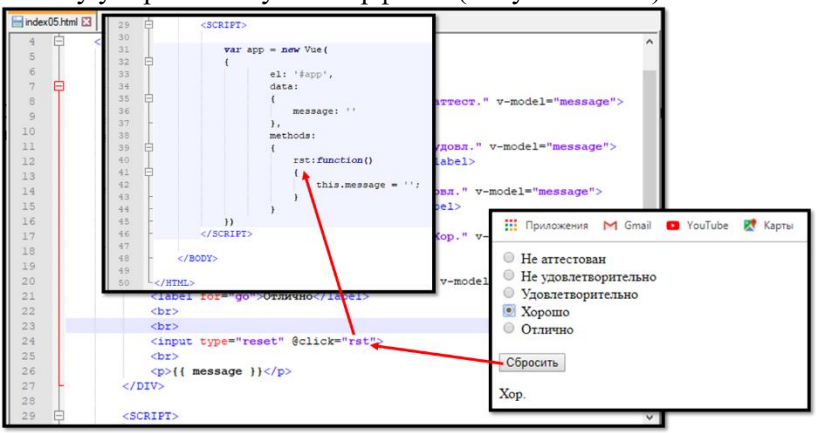


Рисунок 3.6.13 – Демонстрация работы кнопки «Сбросить»

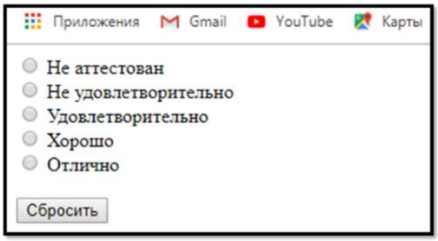


Рисунок 3.6.14 – Состояние интерфейса после нажатия на кнопку «Сбросить»

В этом моменте рассмотрение разветвляющегося вычислительного процесса можно считать завершённым, потому следующим на очереди идёт циклический вычислительный процесс.

4 Циклический вычислительный процесс на базе фреймворка *Vue.js* (директива *v-for*)

На следующем шаге изучения фреймворка «*Vue.js*» рассматривается автоматическое составление статических списков, поскольку списки предполагают использование циклических конструкций.

Помимо циклических конструкций предстоит изучить следующие элементы программирования: массивы; записи; формат текстовой базы данных «*JSON*»; теги, отвечающие за списочные структуры.

Если посмотреть на то, как в *Microsoft Office Word* выглядят списки, то там они бывают нумерованными и маркированными.

Нумерованные списки используются в тех случаях, когда важно отразить приоритет упоминаемых элементов, их последовательность или количество; маркированные списки – это все прочие наборы данных.

В качестве примера рассматривается маркированный список, который создаётся поэлементно без использования циклических конструкций.

Раздел данных содержит первое сообщение (*message1*), предполагаемое к размещению на первой позиции списка. В

исходном примере с «*Hello, Vue!*» на место тегов абзаца/параграфа (*p*) записываются теги маркированного списка (*ul*), а также первое сообщение заключается в теги элемента списка (*li*). Это показано на Рисунке 4.1.

```
<DIV ID="app">
        <UL>
                <LI>{{ message1 }}</LI>
        </UL>
</DIV>


<!-- Сценарий функционирования фреймворка -->
<SCRIPT>

        var app = new Vue(
        {
                el: '#app',
                data:
                {
                        message1: 'Hello, Vue!'
                }
        })
</SCRIPT>
```

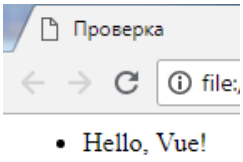


Рисунок 4.1 – Пример кода для составления маркированного списка, состоящего из одного элемента

Структура списка и раздел данных дополняются вторым сообщением (Рисунок 4.2).

```
<DIV ID="app">
        <UL>
                <LI>{{ message1 }}</LI>
                <LI>{{ message2 }}</LI>
        </UL>
</DIV>


<!-- Сценарий функционирования фреймворка -->
<SCRIPT>

        var app = new Vue(
        {
                el: '#app',
                data:
                {
                        message1: 'Hello, Vue!',
                        message2: 'How are You?'
                }
        })
</SCRIPT>
```

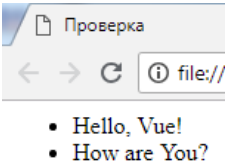


Рисунок 4.2 – Дополнение вторым сообщением экземпляра *Vue.js* и списка

Третье сообщение добавляется по аналогии (Рисунки 4.3).

```
<DIV ID="app">
        <UL>
                <LI>{{ message1 }}</LI>
                <LI>{{ message2 }}</LI>
                <LI>{{ msg3 }}</LI>
        </UL>
</DIV>

<!-- Сценарий функционирования фреймворка -->
<SCRIPT>

        var app = new Vue(
        {
                el: '#app',
                data:
                {
                        message1: 'Hello, Vue!',
                        message2: 'How are You?',
                        msg3: 'Fine. Thank You.'
                }
        })
</SCRIPT>
```

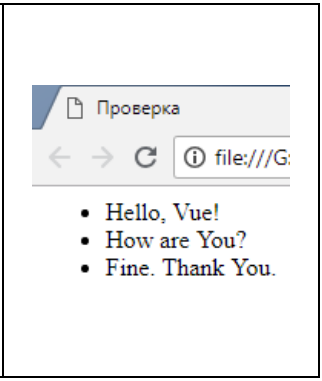


Рисунок 4.3 – Дополнение третьим сообщением экземпляра *Vue.js* и списка

В качестве развития существующий список дополняется нумерованным (Рисунок 4.4). Используются те же три сообщения. Конструкцию, заключённую в теги *ul* необходимо скопировать и заменить в ней теги *ul* на теги нумерованного (*ol*).

```
<DIV ID="app">
        <UL>
                <LI>{{ message1 }}</LI>
                <LI>{{ message2 }}</LI>
                <LI>{{ msg3 }}</LI>
        </UL>

        <OL>
                <LI>{{ message1 }}</LI>
                <LI>{{ message2 }}</LI>
                <LI>{{ msg3 }}</LI>
        </OL>
</DIV>

<!-- Сценарий функционирования фреймворка -->
<SCRIPT>

        var app = new Vue(
        {
                el: '#app',
                data:
                {
                        message1: 'Hello, Vue!',
                        message2: 'How are You?',
                        msg3: 'Fine. Thank You.'
                }
        })
</SCRIPT>
```

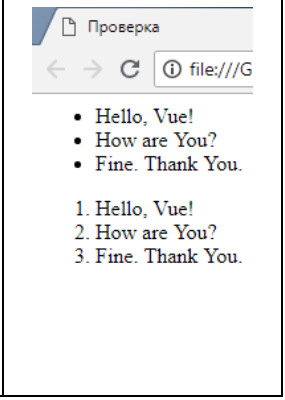


Рисунок 4.4 – Код для составления трёхэлементных маркированного и нумерованного списков

Для того, чтобы отображать данные циклически, необходимо хранить их в определённом формате. Таким форматом для сценариев, написанных на *JavaScript*, как правило, является

«*JSON*». Из Википедии: «*JSON* (*JavaScript Object Notation*) – текстовый формат обмена данными, основанный на *JavaScript*. Как и многие другие текстовые форматы, *JSON* легко читается людьми» [11].

Рассматривается следующий пример, записанный в формате *JSON* (6):

