Методичні вказівки

до виконання лабораторних робіт з курсу «Web-технології»
для студентів спеціальностей
124 – «Системний аналіз»,
126 – «Інформаційні системи та технології»,
151 – «Автоматизація та комп'ютерно-інтегровані технології»,
152 – «Метрологія та інформаційно-вимірювальна техніка»

Міністерство освіти і науки України
Вінницький національний технічний університет

Методичні вказівки

до виконання лабораторних робіт з курсу «Web-технології»
для студентів спеціальностей
124 – «Системний аналіз»,
126 – «Інформаційні системи та технології»,
151 – «Автоматизація та комп'ютерно-інтегровані технології»,
152 – «Метрологія та інформаційно-вимірювальна техніка»

Вінниця
ВНТУ
2019

Рекомендовано до друку Методичною радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 8 від 17.04.2019 р.)

Рецензенти:
Кучерук В. Ю. д.т.н., проф.
Севастьянов В. М. к.т.н., доц.

Методичні вказівки до виконання лабораторних робіт з курсу «Web-технології» для студентів спеціальностей 124 – «Системний аналіз», 126 – «Інформаційні системи та технології», 151 – «Автоматизація та комп'ютерно-інтегровані технології», 152 – «Метрологія та інформаційно-вимірювальна техніка» / Уклад. М. В. Барабан, С. В. Барабан, О. М. Бевз. – Вінниця : ВНТУ, 2019. – 67 с.

У даних методичних вказівках розглянуті теоретичні відомості та практичні рекомендації для створення сучасних веб-сторінок, сайтів і веб-додатків, завдання для лабораторних робіт та вказана рекомендована література. Методичні вказівки розроблені відповідно до плану кафедри та програми дисципліни «Web-технології».

ЗМІСТ

Лабораторна робота № 1

Тема: Вивчення основ Bootstrap 4 при створенні головної сторінки сайту.

Мета: створення сайту з використанням технології Bootstrap 4.

Теоретичні відомості

Bootstrap – це безкоштовний набір інструментів з відкритим кодом, призначений для створення веб-сайтів та веб-додатків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних веб-сайтів і веб-додатків.

Bootstrap – це клієнтський фреймворк, тобто інтерфейс для користувача, на відміну від коду серверної сторони, який знаходиться на сервері. Репозиторій з цим фреймворком є одним із найпопулярніших на GitHub. Серед інших його використовують NASA і MSNBC [1].

Хід роботи

I. Завантаження і встановлення Bootstrap 4

Існує три способи встановлення Bootstrap 4:

1. Використання npm. Встановлення Bootstrap 4 відбувається за допомогою команди *npm install bootstrap*;

2. Використання мережі доставки контенту (CDN). Потрібно написати цей код у вільному просторі проекта:

<link rel=«stylesheet» href=«https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css» integrity=«sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm» crossorigin=«anonymous»>;

3. Завантажити Bootstrap 4 бібліотеку і використовувати її локально.

Структура проекту має виглядати таким чином:

Рисунок 1.1 – Структура проекту

II. Система Bootstrap Grid

Система Bootstrap Grid допоможе створити макет і легко побудувати адаптивний веб-сайт. Треба відмітити, що змінна *-xs-* в Bootstrap 4 більше не існує. Система сітки Bootstrap 4 дозволяє розміщувати до 12 стовпців на

сторінці, тому макет буде ґрунтуватися на цьому. Для того, щоб використовувати Bootstrap, треба додати *row* клас у *div*.

col-lg-2 // class used for large devices like laptops
col-md-2 // class used for medium devices like tablets
col-sm-2// class used for small devices like mobile phones

III. Navbar

Рисунок 1.2 – Navbar

Navbar в Bootstrap 4 має більш сучасний вигляд. Він допоможе побудувати стовпці, щоб отримати їх, необхідно додати Navbar клас до **index.html** файлу:

```
<nav class="navbar navbar-expand-lg fixed-top ">
  <a class="navbar-brand" href="#">Home</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
  </button>

  <div  class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav mr-4">
          <li class="nav-item">
              <a class="nav-link" href="#">About</a>
          </li>
          <li class="nav-item">
              <a class="nav-link " href="#">Portfolio</a>
          </li>
          <li class="nav-item">
              <a class="nav-link " href="#">Team</a>
          </li>
          <li class="nav-item">
              <a class="nav-link " href="#">Post</a>
          </li>
          <li class="nav-item">
              <a class="nav-link " href="#">Contact</a>
          </li>
      </ul>
  </div>
</nav>
```

Також необхідно створити нову папку в директорії, де знаходиться **index.html**, назвати її **css** і в ній створити **main.css** файл, після чого буде існувати можливість налаштувати стиль CSS. Наступний рядок потрібно помістити в head-тег **index.html** файлу:

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

Ще одне завдання – це додавання деяких кольорів для навігаційної панелі в **main.css** файлі:

```
.navbar{
 background:#F97300;
}
.nav-link , .navbar-brand{
 color: #f4f4f4;
 cursor: pointer;
}
.nav-link{
 margin-right: 1em !important;
}
.nav-link:hover{
 background: #f4f4f4;
 color: #f97300;
}
.navbar-collapse{
 justify-content: flex-end;
}
.navbar-toggler{
  background:#fff !important;
}
```

Новий BootstrapGrid будується за допомогою системи Flexbox, тому для вирівнювання потрібно використовувати властивості цієї системи.