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address":
   {
      "streetAddress": "Московское ш., 101, кв.101",
      "city": "Ленинград",
      "postalCode": "101101"
   },
   "phoneNumbers":
   [
       "812 123-1234",
      "916 123-4567"
   ]
}
```

(6)

На первой вертикали кода выставлены «{» – открывающая фигурная скобка и «}» – закрывающая фигурная скобка. Это границы объекта. Откуда известно, что в данном примере рассматривается только один объект. Опытный разработчик заметит, что на самом деле в примере присутствуют два объекта, но относящихся к разным классам.

Далее последует напоминание про объекты и классы. Лучше всего начать рассмотрение с самого приближенного к жизни примера и разобрать в качестве объекта – человека. Как известно, отдельно взятый человек является экземпляром класса людей (класса «Люди»), так как может быть определён согласно некоторому общему представлению (шаблону) людского рода. Каждого человека можно определить довольно большим количеством параметров, но никто, естественно, не возьмётся этого делать.

Но при этом каждая конкретная инстанция хранит о людях письменные сведения – рассматривает каждого человека по отдельному, интересующему только её набору параметров. С этой точки зрения определение множества людей, прямо или

косвенно связанных с этой инстанцией, уже становится вполне решаемой задачей.

Например, для врачей интересующая информация о человеке – это его группа крови, резус фактор, рост, вес и так далее. Для работодателей – это занимаемая человеком должность, величина ставки, статус и прочее. Но есть в человеке (со множества рассматриваемых позиций) и что-то общее – это, например, фамилии, имена, отчества, номера паспортов или любых других идентифицирующих документов и, конечно же, порядковые номера, под которыми людей учитывают в упомянутых инстанциях.

Как только про человека появилась текстовая запись – он рассматривается как строка базы данных или же как объект «Человек» (экземпляр класса «Люди»).

То же, например, и с адресом. Дом построили. Ему присвоили адрес. Но это только в контексте города адрес – это номер дома и наименование улицы, а в контексте человека адрес это нечто существенно большее, нежели просто элемент на городской карте с указанием улицы и номера дома (а иногда корпуса или строения, если номер дома оказался одним и тем же, а дальнейшая нумерация домов на улице уже занята). Для человека адрес рассматривается вплоть до ячейки, в которой он проживает: подъезд, этаж, номер квартиры, код домофона и так далее.

Итак, возвращаясь к (6), рассматривается контактная информация человека, Иванова Ивана, проживающего в Ленинграде, и имеющего два телефонных номера: мобильный и домашний.

К слову об указанных телефонных номерах. Они без ошибок заключены квадратные скобки, поскольку это в формате *JSON* – массив. Типизированная структура для хранения данных. Как правило, тип массива указывается. Номера телефонов – информация однородная, потому её рационально упаковать в одну переменную для удобства хранения.

Из Википедии: «Массив (в некоторых языках программирования также таблица, ряд, матрица) – структура

данных в виде набора компонентов (элементов массива), расположенных в памяти непосредственно друг за другом» [12].

Напоминание про массивы. Из математики известно, что точка на Декартовой координатной плоскости описывается координатами: «икс» и «игрек» («абсциссой» и «ординатой») – *A(x; y)*. Так вот *A* – это имя точки, или же имя одномерного массива, состоящего из двух элементов *(x; y)*. Точка в трёхмерном пространстве описывается тремя координатами: *(x; y; z)*, то есть массивом из трёх элементов. Множество точек в пространстве – это массив одномерных массивов из трёх элементов или двумерный массив, или таблица, или матрица.

И про второй, вложенный объект в примере (6) следует вспомнить теоретические сведения о записях. Конструкция вида: «Человек.Адрес.Город» – это запись. Запись – это последовательная структура, которая во всём многообразии сведений позволяет докопаться до истины. Так структура записи в общем виде это:

[корень].[вложение 1].[вложение 2]. … .[вложение *N*]

Через запись в (6) известно, что Иванов живёт в Ленинграде. При этом если вдруг «Ленинград», внезапно, переименуют в «Санкт-Петербург», то не потребуется каждому Петербуржцу из перечня людей прописывать другой город, – он изменится для всех автоматически. Но в случае, когда сведения вводятся вручную, автоматически ничего не изменится и менять название города потребуется у каждого. Этим и удобны базы данных и оболочки над ними.

Далее продолжается развитие проекта (Рисунок 4.4). Из него необходимо устранить лишние сообщения *message2* и *msg3*. Оставить только маркированный список с одним единственным элементом *message1*. А саму переменную *message1* повсеместно заменить на «*human*», в которую записать *JSON* из (6). Полученное проиллюстрировано на Рисунке 4.5

```
<DIV ID="app">
        <UL>
                <LI>{{ human }}</LI>
        </UL>
</DIV>

<!-- Сценарий функционирования фреймворка -->
<SCRIPT>

        var app = new Vue(
        {
                el: '#app',
                data:
                {
                        human:
                        {
                                "firstName": "Иван",
                                "lastName": "Иванов",
                                "address":
                                {
                                        "streetAddress": "Московское ш., 101, кв.101",
                                        "city": "Ленинград",
                                        "postalCode": "101101"
                                },
                                "phoneNumbers":
                                [
                                        "812 123-1234",
                                        "916 123-4567"
                                ]
                        }

                }
        })
</SCRIPT>
```

Рисунок 4.5 – Формирование объекта «человек» в структуре исходных данных *Vue.js*

Результатом станет вывод в один единственный элемент списка всего текста, записанного в переменную *human*, единой строкой (Рисунок 4.6).

Рисунок 4.6 – Результат вывода объекта в строку списка

В рассмотренном примере сведения о человеке представлены всё в том же сумбурном виде *JSON*, но конечный пользователь не обязан уметь читать и просто читать *JSON* – ему необходимо поставлять сведения в удобном для чтения виде (как, например, это обусловлено на Рисунке 4.7 и отображено на Рисунке 4.8).

```
<DIV ID="app">
        <UL>
                <LI>Персона: {{ human.lastName }} {{ human.firstName }} </LI>
                <LI>Проживает по адресу: {{ human.address.postalCode }},
                        г. {{ human.address.city }},| {{ human.streetAddress }} </LI>
                <LI>Телефон (моб.): +7 {{ human.phoneNumbers[1] }} </LI>
        </UL>
</DIV>
```

Рисунок 4.7 – Разложение сведений о человеке в несколько строк списка для удобства их прочтения

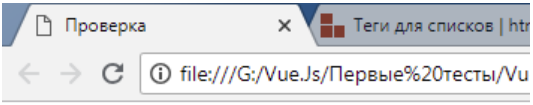


Рисунок 4.8 – Результат отображения сведений о человеке в удобном для чтения виде

По результату видно, что адрес отобразился не полностью. Во втором элементе списка, в третьей подстановке в записи намеренно нарушена иерархическая последовательность. Это и привело к ошибке. На Рисунке 4.9 ошибка исправлена – в подстановку добавлено недостающее звено «*address*». Результат дополнен недостающими сведениями (Рисунок 4.10).

```
<DIV ID="app">
        <UL>
                <LI>Персона: {{ human.lastName }} {{ human.firstName }} </LI>
                <LI>Проживает по адресу: {{ human.address.postalCode }},
                        г. {{ human.address.city }}, {{ human.address.streetAddress }} </LI>
                <LI>Телефон (моб.): +7 {{ human.phoneNumbers[1] }} </LI>
        </UL>
</DIV>
```

Рисунок 4.9 – Исправление ошибки в коде, намеренно допущенной на Рисунке 4.7

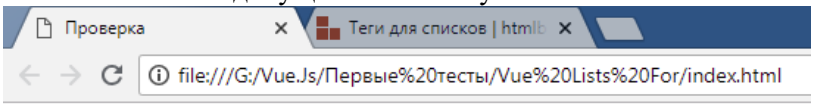


Рисунок 4.10 – Результат отображения сведений о человеке в удобном для чтения виде без ошибок

Далее рассматривается более сложный пример с двумя персонами, записанными в переменную. Имеет место массив людей. С учётом формулировки «*human*» необходимо переименовать в «*people*», добавить открывающую и закрывающую квадратные скобки, обуславливающие массив, скопировать Иванова в буфер обмена, после закрывающей скобки объекта записать запятую и вставить далее содержимое из буфера обмена, переименовав Иванова в Петрова, а также изменив сведения о нём. Не стоит забывать, что в список в

разделе гипертекстовой разметки страницы так же необходимо внести коррективы, связанные изменением структуры данных и некоторых имён в ней, а также сразу предусмотреть вывод информации о Петрове (Рисунок 4.11).

```
<DIV ID="app">
        <UL>
                <LI>Персона: {{ people[0].lastName }} {{ people[0].firstName }} </LI>
                <LI>Проживает по адресу: {{ people[0].address.postalCode }},
                        г. {{ people[0].address.city }}, {{ people[0].address.streetAddress }} </LI>
                <LI>Телефон (моб.): +7 {{ people[0].phoneNumbers[1] }} </LI>

                <LI>Персона: {{ people[1].lastName }} {{ people[1].firstName }} </LI>
                <LI>Проживает по адресу: {{ people[1].address.postalCode }},
                        г. {{ people[1].address.city }}, {{ people[1].address.streetAddress }} </LI>
                <LI>Телефон (моб.): +7 {{ people[1].phoneNumbers[0] }} </LI>
        </UL>
</DIV>

<!-- Сценарий функционирования фреймворка -->
<SCRIPT>

        var app = new Vue(
        {
                el: '#app',
                data:
                {
                        people:
                        [
                                {
                                        "firstName": "Иван",
                                        "lastName": "Иванов",
                                        "address":
                                        {
                                                "streetAddress": "Московское ш., 101, кв.101",
                                                "city": "Ленинград",
                                                "postalCode": "101101"
                                        },
                                        "phoneNumbers":
                                        [
                                                "812 123-1234",
                                                "916 123-4567"
                                        ]
                                },
                                {
                                        "firstName": "Петр",
                                        "lastName": "Петров",
                                        "address":
                                        {
                                                "streetAddress": "Дмитровское ш., 43, к.1, кв.15",
                                                "city": "Москва",
                                                "postalCode": "127434"
                                        },
                                        "phoneNumbers":
                                        [
                                                "985 132-5476"
                                        ]
                                }
                        ]