Наприклад, щоб помістити меню NavBar справа, потрібно додати властивість *justify-content* і встановити її в значенні flex-end:

```
.navbar-collapse{
 justify-content: flex-end;
}
```

Для того, щоб надати навігаційній панелі фіксовану позицію, потрібно додати navbar-fixed-top всередину тегу nav.

Для того, щоб змінити колір фону Navbar на світлий, необхідно додати в **main.css** файл .bg-light, Для темного фону додайте .bg-dark, а для світло-блакитного фону потрібно додати .bg-primary. Ось як це має виглядати:

```
.bg-dark{
background-color:#343a40!important
}
.bg-primary{
background-color:#343a40!important
}
```

IV. Заголовок

```
<header class="header">
</header>
```

Тепер варто створити макет для заголовка. Необхідно переконатися, що заголовок займає висоту вікна, тому буде використовуватись код JQuery.

Спочатку потрібно створити нову папку в директорії, де знаходиться **index.html**, назвати її **js** і в ній створити файл з ім'ям **main.js**, після чого внести його в файл index.html перед закриттям body тегу:

```
<script type="text/javascript" src='js/main.js'></script>
```

В main.js файл вставте цей код JQuery:

```
$(document).ready(function(){
$('.header').height($(window).height());
})
```

Наступним кроком буде встановлення фонового зображення в заголовок у **main.css**:

```
/*header style*/
.header{
 background-image: url('../images/headerback.jpg');
 background-attachment: fixed;
 background-size: cover;
 background-position: center;
}
```

Далі потрібно додати накладку, щоб заголовок виглядав більш професійно: додати це до **index.html** файлу:

```
<header class="header">
<div class="overlay"></div>
</header>
```

Потім необхідно додати код в **main.css** файл:

```
.overlay{
 position: absolute;
 min-height: 100%;
 min-width: 100%;
 left: 0;
 top: 0;
 background: rgba(244, 244, 244, 0.79);
}
```

Тепер потрібно доповнити опис всередині заголовка, для цього необхідно створити div і дати йому клас .container, тобто клас Bootstrap, який допоможе додати зміст і зробити макет більш оптимізованим:

```
<header class="header">
<div class="overlay"></div>
<div class="container"></div>
</header>
```

Потім добавити ще один div, який буде містити опис.

```
<div class="description text-center">
        <h3> Hello ,Welcome To My officail Website
        <p>
        cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
        proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

```
            </p>
            <button class="btn btn-outline-secondary">See more</button>
            </h3>
        </div>
```

Після чого додати йому класи .descriptioni та .text-center, щоб переконатися, що зміст поміщається в центрі сторінки.

Додамо клас btn btn-outline-secondary до елемента button. У Bootstrap існує багато інших класів для кнопок, можна перевірити деякі приклади:

CodePen: кнопки bootstrap 4, Кнопка головна, Кнопка за замовчуванням, Кнопки небезпеки, Інформації, Кнопка попередження, Кнопка успішного натискання, Темна кнопка та ін.

Ось як цей стиль виглядає для .descriptiony у **main.css** файлі:

```
    .description{
   position: absolute;
   top: 30%;
   margin: auto;
   padding: 2em;
    }
.description h1{
 color:#F97300 ;
}
.description p{
 color:#666;
 font-size: 20px;
 width: 50%;
 line-height: 1.5;
}
.description button{
 border:1px  solid #F97300;
 background:#F97300;
 color:#fff;
}
```

Після всіх цих кроків заголовок має виглядати так:

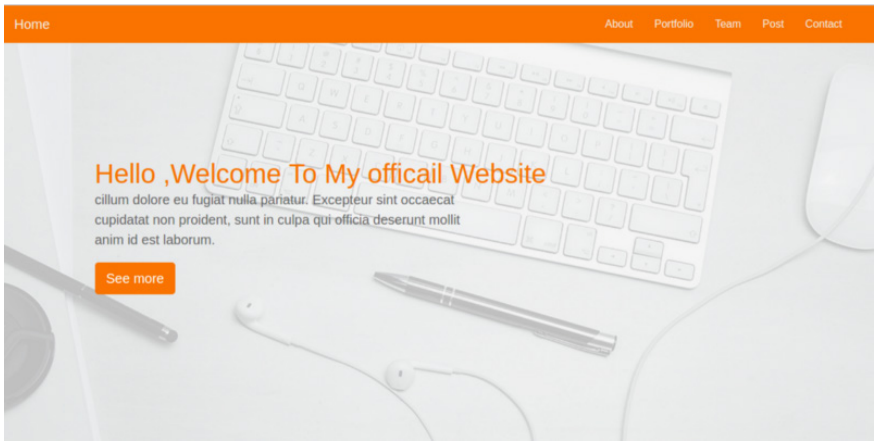

Рисунок 1.3 – Заголовок

V. Розділи

Рисунок 1.4 – Розділи

У цьому пункті буде йти мова про BootstrapGrid, за допомогою якого можна розділити розділ на дві частини. Для того, щоб почати будувати Grid, потрібно присвоїти .row клас для div

Перший розділ буде знаходитись зліва і містити зображення, другий розділ буде знаходитись справа і містити опис. Кожен div буде займати 6 стовпців – це означає половину ділянки. Треба пам’ятати, що Grid ділиться на 12 стовпців. У першому div зліва:

```
<div class="row">
// left side
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/team-3.jpg" class="img-fluid">
<span class="text-justify">S.Web Developer</span>
</div>
</div>
```

Після додавання HTML-елементів справа структура коду буде виглядати так:

```
<div class="row">
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/team-3.jpg" class="img-fluid">
<span class="text-justify">S.Web Developer</span>
</div>
<div class="col-lg-8 col-md-8 col-sm-12 desc">
<h3>D.John</h3>
```

```
<p>
 </p>
</div>
</div>
```

Ось як це виглядає загалом:

```
.about{
 margin: 4em 0;
 padding: 1em;
 position: relative;
}
.about h1{
 color:#F97300;
 margin: 2em;
}
.about img{
 height: 100%;
    width: 100%;
    border-radius: 50%
}
.about span{
 display: block;
 color: #888;
 position: absolute;
 left: 115px;
}
.about .desc{
 padding: 2em;
 border-left:4px solid #10828C;
}
.about .desc h3{
 color: #10828C;
}
.about .desc p{
 line-height:2;
 color:#888;
}
```

VI. Розділ для Портфоліо

Тепер потрібно перейти до наступного кроку і зробити розділ портфоліо, який буде містити галерею.

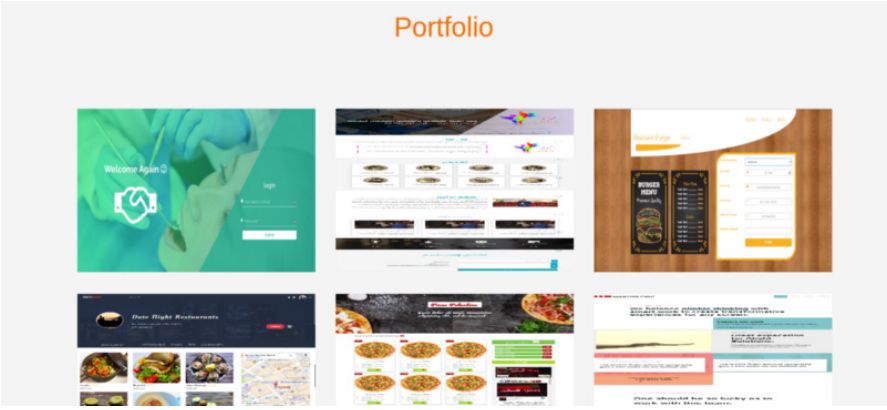

Рисунок 1.5 – Портфоліо

Структура HTML-коду в розділі Портфоліо виглядає так:

```
<!-- portfolio -->
<div class="portfolio">
<h1 class="text-center">Portfolio</h1>
<div class="container">
<div class="row">
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/portfolio/port13.png" class="img-fluid">
</div>
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/portfolio/port1.png" class="img-fluid">
</div>
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/portfolio/port6.png" class="img-fluid">
</div>
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/portfolio/port3.png" class="img-fluid">
</div>
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/portfolio/port11.png" class="img-fluid">
</div>
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/portfolio/electric.png" class="img-fluid">
</div>
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/portfolio/Classic.jpg" class="img-fluid">
</div>
```

```
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/portfolio/port1.png" class="img-fluid">
</div>
<div class="col-lg-4 col-md-4 col-sm-12">
<img src="images/portfolio/port8.png" class="img-fluid">
</div>
</div>
</div>
</div>
```

Додавання класа .img-fluid до кожного зображення робить його плаваючим.

Кожен елемент в галереї буде займати 4 колонки (col- md -4 для середніх пристроїв, col-lg-4 для великих пристроїв). Це дорівнює 33.33333% на таких великих пристроях, як настільні комп'ютери і великі планшети, і 12 колонок на невеликому пристрої (iPhone та інші мобільні пристрої) будуть займати 100% ємності.

Код виглядатиме так:

```
/*Portfolio*/
.portfolio{
 margin: 4em 0;
   position: relative;
}
.portfolio h1{
 color:#F97300;
 margin: 2em;
}
.portfolio img{
  height: 15rem;
  width: 100%;
  margin: 1em;
}
```

VII. Розділ Блога

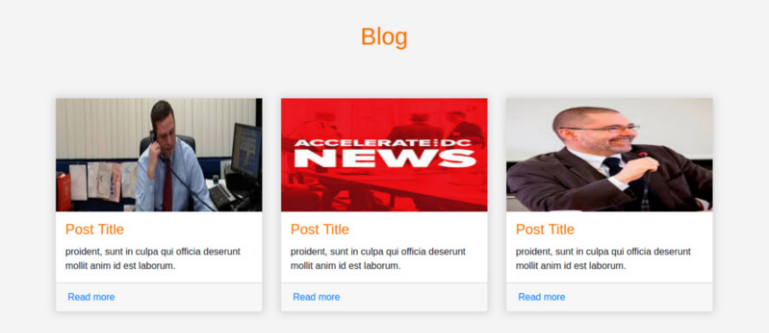

Рисунок 1.6 – Блог

Карти

Карти в Bootstrap 4 допомагають зробити дизайн блога набагато простіше, вони підходять для статей і повідомлень. Для того, щоб створити карту, потрібно використовувати клас .card і задати його в елементі *div*.

Клас карти містить безліч функцій:

- .card-header: Визначає заголовок карти
- .card-body: Для тіла карти
- .card-title: Назва карти
- card-footer: Визначає колонтитул карти.
- .card-image: Для рисунка карти

Отже, сайт в HTML має виглядати приблизно так:

```
<!-- Posts section -->
<div class="blog">
<div class="container">
<h1 class="text-center">Blog</h1>
<div class="row">
<div class="col-md-4 col-lg-4 col-sm-12">
<div class="card">
<div class="card-img">
<img src="images/posts/polit.jpg" class="img-fluid">
</div>
<div class="card-body">
<h4 class="card-title">Post Title</h4>
<p class="card-text">
    proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
</p>
</div>
<div class="card-footer">
<a href="" class="card-link">Read more</a>
</div>
</div>
</div>
<div class="col-md-4 col-lg-4 col-sm-12">
<div class="card">
<div class="card-img">
<img src="images/posts/images.jpg" class="img-fluid">
</div>
<div class="card-body">
<h4 class="card-title">Post Title</h4>
<p class="card-text">
    proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
</p>
</div>
<div class="card-footer">
<a href="" class="card-link">Read more</a>
```

```
</div>
</div>
</div>
<div class="col-md-4 col-lg-4 col-sm-12">
<div class="card">
<div class="card-img">
<img src="images/posts/imag2.jpg" class="img-fluid">
</div>
<div class="card-body">
<h4 class="card-title">Post Title</h4>
<p class="card-text">
      proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
</p>
</div>
<div class="card-footer">
<a href="" class="card-link">Read more</a>
</div>
</div>
</div>
</div>
</div>
</div>
```

Далі необхідно додати деякі CSS-стилі для карти:

```
      .blog{
 margin: 4em 0;
 position: relative;
}
.blog h1{
 color:#F97300;
 margin: 2em;
}
.blog .card{
 box-shadow: 0 0 20px #ccc;
}
.blog .card img{
 width: 100%;
 height: 12em;
}
.blog .card-title{
 color:#F97300;
}
.blog .card-body{
 padding: 1em;
}
```

Після додавання розділу блога на сайт, дизайн має виглядати приблизно так:

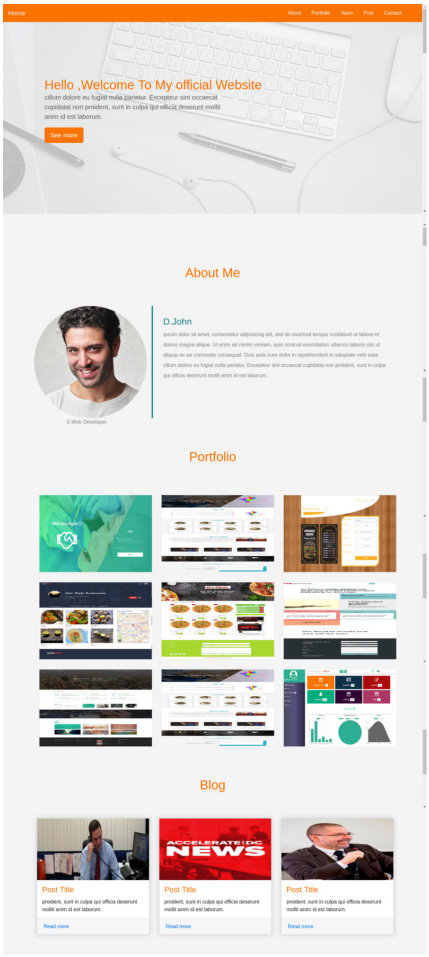

Рисунок 1.7 – Дизайн на даний момент

VIII. Розділ команд

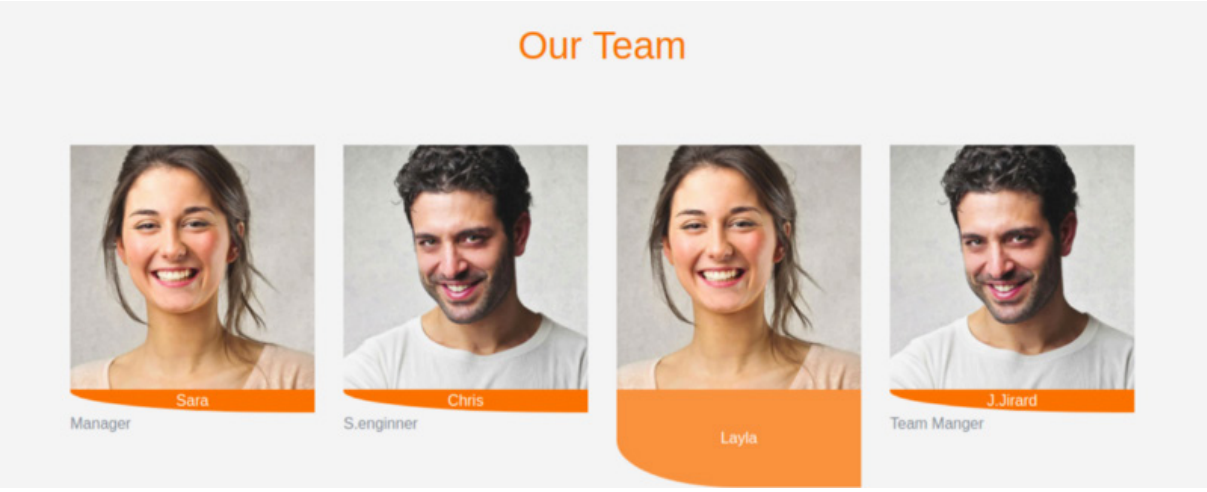

Рисунок 1.8 – Команда

У цьому розділі буде використовуватись підвісна система, щоб розподілити простір між зображеннями. Кожне зображення займає 3 колонки (**.col-md-3**), тобто, пропуск буде становити 25% від загальної площі.

Структура HTML:

```
<!-- Team section -->
<div class="team">
<div class="container">
<h1 class="text-center">Our Team</h1>
<div class="row">
<div class="col-lg-3 col-md-3 col-sm-12 item">
<img src="images/team-2.jpg" class="img-fluid" alt="team">
<div class="des">
    Sara
</div>
<span class="text-muted">Manager</span>
</div>
<div class="col-lg-3 col-md-3 col-sm-12 item">
<img src="images/team-3.jpg" class="img-fluid" alt="team">
<div class="des">
     Chris
</div>
<span class="text-muted">S.enginner</span>
</div>
<div class="col-lg-3 col-md-3 col-sm-12 item">
<img src="images/team-2.jpg" class="img-fluid" alt="team">
<div class="des">
    Layla
</div>
<span class="text-muted">Front End Developer</span>
</div>
<div class="col-lg-3 col-md-3 col-sm-12 item">
<img src="images/team-3.jpg" class="img-fluid" alt="team">
<div class="des">
    J.Jirard
</div>
<span class="text-muted">Team Manger</span>
</div>
</div>
</div>
</div>
```

Тепер потрібно додати стиль до цього розділу:

```
.team{
margin: 4em 0;
position: relative;
```

```
}
.team h1{
 color:#F97300;
 margin: 2em;
}
.team .item{
 position: relative;
}
.team .des{
 background: #F97300;
 color: #fff;
 text-align: center;
 border-bottom-left-radius: 93%;
 transition:.3s ease-in-out;
      }
```

За допомогою анімації створиться ефект наведення курсора на зображення.

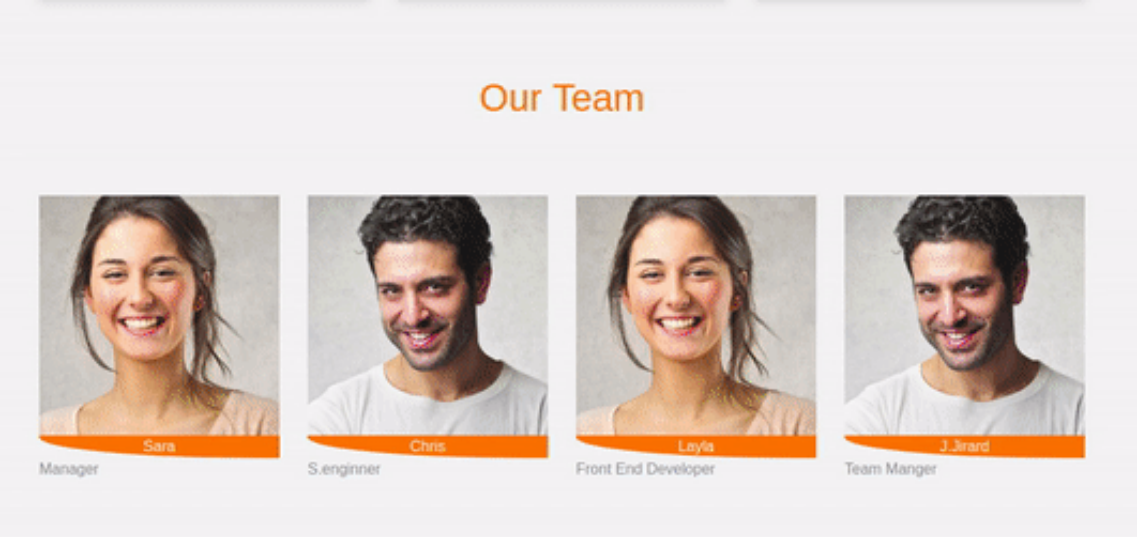

Рисунок 1.9 – Розділ команда

Для того, щоб зробити цей ефект, потрібно додати код, наведений нижче у main.css файл:

```
      .team .item:hover .des{
 height: 100%;
 background:#f973007d;
 position: absolute;
 width: 89%;
 padding: 5em;
 top: 0;
 border-bottom-left-radius: 0;
}
```

IX. Контактна форма

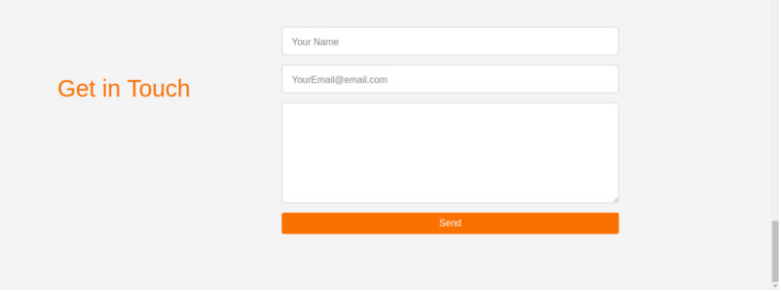

Рисунок 1.10 – Контактна форма

Контактна форма – це останній розділ, який потрібно додати. Контактна форма буде містити форму, через яку відвідувачі зможуть надіслати електронний лист або отримати зворотний зв'язок. У цій частини необхідно буде використовувати деякі класи Bootstrap, щоб зробити дизайн більш сучасним і оптимізованим.

Додаткова інформація розміщена у Bootstrap у 4 документі. В формі потрібно буде додавати кожне нове поле для введення інформації в div і додавати до нього клас .form-group.

index.html файл виглядає так:

```
<!-- Contact form -->
<div class="contact-form">
<div class="container">
<form>
<div class="row">
<div class="col-lg-4 col-md-4 col-sm-12">
<h1>Get in Touch</h1>
</div>
<div class="col-lg-8 col-md-8 col-sm-12 right">
<div class="form-group">
<input type="text" class="form-control form-control-lg" placeholder="Your Name"
name="">
</div>
<div class="form-group">
<input type="email" class="form-control form-control-lg"
placeholder="YourEmail@email.com" name="email">
</div>
<div class="form-group">
<textarea class="form-control form-control-lg">
</textarea>
</div>
<input type="submit" class="btn btn-secondary btn-block" value="Send" name="">
</div>
</div>
</form>
</div>
</div>
```

Зміна стилю в **main.css**

```
.contact-form{
 margin: 6em 0;
 position: relative;
}
.contact-form h1{
 padding:2em 1px;
 color: #F97300;
}
.contact-form .right{
 max-width: 600px;
}
.contact-form .right .btn-secondary{
 background:  #F97300;
 color: #fff;
 border:0;
}
.contact-form .right .form-control::placeholder{
 color: #888;
 font-size: 16px;
}
```

X. Шрифти

Для роботи будуть використовуватись шрифти APIGoogleFont завдяки їх більш сучасному вигляду порівняно зі стандартними, а також Raleway, який підходить для створюваного шаблону. Далі необхідно додати цей код у **main.css** файл:

```
@importurl('https://fonts.googleapis.com/css?family=Raleway');
і встановіть глобальний стиль HTML і теги для заголовка:
html,h1,h2,h3,h4,h5,h6,a{
 font-family: "Raleway";
}
```

XII. Ефект «скролінгу»

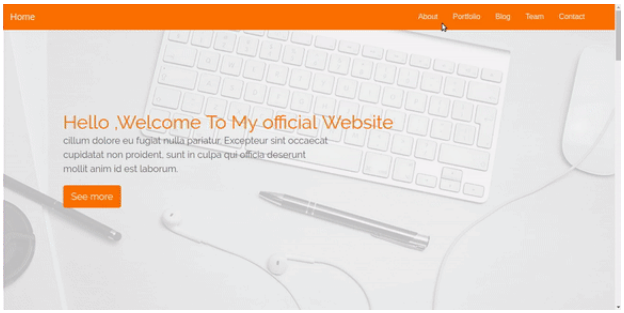

Рисунок 1.11 – Ефект прокручування

Останнє, що залишилося, – це додати відсутній ефект прокручування. Тут потрібно використовувати деякі частини коду з JQuery. Необхідно додати цей код в **main.js** файл:

```
    $(".navbar a").click(function(){
 $("body,html").animate({
  scrollTop:$("#" + $(this).data('value')).offset().top
 },1000)
})
```

Після чого потрібно додати data-value для кожного посилання NavBar:

```
    <li class="nav-item">
<a class="nav-link" data-value="about" href="#">About</a>
</li>
<li class="nav-item">
<a class="nav-link " data-value="portfolio" href="#">Portfolio</a>
</li>
<li class="nav-item">
<a class="nav-link " data-value="blog" href="#">Blog</a>
</li>
<li class="nav-item">
<a class="nav-link " data-value="team" href="#">
      Team</a>
</li>
<li class="nav-item">
<a class="nav-link " data-value="contact" href="#">Contact</a>
</li>
```

Та встановити id для кожного розділу.

Примітка. id мають бути ідентичні data-value атрибуту для правильної роботи:

.

Контрольні запитання

1. Для чого потрібен doctype і скільки різновидів ви можете назвати?
2. У чому різниця між standards mode і quirks mode?
3. Як потрібно оформляти сторінку, у якій контент може бути різними мовами?
4. Що потрібно мати на увазі при розробці багатомовних сайтів?
5. Чи можна використовувати синтаксис XHTML у HTML5?
6. Як використовувати XML у HTML5?
7. Чим корисні data-атрибути?
8. Які box-моделі існують у HTML4 і чи є відмінності у HTML5?
9. Якщо розглядати HTML5 як відкриту web-платформу, на чому вона будується, з яких компонентів складається?
10. Поясніть різницю між cookies, sessionStorage і localStorage.

Лабораторна робота № 2

Тема: Оптимізація сайту – портфоліо за допомогою CSS-змінних

Мета: навчитись використовувати CSS-змінні при верстанні власного сайту

Теоретичні відомості

Новою особливістю CSS є змінні, вона дає змогу створювати змінні таблиці стилів без необхідності робити будь-які налаштування. CSS-змінні дозволяють спростити «старий» спосіб налаштування стилів [2]:

```
    h1 {
  font-size: 30px;
}
    navbar > a {
  font-size: 30px;
}
    ... замість цього:
    :root {
  --base-font-size: 30px;
}
    h1 {
  font-size: var(--base-font-size);
}
    navbar > a {
  font-size: var(--base-font-size);
}
```

Хоча синтаксис може здатися трохи незвичним, він дає очевидну перевагу, оскільки тепер можна змінювати розмір шрифту в усьому додатку, замінюючи лиш змінну --base-font-size [3].

Хід роботи

1. Встановлення

У даній роботі буде оптимізований сайт портфоліо, який виглядає так:

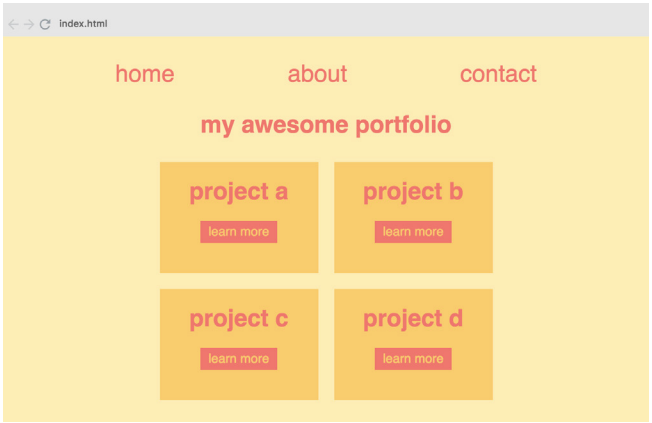

Рисунок 2.1 – Вигляд сайту на персональному комп’ютері

Ось такий вигляд сайт буде мати, якщо дивитися на нього на звичайному робочому столі. Проте, як можна помітити на нижньому зображенні, ця схема не працює у мобільному телефоні.

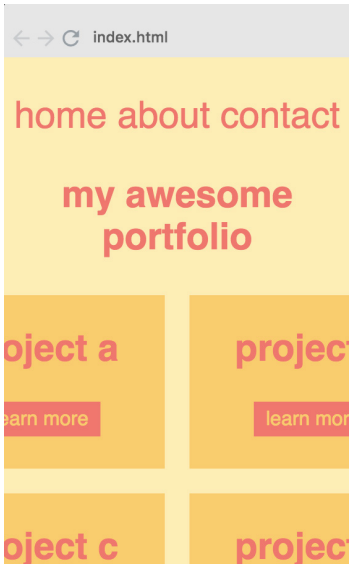

Рисунок 2.2 – Вигляд сайту на мобільному пристрої

Зверху – це те, як зображення виглядає на мобільному телефоні спочатку. Знизу – це те, як воно має виглядати.

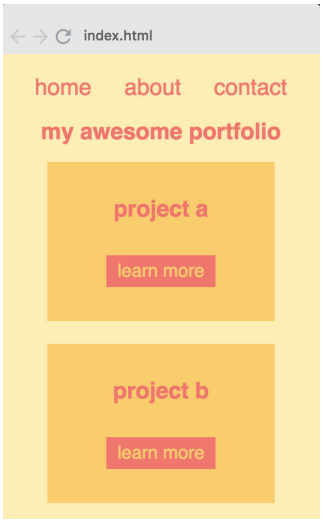

Рисунок 2.3 – Ефект прокручування

Щоб оптимізувати схему для відображення на мобільному телефоні, необхідно змінити кілька речей у налаштуванні стилю.

Переставити grid так, щоб він розміщувався вертикально в 1 колонку, а не йшов горизонтально у 2 колонки.

Перемістити весь текст трохи вгору

Зменшити шрифт.

Для того, щоб зробити це, потрібно змінити нижченаведений CSS:

```
    h1 {
  font-size: 30px;
}
    #navbar {
  margin: 30px 0;
}
    #navbar a {
  font-size: 30px;
}
    .grid {
  margin: 30px 0;
  grid-template-columns: 200px 200px;
}
```

Потрібно зробити такі корегування.

1. Зменшити розмір шрифту h1 до 20px
2. Зменшити відступ вище і нижче #navbar до 15px
3. Зменшити розмір шрифту всередині #navbar до 20px
4. Зменшити відступ над .grid до 15px
5. Змінити вигляд .grid з розміщення в дві колонки до однієї колонки

«Старий» спосіб

Все це можна зробити і без CSS-змінних. [4] Але це вимагає більшої кількості коду, оскільки для більшості з наведених вище пунктів потрібна власна частина запитів для їх зміни. Наприклад така:

```
    @@media all and (max-width: 450px) {

 navbar {
   margin: 15px 0;
 }

 navbar a {
   font-size: 20px;
 }

 h1 {
   font-size: 20px;
 }
       .grid {
   margin: 15px 0;
   grid-template-columns: 200px;
 }
    }
```

Новий спосіб

Це може бути вирішено за допомогою CSS-змінних. Для початку необхідно взяти початковий код, що був наведений наа початку роботи.