                }
        })
</SCRIPT>
```

Рисунок 4.11 – Изменение структуры приложения под работу с массивом людей

Результат вывода сведений о двух персонах представлен на Рисунке 4.12

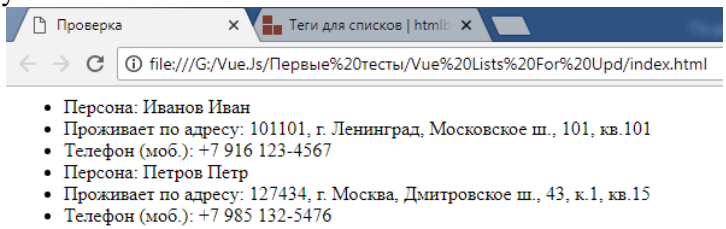


Рисунок 4.12 – Результат отображения сведений о двух персонах в удобном для чтения виде

В представленной структуре заметна определённая однотипность, которую можно поручить циклической конструкции. Аналогичный пример имеется в официальном руководстве *Vue.js*, который представлен на Рисунке 4.13.

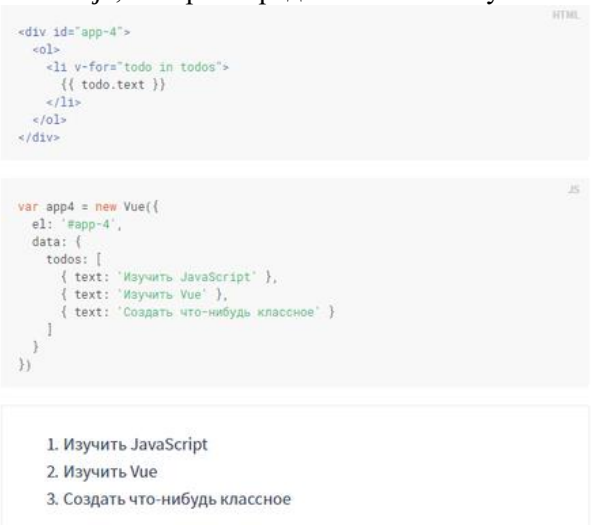


Рисунок 4.13 – Копия фрагмента из официального руководства *Vue.js*. Пример использования директивы *v-for*

Разобравшись с примером, в директиве *v-for* в разрабатываемом одностраничном приложении необходимо записать «*human in people*» вместо «*todo in todos*» (из примера). Циклический оператор в директиве *v-for* будет просматривать по очереди каждый элемент массива «*people*», а оператор *in* будет присваивать в переменную «*human*» рассматриваемый *i*-й элемент массива («*people[i]*»). Работа директивы аналогична работе оператора *forEach* из си-образных языков программирования (Рисунок 4.14). Результат работы представлен на Рисунке 4.15.

```
<DIV ID="app">
        <UL>
                <LI v-for="human in people">Персона: {{ human.lastName }} {{ human.firstName }}<BR>
                        Проживает по адресу: {{ human.address.postalCode }},
                        г. {{ human.address.city }}, {{ human.address.streetAddress }}<BR>
                        Телефоны: {{ human.phoneNumbers }}
                </LI>
        </UL>
</DIV>
```

Рисунок 4.14 – Изменение структуры приложения под работу с массивом людей циклически

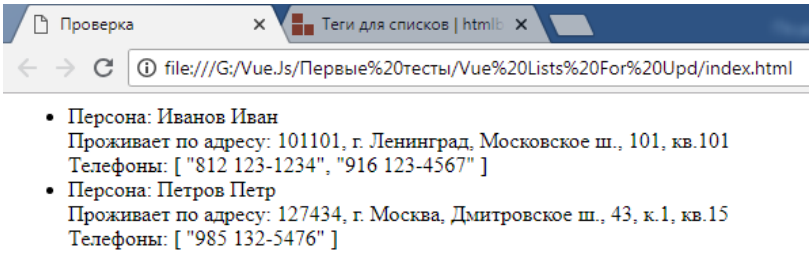


Рисунок 4.15 – Результат составления списка с использованием циклической конструкции

По итогам рассмотрения всех типов вычислительных процессов в рамках изучения возможностей фреймворка *Vue.js* можно заключить, что любой обучающийся, освоивший материалы, представленные выше, в состоянии разработать простейшую информационную систему управления в формате одностраничного веб-приложения.

Однако, для разработки более интерактивного одностраничного веб-приложения потребуется рассмотрение ещё нескольких дополнительных элементов управления и более широкого набора директив фреймворка. Попытка сделать это предпринята в пятом разделе учебно-методических указаний.

5. Формирование функционального списка (директива *v-bind, select*)

Для динамического связывания данных с определенными атрибутами во *Vue.js* используется директива *v-bind*, которая имеет сокращённую (ленивую форму) «:».

К примеру, чтобы связать свойство *src* элемента с переменной *imageSrc* из перечня данных, хранимых на жёстком диске, необходимо записать следующее:

```
<img v-bind:src="imageSrc">
<!-- или сокращенно -->                    (7)
<img :src="imageSrc">
```

HTML-тег <*select*> позволяет создать интерфейсный элемент управления в виде раскрывающегося списка, а также список с одним или множественным выбором (Рисунок 5.1).

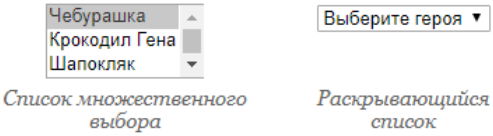


Рисунок 5.1 – Разновидности списка, размещаемого с использованием тега <*select*>

Конечный вид списка зависит от использования атрибута *size* (размер) тега <*select*>, который устанавливает высоту списка. Ширина списка определяется самым широким (длинным) текстом, указанным в теге <*option*>, а также может быть изменена при помощи настройки стиля.

Каждый пункт списка создается при помощи тега <*option*>, который должен быть вложен в контейнер <*select*>. Если в задаче планируется отправка данных из списка на сервер, то потребуется размещение тега <*select*> внутри формы. Это так же необходимо, когда к данным списка идёт обращение через сценарии.

На Рисунке 5.2 представлен пример синтаксиса списка, создаваемого вручную (без помощи средств *JavaScript*).

```
<select>
  <option>Пункт 1</option>
  <option>Пункт 2</option>
</select>
```

Рисунок 5.2 – Синтаксис списка, размещаемого с использованием тега <*select*>

Помимо настраиваемого атрибута *size* для работы со списками полезно знать и помнить и другие атрибуты, представленные на Рисунке 5.3.

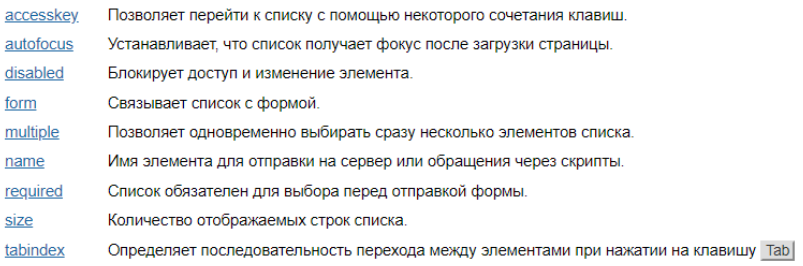


Рисунок 5.3 – Перечень настраиваемых атрибутов списка, размещаемого с использованием тега *<select>*

Для примера создано одностраничное веб-приложение на базе фреймворка *Vue.js*, в котором присутствует функциональный список, состоящий из двух сотрудников некоторой организации.

При выборе сотрудника из списка в качестве параграфа (абзаца) текста выводится занимаемая этим сотрудником должность.

Использован список *<select>* с возможностью множественного выбора *multiple*. Список выбранными элементами динамически/реактивно связан с переменной *selected (v-model="selected")*, объявленной в разделе исходных данных (*data*) фреймворка.

Визуально это удобно, но данный режим накладывает ряд ограничений. Реально несколько элементов в решаемой задаче выделяться не будут, потому сведения всегда должны быть предоставлены только элементом с нулевым индексом в массиве выбранных элементов списка.

При необходимости механизм может быть легко расширен до вывода на страницу списка выбранных элементов.

Обучающиеся могут попробовать проделать это самостоятельно.

В качестве начального приближения выбранных элементов должен быть указан пустой массив (**selected: []**). Это очень важный момент, поскольку компилятор браузера не пропустит ни пустое значение (*null*), ни пустую строку («»), ни пустой объект ({}). Поймёт он только пустой массив (Рисунок 5.4).

Следует отметить, что список в режиме множественного выбора размещён в одностраничном приложении только из соображений, что он выглядит как статический, а не как выпадающий список.

Получено работоспособное одностраничное приложение, отвечающее нуждам поставленной задачи: от выбранного из списка сотрудника может быть получена информация о занимаемой им должности.

Отчётливо видно, что по завершении работы со списком состояние ранее выбранного элемента фиксируется.

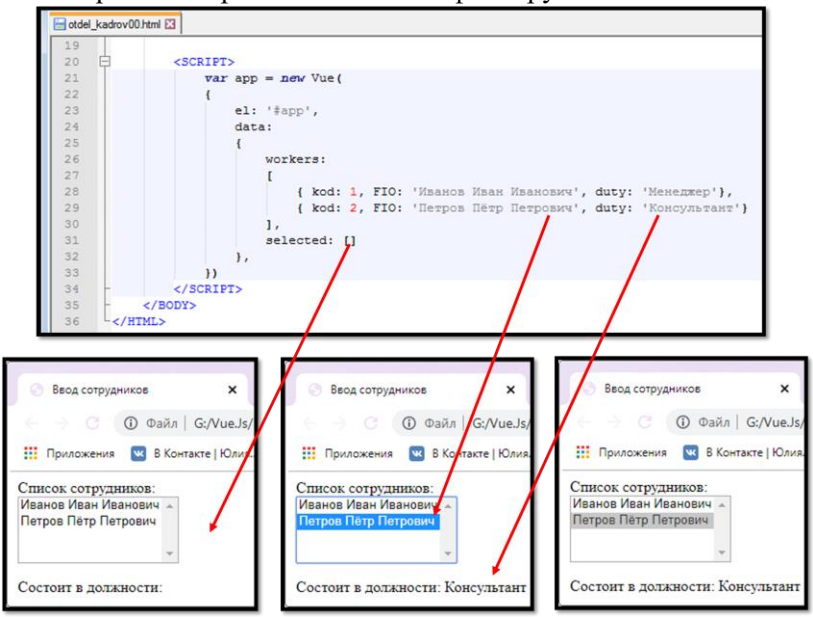


Рисунок 5.4 – Работа с элементами списка по умолчанию: при потере фокуса выбор элемента сохраняется

Это не очень красиво с точки зрения пользовательского интерфейса, потому далее приложение развито и дополнено сбросом состояния по завершении работы со списком.

В гипертекстовой части функциональный список дополнен реакцией на событие потери фокуса (*@blur*).

В этот момент необходимо произвести очистку массива выбранных элементов. Для этих целей в сценарной части

организован раздел пользовательских методов (*methods*), в котором записан новый метод «*rst*», необходимый для сброса. В его теле на место ранее выбранного элемента записывается пустой массив (***this.selected = [];***). Это показано на Рисунках 5.5 и 5.6.

```
<DIV ID="app">
    Список сотрудников:
    <br>
    <select @blur="rst" v-model="selected" multiple>
        <option v-for="w in workers" :value="w">
            {{ w.FIO }}
        </option>
    </select>
```

Рисунок 5.5 – Связь списка *select* с обработчиком события *blur*, вызывающего метод *rst* при потере фокуса интерфейсным элементом управления

```
methods:
{
    rst:function()
    {
        this.selected = [];
    },
}
```

Рисунок 5.6 – Метод *rst*, сбрасывающий ранее выполненный выбор элементов в функциональном списке *select*

Элементы списка <*option*> набираются циклически посредством применения директивы *v-for*. На каждом витке цикла рассматривается конкретная строчка-объект массива сотрудников *workers*, записываемая в переменную с именем *w*.

Важно отметить, что функциональный список является не просто элементом индикации информации с возможностью маркирования входящих в его состав строк, он по своей сути является контейнером, способным хранить сами объекты. И это его контейнерное свойство крайне важно для нужд управления данными.

В качестве закрепляемого в элементе списка значения (*value*) поступает сам объект «сотрудник» (*w*). Динамическая/реактивная связь значений реализуется посредством применения директивы *v-bind*.

Между открывающим и закрывающим тегами *<option></option>* в «усатых» скобках размещается информация, которая выводится в строках списка. В данном случае – это Ф.И.О. сотрудников.

С целью повышения устойчивости одностраничного веб-приложения организована проверка на непустой массив выбранных из списка элементов. Так в «усатые» скобки абзаца заключён тернарный условный оператор – сокращённая форма записи условного оператора: ***{{ selected.length > 0 ? selected[0].duty : '' }}***.

Слева от знака вопроса «?» указывается логическое выражение (в данном случае проверка на непустой массив – количество элементов такого массива должно быть строго больше нуля), справа значение или действие, которое должно быть выполнено в случае выполнения логического выражения, а через двоеточие – значение или действие, которое должно выполняться в случае невыполнения логического выражения.

Так для случая выполнения логического выражения от элемента массива (с нулевым индексом) среди всех выбранных элементов списка получается и выводится в абзац значение должности сотрудника, в ином случае, когда массив пуст, – пустая строка текста. Вышесказанное отмечено на Рисунке 5.7.

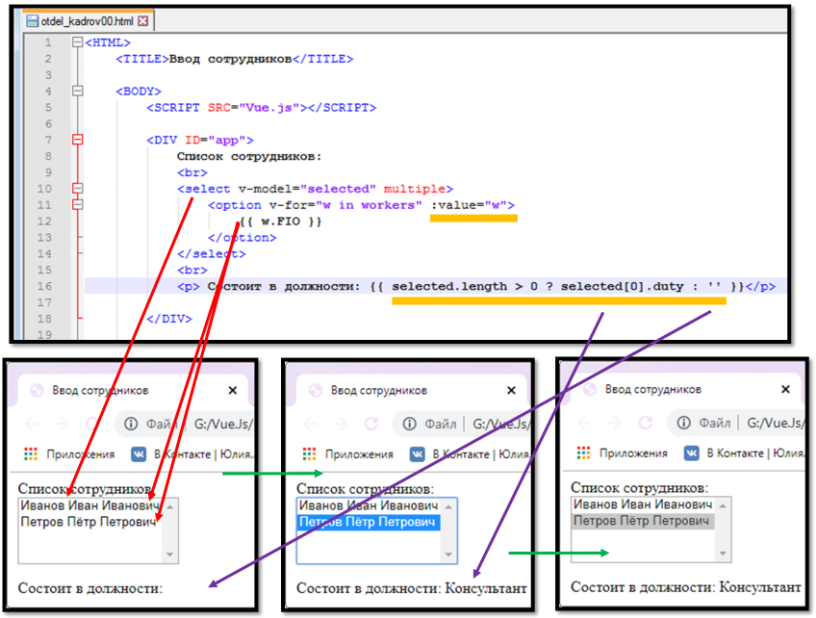


Рисунок 5.7 – Связь кодовых конструкций с состоянием интерфейсных элементов управления

Как известно, на предприятиях обычно имеется ограниченный набор должностей. С точки зрения составления реляционных баз данных – необходимо выносить все эти должности в отдельную таблицу и удерживать в базе в качестве связанных объектов с основными интересующими объектами.

Дальнейшее развитие одностраничного приложения состоит в следующем: в разделе исходных данных фреймворка в массиве (*duties*) записан перечень должностей и, одновременно, в массиве сотрудников все строковые значения должностей изменены на соответствующие им объекты класса «должность» (Рисунок 5.8).

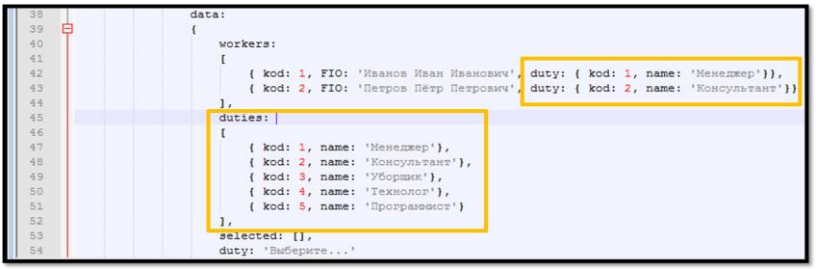


Рисунок 5.8 – Подготовка исходных данных к работе с выпадающим перечнем должностей

В гипертекстовой части при этом дела обстоят сложнее. В связи с тем, что для начального или сброшенного состояния какие-либо значения должностей отсутствуют, то невозможно абсолютно для всех случаев связать высвечиваемую в списке должность «нулевого» сотрудника из массива выбранных сотрудников, поскольку с самого начала не существует этого «нулевого» сотрудника (никто не выбран из списка).

В строках 18–29 прописано условное отображение (*v-if, v-else*). Для случая, когда выбранных из списка сотрудников нет, выбранная из выпадающего списка должность записывается в отдельную глобальную переменную *duty*, что обеспечивает задел для последующего добавления в список нового сотрудника с этой должностью (Рисунок 5.9).

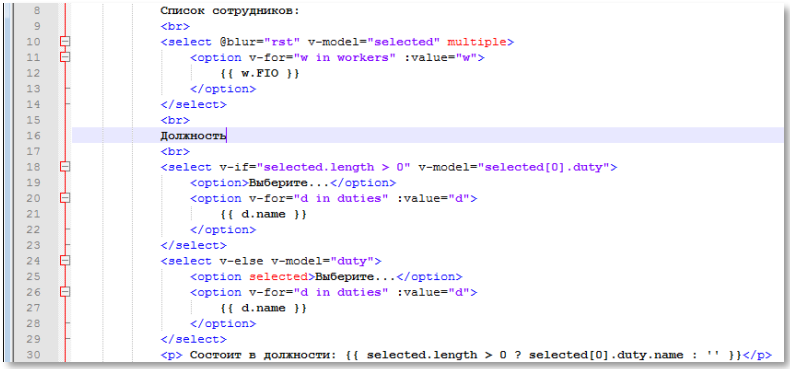


Рисунок 5.9 – Условная отрисовка выпадающего списка должностей в зависимости от складывающейся ситуации с выбранными элементами в основном списке сотрудников

Всё отмеченное ранее отработано на графическом пользовательском интерфейсе (Рисунок 5.10). В начальный момент никто не выбран и список «Должность» находится в состоянии «Выберите…». При этом от не пуст. Как только из списка сотрудников выбирается кто-либо, значение выбранного индекса в списке должностей перемещается на позицию, соответствующую занимаемой этим сотрудником должности.

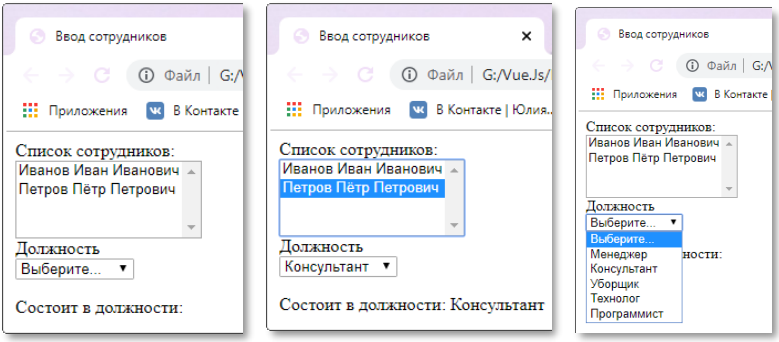


Рисунок 5.10 – Проверка работоспособности взаимодействия основного списка и выпадающего списка

В ходе повествования был отмечен задел для добавления нового сотрудника. Потому далее ставится задача расширения перечня сотрудников программными средствами.

Должности уже динамически связаны с фреймворком *Vue.js*, осталось настроить динамическую связь «Ф.И.О.» сотрудников. Для этого создаётся переменная *FIO* в разделе исходных данных (Рисунок 5.11).

```
45                  duties:
46                  [
47                      { kod: 1, name: 'Менеджер'},
48                      { kod: 2, name: 'Консультант'},
49                      { kod: 3, name: 'Уборщик'},
50                      { kod: 4, name: 'Технолог'},
51                      { kod: 5, name: 'Программист'}
52                  ],
53                  selected: [],
54                  duty: 'Выберите...',
55                  FIO: ''
```

Рисунок 5.11 – Переменная для хранения Ф.И.О. выбранного или создаваемого нового сотрудника

Вместе с тем создаётся аналогичный блок условного форматирования в гипертекстовой части (строки 19, 20) одностраничного веб-приложения (Рисунок 5.12).

```
8           Список сотрудников:
9           <br>
10          <select @blur="rst" v-model="selected" multiple>
11              <option v-for="w in workers" :value="w">
12                  {{ w.FIO }}
13              </option>
14          </select>
15          <br>
16          <br>
17          Ф.И.О.
18          <br>
19          <input v-if="selected.length > 0" v-model="selected[0].FIO">
20          <input v-else v-model="FIO">
21          <br>
22          Должность
23          <br>
```

Рисунок 5.12 – Условная отрисовка текстового поля для ввода/отображения Ф.И.О. сотрудника

В результате проделанных операций в коде, графический пользовательский интерфейс приобрёл возможность отображения в дополнительном текстовом поле значения «Ф.И.О.» выбранного из списка сотрудника (Рисунок 5.13).

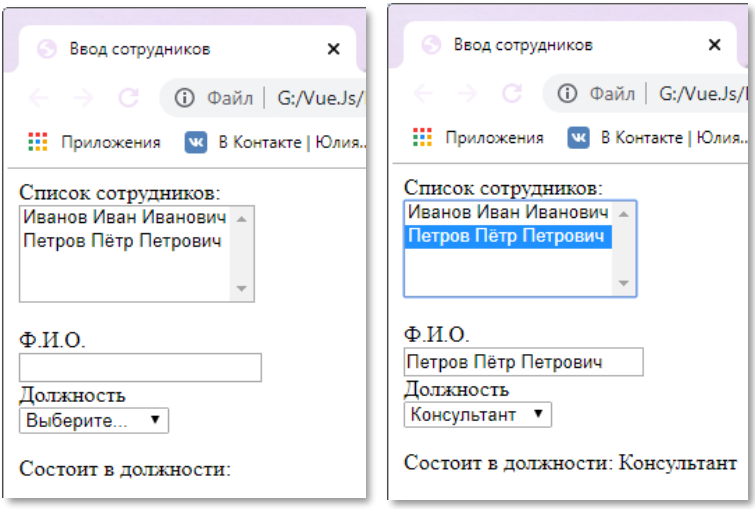


Рисунок 5.13 – Дублирование в поле Ф.И.О. соответствующего параметра от выбранного из списка сотрудника

Для полноценного решения поставленной задачи осталось добавить кнопку, в результате нажатия на которую, введённые сведения о новом сотруднике записывались бы в список. Так в сценарной части предусмотрен метод (*addWorker*, строки 80-90), исполняемый по нажатии на кнопку «Ввести» (Рисунок 5.14).

```
66          methods:
67          {
68              rst:function()
69              {
70                  this.selected = [];
71              },
72              getMaxKod:function(inp)
73              {
74                  var max = 0;
75                  for (var i = 0; i < inp.count; i++)
76                      if (inp[i] > max)
77                          max = inp[i];
78                  return max;
79              },
80              addWorker:function()
81              {
82                  var wrkr = new Object();
83                  wrkr.kod = this.getMaxKod(this.workers) + 1;
84                  wrkr.FIO = this.FIO;
85                  wrkr.duty = this.duty;
86                  this.workers.push(wrkr);
87                  this.FIO = '';
88                  this.duty = 'Выберите...';
89                  this.rst();
90              }
91          }
```

Рисунок 5.14 – Метод добавления в перечень сотрудников нового сотрудника

В методе создаётся новый объект, в который на место кода записывается найденный максимальный код среди существующих (выполняется в пользовательском методе *getMaxKod()*, которому в качестве входного параметра передаётся перечень сотрудников), увеличенный на единицу. В поле «Ф.И.О.» переписывается значение из глобальной переменной «Ф.И.О.», в которой хранится введённое с клавиатуры текстовое значение. В поле «Должность» записывается объект из глобальной переменной «Должность», соответствующий ранее выбранному объекту-должности из выпадающего списка.

После выполнения всех упомянутых операций созданный объект заносится в массив и все введённые значения удаляются

из полей – графический пользовательский интерфейс сбрасывается к умолчаниям, в списке на последней позиции появляется новый сотрудник.

Для свойства доступности кнопки вводится следующее правило: если из списка не выбран ни один сотрудник – кнопка доступна для нажатия (то есть можно ввести нового), если хотя бы один сотрудник выбран, то кнопка недоступна (редактирование существующих сотрудников запрещено). Строка, содержащая соответствующее, правило отмечена на Рисунке 5.15.

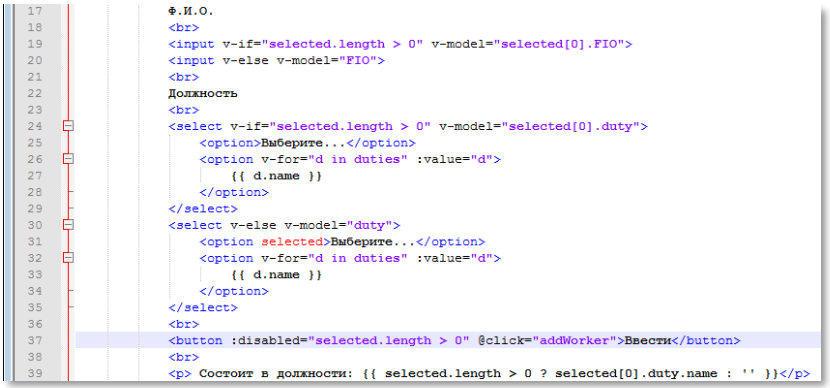


Рисунок 5.15 – Правило смены режимов ввода/просмотра сотрудника

Далее в рамках графического пользовательского интерфейса отработаны все те состояния и ограничения, которые были отмечены ранее по тексту.

На Рисунке 5.16 (начало) представлены следующие состояния:

- инициализации (загрузка одностраничного приложения);
- выбора сотрудника из списка (кнопка «Ввести» блокируется);
- ввода параметров нового сотрудника.

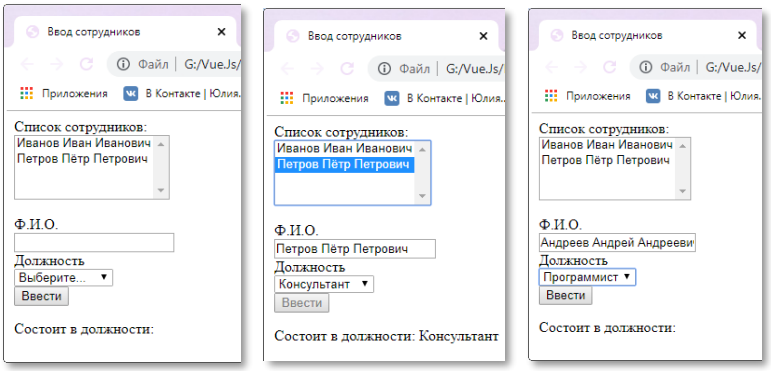


Рисунок 5.16 – Работа с созданным одностраничным приложением, содержащим функциональный список (начало)

На Рисунке 5.16 (продолжение) представлены следующие состояния:

- ввода параметров нового сотрудника (изображение продублировано);
- нажатия на кнопку «Ввести» (новый сотрудник добавлен в список, состояние элементов приложения сброшено к умолчаниям);
- просмотра сведений о новом сотруднике, занесённом в список.

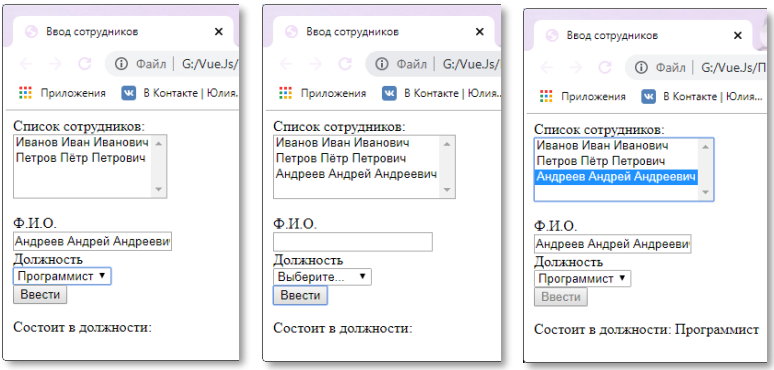


Рисунок 5.16 – Работа с созданным одностраничным приложением, содержащим функциональный список (продолжение)

Приложение 1. Список тем курсового проекта по дисциплине «Информационное обеспечение систем управления»

Вариант №1. Разработка текстовой базы данных и веб-оболочки на тему «Магазин по продаже оргтехники».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников магазина оргтехники. Магазин реализует типовой товар. Постоянным клиентам и покупателям, набравшим товара на сумму *X* рублей, предоставляется скидка.

Готовые запросы:

- показывать ассортимент выбранного товара и его цену;
- находить товар по названию (по изготовителю);
- набирать комплектующие на определённую сумму;
- рассчитывать стоимость покупки, учитывая скидку;
- показывать количество (стоимость) проданного товара (по выбранному товару, по магазину в целом) за отчётный период, например, в виде диаграммы.

Вариант №2. Разработка текстовой базы данных и веб-оболочки на тему: «Архив дел уголовного розыска».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников архива уголовного розыска. В архиве лежат Дела за разные годы.

Готовые запросы:

- выдавать список Дел по фамилии, имени, отчеству (допускается объединение этих атрибутов в атрибут ФИО) осуждённого;
- выдавать список Дел, которые вёл указанный следователь;
- находить Дела по их содержанию (совпадение по словам или их сочетаниям);
- выдавать список Дел по указанной статье преступления (по характеру преступления).

Вариант №3. Разработка текстовой базы данных и веб-оболочки на тему: «Кафедра Университета».

Описание предметной области. База данных создаётся для информационного обслуживания руководящего состава кафедры. База данных должна содержать информацию о преподавателях и инженерах, работающих на кафедре, а также об их занятости (по научно-исследовательским работам, преподаваемым дисциплинам, организационной работе с обучающимися, методической работе).

Готовые запросы**:**

- выдавать сводную информацию обо всех работниках кафедры;
- выдавать информацию о занятости преподавателя научно-исследовательской работой;
- выдавать информацию о занятости преподавателя организационной работой с обучающимися;
- выдавать информацию о занятости преподавателя методической работой;
- выдавать информацию о преподавателе, ведущем указанный вид занятий по указанной дисциплине;
- выдавать информацию о видах занятий, которые проводятся по выбранной дисциплине.

Вариант №4. Разработка текстовой базы данных и веб-оболочки на тему: «Больница».

Описание предметной области. База данных создаётся для информационного обслуживания медицинских работников, пациентов и посетителей больницы (навещающих). База данных должна содержать информацию о медицинских работниках больницы, лечащихся у них пациентах и размещении пациентов по палатам.

Готовые запросы:

- выдавать сведения о медицинских работниках;
- выдавать сведения о медицинских работниках, наблюдающих за определённым пациентом;
- выдавать сведения о пациентах;
- выдавать сведения о пациентах, наблюдаемых у определенного медицинского работника;
- выдавать размещение пациентов по палатам.

Вариант №5. Разработка текстовой базы данных и веб-оболочки на тему «Автосервис».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников автосервиса. Автосервис предлагает различные услуги по ремонту, покраске, помывке, тюнингу и прочим работам по восстановлению и модификации автомобилей. В автосервисе работают сотрудники, специализирующиеся на определённых видах работ. Каждый специалист может принять к работе и обслужить несколько машин. Каждая машина, находящаяся в сервисе, может содержать несколько проблем/неполадок. Один и тот же клиент может владеть несколькими автомобилями. Для постоянных клиентов предусмотрена система скидок.

Готовые запросы:

- выдавать список услуг, предлагаемых автосервисом с указанием цен на соответствующие услуги;
- выдавать список машин, находящихся в автосервисе;
- выдавать информацию об указанной машине (выводить перечень оказываемых услуг);
- выдавать информацию о работе, проделанной указанным мастером за отчётный период времени (день, месяц, квартал, год);
- рассчитывать общую стоимость услуг для клиентов.

Вариант №6. Разработка текстовой базы данных и веб-оболочки на тему «Библиотека».

Описание предметной области. База данных создаётся для обслуживания частной библиотеки. Библиотека содержит книги разных авторов, изданий, тематик. База данных предназначена для поиска выбранной книги на полке.

Готовые запросы:

- выдавать список книг по названию;
- выдавать список трудов указанного автора (учитывать труды, выполненные в соавторстве);
- выдавать список книг по указанной тематике;
- выдавать список книг указанного издательства;
- выдавать местонахождение указанной книги.

Вариант №7. Разработка текстовой базы данных и веб-оболочки на тему: «Детский сад».

Описание предметной области. База данных создается для информационного обслуживания администрации детского сада. Детский сад посещают дети от 2-х до 7-ми лет. Дети, которым менее 3-х лет, в сентябре посещают в ясли. В каждой группе работает пара воспитателей (один – утром, другой – вечером), а также нянечка. Детей в каждой группе не может быть больше 15-ти человек. С детьми проводятся занятия музыкой, физкультурой и рисованием в отдельных комнатах по два раза в неделю. С детьми обязательно гуляют два раза в день.

Готовые запросы:

- показывать список детей указанной группы;
- показывать список детей указанного возраста;
- показывать занятость указанного воспитателя;
- показывать занятость указанной группы в указанный день недели;
- показывать процентное отношение мальчиков и девочек в указанной группе (например, в виде диаграммы);
- находить название (и/или №) группы по ФИО ребёнка.

Вариант №8. Разработка текстовой базы данных и веб-оболочки на тему: «Гостиница».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников гостиницы и клиентов. В гостинице имеются номера, которые различаются по количеству мест, количеству комнат, удобствам и т.д. От всех этих параметров и их сочетания зависит цена гостиничного номера. Посетители могут занимать номера, бронировать их заранее. Постоянным посетителям и детям до 12 лет с родителями предоставляется скидка.

Готовые запросы:

- предоставлять (и бронировать) посетителям номер в гостинице в зависимости от их потребностей (количества мест для размещения группы гостей и т.д.);
- показывать количество свободных (занятых) мест по категории номеров указанного типа (например, в виде диаграммы);
- показывать список постоянных посетителей и предоставляемую им скидку;
- показывать информацию по указанному номеру (количество мест, комнат, наличие холодильника, телевизора, бара и т.д.).

Вариант №9. Разработка текстовой базы данных и веб-оболочки на тему: «Аптека».

Описание предметной области. База данных создаётся для информационного обслуживания посетителей аптеки. В аптеку города поступает ассортимент лекарств со склада каждые 7 дней. Аптека предлагает услуги по продаже лекарств и их бронированию. Срок бронирования лекарств – 3 дня. В справочной системе аптеки можно получить информацию о лекарствах, находящихся в самой аптеке: название, форма выпуска, срок годности, аннотация, цена, изготовитель.

Готовые запросы

- выдавать сведения о лекарствах;
- предоставлять покупателям возможность бронирования лекарств, сроком на 3 дня;
- выдавать информацию о поступлении лекарства в аптеку, исходя из ассортимента на складе;
- выдавать информацию о продажах указанного лекарства за неделю (месяц, год);
- выполнять поиск лекарства по названию, форме выпуска, изготовителю;
- выдавать список лекарств, применяемых для указанной болезни (легких недугах).

Вариант №10. Разработка текстовой базы данных и веб-оболочки на тему: «ВУЗ».

Описание предметной области. База данных создаётся для информационного обслуживания работников деканата. В ВУЗе учатся студенты, разделённые на группы. По каждой специальности обучается несколько групп. Каждая кафедра ведёт занятия по нескольким дисциплинам. Занятия для студентов проводят преподаватели с разных кафедр. Лабораторные работы ведут два преподавателя.

Готовые запросы**:**

- выдавать информацию о студенте по № зачетной книжки, по ФИО;
- выдавать список дисциплин, читаемых указанной кафедрой;
- выдавать список преподавателей, проводящих занятия в указанной группе;
- выдавать список групп, обучающихся на данной специальности.

Вариант №11. Разработка текстовой базы данных и веб-оболочки на тему: «Продажа билетов в кино».

Описание предметной области. База данных создается для информационного обслуживания сотрудников кинотеатра (кассиров). В кинотеатре имеется несколько залов, в которых одновременно могут идти разные фильмы. Каждый фильм могут показывать несколько раз в день по сеансам (в установленные часы). На фильм может налагаться ограничение по возрасту.

Готовые запросы:

- показывать сеансы на указанное время, если имеются ограничения по возрасту, то сообщать об этом;
- показывать список фильмов по залам, идущих сегодня в кинотеатре;
- выдавать информацию о цене билета на указанный сеанс и указанный фильм;
- выдавать информацию о кратком содержании, задействованных актёрах и режиссёрах указанного фильма;
- продавать билет на выбранный сеанс указанного фильма;
- отражать продажи по сеансам (например, в виде диаграммы).

Вариант №12. Разработка текстовой базы данных и веб-оболочки на тему: «Хлебопекарня».

Описание предметной области. База данных создается для информационного обслуживания сотрудников хлебопекарни. Хлебопекарня выпекает изделия на заказ. Продукты заказывают у поставщиков, выпечку отправляют в магазины города.

Готовые запросы:

- показывать ассортимент товара и его цену;
- показывать сумму (количество) заказываемого продукта у указанного поставщика за отчётный период;
- показывать список товара (общую сумму заказа, суму заказа указанного товара), заказанного указанным магазином;
- показывать количество произведенного товара (за отчётный период, по названию), например, в виде диаграммы.

Вариант №13. Разработка текстовой базы данных и веб-оболочки на тему: «Складской учёт».

Описание предметной области. База данных создаётся для информационного обслуживания склада. Некоторая фирма имеет склад товаров. Эти товары фирма получает от производителя и расфасовывает их для магазинов. Также данная фирма имеет возможность оптового отпуска ассортимента товаров. Для постоянных клиентов предусмотрена система скидок.

Готовые запросы**:**

- выдавать ассортимент товара, находящегося на складе в настоящий момент;
- выдавать ассортимент товара, заказанного указанным магазином;
- показывать список продаж за указанный период времени;
- показывать список клиентов, имеющих скидку.

Вариант №14. Разработка текстовой базы данных и веб-оболочки на тему: «Издательство».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников издательства. Издательство печатает книги, журналы, методические пособия и т.д. Каждый автор может написать работу в соавторстве.

Готовые запросы:

- выдавать список трудов указанного автора (учитывать труды, выполненные в соавторстве);
- выдавать список трудов по выбранному разделу (книги, журналы, методические пособия и т.д.);
- выдавать информацию об указанном авторе;
- показывать количество напечатанных работ за отчётный период (квартал, год).

Вариант №15. Разработка текстовой базы данных и веб-оболочки на тему: «Строительная компания».

Описание предметной области. База данных создаётся для информационного обслуживания клиентов компании. Компания строит дачные домики, бани и т.д. Имеются готовые проекты с

фиксированной ценой, проекты на заказ стоят дороже на *x*%. На объект выезжает бригада, состоящая из рабочих и прораба.

Готовые запросы:

- показывать список предоставляемых услуг и цену на них;
- показывать количество свободных (занятых) бригад (например, в виде диаграммы);
- показывать работу, проводимую указанной бригадой;
- показывать список работ, проводимых у указанного заказчика;
- рассчитывать стоимость выполненных услуг для указанного заказчика.

Вариант №16. Разработка текстовой базы данных и веб-оболочки на тему: «Контора адвоката».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников конторы. Контора оказывает юридические услуги (ведение дела в суде, консультация) по разным делам (гражданские, уголовные). Каждый адвокат специализируется в одной области (жилищные, семейные, убийства и т.д.). На каждое обращение в контору заводится Дело (№, содержание, адвокат, клиент).

Готовые запросы:

- показывать список предоставляемых услуг и их цену;
- выдавать список клиентов, обращавшихся за указанной услугой;
- выдавать список свободных адвокатов по выбранной услуге;
- выдавать содержание Дела по его номеру.

Вариант №17. Разработка текстовой базы данных и веб-оболочки на тему: «Спортивная команда по баскетболу».

Описание предметной области. База данных создаётся для информационного обслуживания администратора команды. Женский баскетбольный клуб, играющий в суперлиге в дивизионе А, состоит из 4 тренеров (главный и второй тренеры для основного и дублирующего состава), 24 игроков (12 – основой состав и 12 – дубль), врача, массажиста и администратора команды. На каждого члена команды имеется досье (ФИО, возраст, рост, вес, звание (м/с, заслуженный тренер СССР или России и т.д.), амплуа). Результаты каждого матча фиксируются в таблицу индивидуально (фолы, подборы (свой/чужой щит), перехваты, потери, очки). Команда играет в чемпионате России и Евролиге.

Готовые запросы:

- выводить счёт указанного матча для основного состава (дубля);
- показывать список команд чемпионата России (Евролиги);
- выдавать сведения об указанном игроке;

- выдавать список игроков для двух составов для указанного амплуа (центровой, разыгрывающий, защитник);
- выдавать список игроков выбранной команды с тренерами.
- показывать результат указанного матча по указанному игроку (команде в целом);
- показывать самого результативного игрока в указанном матче.

Вариант №18. Разработка текстовой базы данных и веб-оболочки на тему: «Поликлиника».

Описание предметной области. База данных создаётся для информационного обслуживания регистрационного отдела поликлиники. База данных должна содержать информацию о врачах, ведущих приём, расписании приёма, и пациентах, проживающих на участке, закрепленном за определённой поликлиникой.

Готовые запросы**:**

- выдавать сводную информацию обо всех врачах поликлиники;
- выдавать сводную информацию о пациентах;
- выдавать информацию о записи пациента к врачу;
- выдавать информацию о приёме врачей на указанную дату;
- выдавать информацию о пациентах, имеющих льготы на приобретение лекарств.

Вариант №19. Разработка текстовой базы данных и веб-оболочки на тему: «Туристическое бюро».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников фирмы. Фирма оказывает услуги в туристическом бизнесе. Клиентам предлагаются путёвки в разные страны, города. Путёвки отличаются содержанием программы (отдых, экскурсии, туризм и т.д.), имеется возможность выбора путёвки по цене (в зависимости от места проживания, всё включено и т.д.). Для постоянных клиентов и детей до 12 лет имеются скидки.

Готовые запросы:

- выдавать список стран;
- выдавать список городов;
- покупать путёвку в выбранное место с расчётом её стоимости;
- показывать весь ассортимент путёвок в указанное место;
- выбирать путёвку по содержанию, по цене и т.д.;
- показывать список самых популярных путёвок (по месту пребывания, по содержанию, в целом).

Вариант №20. Разработка текстовой базы данных и веб-оболочки на тему: «Фильмотека».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников и посетителей магазина. Магазин продает различные виды фильмов (на различных носителях).

Готовые запросы:

- найти фильм по названию (по жанру, исполнителям, режиссёру);
- показывать список возможных носителей для выбранного фильма;
- показывать список фильмов по жанрам;
- показывать информацию по выбранному фильму (жанр, исполнители, режиссёр, краткое содержание);
- показывать список наиболее продаваемых фильмов;
- показывать количество проданных фильмов за отчётный период (сумму продаж).

Вариант №21. Разработка текстовой базы данных и веб-оболочки на тему: «Магазин обуви».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников магазина. В магазине имеется несколько отделов обуви (мужская, женская, детская, спортивная и т.д). Обувь имеет размер, цвет, фасон, цену, разделяется на зимнюю, летнюю, весна-осень и т.д., нумеруется артикулом. Для постоянных клиентов предусмотрена система скидок в виде карты.

Готовые запросы:

- выдавать информацию о количество пар по заданному артикулу, размеру, цвету и т.д.
- выдавать ассортимент обуви по отделам, сезону;
- выдавать информацию об указанной обуви (цена, страна изготовитель);
- продавать выбранный товар;
- показывать продажи за выбранный период времени (например, в виде диаграммы).

Вариант №22. Разработка текстовой базы данных и веб-оболочки на тему: «Транспортная/логистическая компания».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников компании. Компания предоставляет услуги по доставке грузов воздушным, речным, морским путём, по железной дороге или автомобилем по различным маршрутам.

Готовые запросы:

- выдавать список маршрутов, обслуживаемых компанией и цену на них;
- выдавать список транспорта, занятого на указанном маршруте;

- выдавать список вариантов проезда по указанному маршруту (морской и т.д);
- рассчитывать стоимость услуг, оказанных по перевозке указанного груза. Постоянным клиентам предоставляется скидка.
- показывать грузооборот по указанному маршруту.

Вариант №23. Разработка текстовой базы данных и веб-оболочки на тему: «Магазин детских товаров».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников магазина. Магазин продаёт детские товары разного вида: одежда, обувь, питание, игрушки, письменные принадлежности, предметы по уходу за телом и т.д. По каждому виду имеется богатый ассортимент товаров. Постоянным клиентам и покупателям, чья сумма покупки превысила *x* руб., предоставляется система скидок.

Готовые запросы:

- находить товар по его названию, показывать отдел, где его можно приобрести;
- показывать ассортимент выбранного отдела;
- показывать товарооборот по выбранным отделам (сумму продаж), например, в виде диаграммы;
- рассчитывать стоимость покупки, учитывая скидки;
- показывать наличие выбранного размера (цвета, фасона) для обуви и одежды.

Вариант №24. Разработка текстовой базы данных и веб-оболочки на тему: «Компания по услугам связи».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников и абонентов компании. Компания предоставляет услуги связи. Каждый абонент обслуживается по выбранному тарифу. Каждый тариф имеет свой набор услуг и стоимость за единицу услуги.

Готовые запросы:

- показывать список услуг;
- показывать список тарифов и цену за единицу услуги;
- показывать список абонентов по указанному тарифу;
- показывать список самых распространённых тарифов;
- показывать процентное распределение тарифов среди абонентов (например, в виде диаграммы).

Вариант №25. Разработка текстовой базы данных и веб-оболочки на тему: «Отдел кадров».

Описание предметной области. База данных создаётся для информационного обслуживания работников отдела кадров. В фирме

работают сотрудники на разных должностях и разных ставках. Сотрудники могут иметь детей, быть пенсионерами или иметь инвалидность. Сотрудникам предоставляет очередной отпуск. Некоторые сотрудники могут быть в отпуске по уходу за ребёнком.

Готовые запросы**:**

- показывать список сотрудников фирмы по ФИО, по должности, ставке, имеющих детей, пенсионеров, находящихся в очередном отпуске, находящихся в отпуске по уходу за ребёнком;
- выдавать информацию о должностях сотрудников;
- выдавать информацию о предыдущих местах работы сотрудников;
- находить сотрудников по ФИО.

Вариант №26. Разработка текстовой базы данных и веб-оболочки на тему: «Школа».

Описание предметной области. База данных создаётся для информационного обслуживания администрации школы. В школе учатся дети, распределённые по классам. У каждого класса свой классный руководитель. Каждый учитель имеет своё направление, вести он может 6 уроков день максимум, по выбранному направлению – один учитель.

Готовые запросы:

- показывать список учащихся указанного класса;
- показывать классного руководителя указанного класса;
- показывать занятость указанного учителя;
- показывать успеваемость указанного ученика;
- показывать список учеников, учащихся без троек (например, в процентном отношении в виде диаграммы).

Вариант №27. Разработка текстовой базы данных и веб-оболочки на тему: «Кондитерская фабрика».

Описание предметной области. База данных создаётся для информационного обслуживания администрации фабрики. Фабрика изготавливает кондитерские изделия. Для изготовления товара требуются продукты, которые фабрика заказывает у поставщиков. Готовые товары расфасовываются для магазинов.

Готовые запросы:

- показывать список магазинов, заказывающих указанный товар;
- показывать список продуктов, заказываемых у указанного поставщика;
- показывать ассортимент указанного товара и цену;
- выбирать наиболее популярный вид указанного товара (например, в виде диаграммы);

- показывать стоимость произведённого товара за отчётный период времени.

Вариант №28. Разработка текстовой базы данных и веб-оболочки на тему: «Компания по продаже недвижимости».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников и клиентов фирмы. Компания покупает и продаёт дома и квартиры. С каждой сделки компания имеет *x*%. Клиенты также покапают и продают свои дома и квартиры.

Готовые запросы:

- показывать все предложения на указанный вид недвижимости;
- показывать спрос на указанный вид недвижимости;
- показывать сведения о клиенте по его № паспорта (ФИО);
- находить спрос (предложение) по выбранной цене, расположении, площади;
- показывать прибыль компании (без учёта налогов и т.д.) за отчётный период;
- показывать список самых популярных спросов (предложений).

Вариант №29. Разработка текстовой базы данных и веб-оболочки на тему: «Овощной магазин».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников магазина. Магазин получает товар со склада и расфасовывает его для продажи. Некоторые овощи могут быть испорчены.

Готовые запросы:

- показывать ассортимент товара на выбранный день;
- показывать приход-расход выбранного товара;
- показывать прибыль магазина (без учёта налогов и т.д) по указанному товару (за отчётный период времени);
- показывать в процентном отношении количество хорошего и испорченного товара (например, в виде диаграммы) для выбранного товара;
- показывать количество проданных единиц указанного товара за отчётный период.

Вариант №30. Разработка текстовой базы данных и веб-оболочки на тему: «Мебельный магазин».

Описание предметной области. База данных создаётся для информационного обслуживания сотрудников магазина. Магазин продаёт товары, изготовленные на фабриках, берёт заказы на изготовление мебели по индивидуальному проекту (с увеличением стоимости на *x*%).

Готовые запросы:

- показывать ассортимент предлагаемой мебели и цену товара;
- показывать количество проданной мебели выбранного образца за отчётный период времени;
- показывать список сделанных заказов за отчётный период времени;
- рассчитывать стоимость заказа клиента по индивидуальному проекту.

Вариант №31. Разработка текстовой базы данных и веб-оболочки на тему: «Магазин музыкальных произведений».

Описание предметной области. База данных создаётся для информационного обслуживания клиентов магазина. Покупатели самостоятельно выбирают и находят понравившийся им товар.

Готовые запросы:

- показать все произведения указанного композитора (исполнителя), имеющиеся в магазине;
- показать местоположение выбранного произведения;
- показать список носителей для выбранного произведения;
- показать список произведений по жанру исполнения;
- находить произведение по названию, году выпуска, альбому и т.д.
- показывать список произведений указанного композитора (исполнителя) по выбранным годам творчества.

Приложение 2. Схема курсового проекта по дисциплине «Информационное обеспечение систем управления»

Пояснительная записка к курсовому проекту должна содержать следующие основные разделы:

1. Правильно оформленный титульный лист, содержащий следующие надписи:
   - полное наименование университета на момент сдачи пояснительной записки: Федеральное государственное автономное образовательное учреждение высшего образования «Российский университет транспорта» (РУТ (МИИТ),
   - наименование института на момент сдачи пояснительной записки: Институт транспортной техники и систем управления (ИТТСУ),

• наименование выпускающей кафедры на момент сдачи пояснительной записки: Управление и защита информации (УиЗИ),
• надпись «Курсовой проект»,
• по дисциплине: «Информационное обеспечение систем управления»,
• на тему: «Проектирование реляционной базы данных [аптеки/отдела кадров/магазина/и т.д]»,
• «Выполнил: », «Вариант: », «Проверил: »,
• «город – год» (г. Москва – 2019).

2. Формулировку цели на курсовой проект.
3. Постановку задачи курсового проектирования.
   3.1. Описание предметной области.
   3.2. Перечень готовых запросов.
4. Обоснование выбора БД, СУБД и среды программирования.
5. Инфологическое проектирование базы данных.
   5.1. Выделение базовых сущностей предметной области. Анализ предметной области.
   5.2. Анализ информационных задач и круга пользователей системы (Реферативная часть).
6. Логическое проектирование базы данных. Моделирование БД.
   6.1. Составление *ER*-диаграммы.
   6.2. Преобразование *ER*-диаграммы в структуру базы данных (блок-схема отношений).
   6.3. Описание процесса нормализации модели базы данных.
   6.4. Определение дополнительных ограничений на атрибуты.
7. Физическое проектирование ИОСУ.
   7.1. Разработка графического пользовательского интерфейса оболочки системы.
   7.2. Структурирование таблиц базы данных, подготовка оболочки к сопряжению с базой данных.
   7.3. Перечень дополнительных возможностей системы и их описание.
   7.4. Семантическая блок-схема программы.
   7.5. Технологическая блок-схема процесса разработки программного обеспечения.
   7.6. Программная реализация готовых запросов. Демонстрация работы.
   7.7. Листинг программного обеспечения.
8. Вывод о проделанной работе.
9. Библиографический список (список литературы).

Список литературы:

1. Васильева М.А. Методические указания к лабораторной работе «Создание таблиц баз данных» по дисциплине «Информационное обеспечение систем управления». – М.: МИИТ, 2007. – 23 с.
2. Васильева М.А. Методические указания к лабораторной работе «Навигационный способ доступа к базе данных» по дисциплине «Информационное обеспечение систем управления». – М.: МИИТ, 2007. – 14 с.
3. Википедия. Свободная энциклопедия [Электронный ресурс] : Фреймворк. *URL*: *https://ru.wikipedia.org/wiki/Фреймворк* (дата обращения: 27.07.2018).
4. Википедия. Свободная энциклопедия [Электронный ресурс] : *JavaScript. URL*: *https://ru.wikipedia.org/wiki/JavaScript* (дата обращения: 06.01.2019).
5. Флэнаган Д. *JavaScript*. Карманный справочник. Сделайте веб-страницы интерактивными! / Перевод А.Г. Сысонюк. – Москва.: Издательский дом "Вильямс", 2015. – С. 320. – 1000 экз. – ISBN 978-5-8459-1948-9 (рус.).
6. Википедия. Свободная энциклопедия [Электронный ресурс] : Мультипарадигменное программирование. *URL*: *https://ru.wikipedia.org/wiki/Мультипарадигменное_программирование* (дата обращения: 06.01.2019).
7. Википедия. Свободная энциклопедия [Электронный ресурс] : Парадигма программирования. *URL*: *https://ru.wikipedia.org/wiki/Парадигма_программирования* (дата обращения: 06.01.2019).
8. *Vue.js* [Электронный ресурс] : Введение – *Vue.js*. *URL*: *https://ru.vuejs.org/v2/guide/* (дата обращения: 27.07.2018).
9. *Vue.js* [Электронный ресурс] : Редактор *Markdown* – *Vue.js*. *URL*: *https://ru.vuejs.org/v2/examples/* (дата обращения: 27.07.2018).
10. Википедия. Свободная энциклопедия [Электронный ресурс] : *Document Object Model*. *URL*: *https://ru.wikipedia.org/wiki/Document_Object_Model* (дата обращения: 08.12.2019).
11. Википедия. Свободная энциклопедия [Электронный ресурс] : *JSON*. *URL*: *https://ru.wikipedia.org/wiki/JSON* (дата обращения: 08.12.2019).
12. Википедия. Свободная энциклопедия [Электронный ресурс] : Массив (тип данных). *URL*: *https://ru.wikipedia.org/wiki/Массив_(тип_данных)* (дата обращения: 08.12.2019).

Оглавление

УЧЕБНО-МЕТОДИЧЕСКОЕ ИЗДАНИЕ

Сафронов Антон Игоревич
Котова Анастасия Ильинична

Проектирование типовой информационной системы управления с использованием технологии *web*-программирования на базе фреймворка *Vue.js*

Учебно-методическое пособие
для проведения аудиторных занятий по дисциплине
«Информационное обеспечение систем управления»