```
    :root {
 --base-font-size: 30px;
 --columns: 200px 200px;
 --base-margin: 30px;
```

```
}
```

Тепер потрібно переписати його під програму:

```
#navbar {
  margin: var(--base-margin) 0;
}
#navbar a {
  font-size: var(--base-font-size);
}
h1 {
  font-size: var(--base-font-size);
}
.grid {
  margin: var(--base-margin) 0;
  grid-template-columns: var(--columns);
```

Після переписання коду можна змінити значення змінних всередині для досягнення потрібного результату:

```
@media all and (max-width: 450px) {
  :root {
    --columns: 200px;
    --base-margin: 15px;
    --base-font-size: 20px;
}
```

Новий спосіб набагато простіший, ніж старий. Все, що потрібно, – це зв’язок з root, замість того, щоб пов’язувати кожен елемент з кожним.

Завдяки новому способу, запит скоротився від змінної з чотирисекційного коду до однієї з тринадцяти ліній.

Для того, щоб зрозумітити користь цього способу, потрібно уявити повномасштабний сайт, де, наприклад, безліч --base-margin контролюють більшу частину вільного простору програми. Логічніше об’єднати все в одну змінну, ніж зв’язувати кожну з кожною.

Підводячи підсумок, за CSS-змінними, безумовно, стоїть майбутнє оптимізації кодування.

Контрольні запитання

1. Що таке CSS-змінна?
2. Чим відрізняється верстка за допомогою CSS-змінних від верстки за допомогою Flexbox?
3. Які браузери підтримують CSS-змінні? А які не підтримують?
4. Поясніть, що означає «Семантична розмітка».
5. Які CSS-препроцесори ви знаєте і в чому їх особливості?

Лабораторна робота № 3

Тема: Введення в JavaScript.

Мета: вивчення основ Javascript, введення в JQuery, закріплення знань з CSS3, butons в HTML5.

Теоретичні відомості

JavaScript (**JS**) — динамічна, об'єктно-орієнтована реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що дає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічну та слабку типізацію, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу [5].

Мова JavaScript використовується для:

- написання сценаріїв веб-сторінок для надання їм інтерактивності;
- створення односторінкових веб-додатків (ReactJS, AngularJS, Vue.js);
- програмування на стороні сервера (Node.js);
- стаціонарних додатків (Electron, NW.js);
- мобільних додатків (React Native, Cordova);
- сценаріїв в прикладному ПЗ (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter);
- всередині PDF-документів тощо.

Незважаючи на схожість назв, мови Java та JavaScript є двома різними мовами з відмінною семантикою, хоча й мають схожі риси в стандартних бібліотеках і правилах іменування. Синтаксис обох мов отриманий «у спадок» від мови C, але семантика та дизайн JavaScript є результатом впливу мов Self та Scheme.

Хід роботи

Виконувати дану лабораторну роботу рекомендовано в онлайн редакторі Codepen – https://codepen.io/, де потрібно безкоштовно зареєструватись для створення власних проектів – pens. Профіль викладача – https://codepen.io/barabanchik/.

Крок 1: Створення кнопки

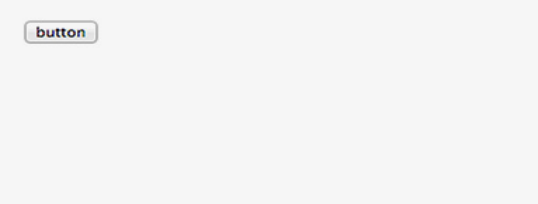

Рисунок 3.1 – Кнопка

- На вкладці HTML, в підпапці , потрібно додати новий тег <button> і написати текст перед закінченням. За замовчуванням можна вказати «click me». Код має виглядати так:

* *
<button>clickme</button>
* *

- Запустити попередній перегляд. Має відображатись кнопка з надписом clickme Це те, що робить buttontag. Це швидкий і простий спосіб зробити елементи, які виглядають як кнопки на веб-сторінках.
- Натиснути на кнопку. Нічого не відбудеться, тому що Buttontag просто створює об’єкт кнопки. Йому не повідомили, що робити далі після кліку. Протягом наступних двох кроків необхідно задавати ці дії.

Крок 2: Зробити клас кольору

Кнопці можна задати виконання таких різних дій, як завантаження зображень, анімація об’єктів, відкриття нових сторінок та ін. Однак для цього проекту потрібно зробити так, щоб кнопка змінювала колір фону. Для цього необхідно:

- Перейти на вкладку CSS і додати новий клас .black. Слід починати своє ім’я класу з відступу, це є позначенням для початку створення класу.
- Додати властивість фону кольору для цього класу. За замовчуванням – це black.

Код має виглядати так:

.black { background-color: black; }

Крок 3: Javascript

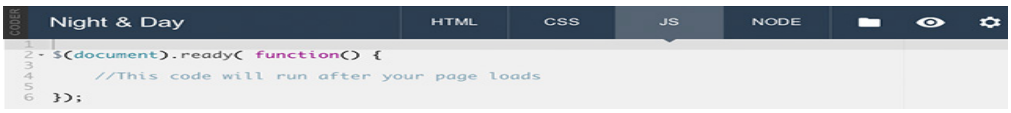

Рисунок 3.2 – Блок з функцією на JS

Потрібно перейти до вкладки JS. Javascript – це мова програмування, яка дозволяє контролювати створені прикладні функції. Javascript можна використовувати для того, щоб реагувати на клацання миші, виконувати обчислення, або анімувати елементи на екрані.

Javascript відрізняється від HTML і CSS граматикою та довжиною коду. HTML і CSS показують тільки те, що написано між тегами і фігурними дужками. Вони статичні, коли їх встановити певним чином, вони залишаються такими самими, поки хтось не поверне їх назад і не змінить. Javascript працює за іншими правилами. Вона може дивитися на те, що відбувається в проекті, і проводити розрахунки або запускати логічні відповіді, які змінюють те, що написано в HTML і CSS.

У більш складних проектах Javascript буде насправді робити складні математичні розрахунки і виконувати такі багатофункціональні методи, як виклик у грі, анімацію форми і т. п.

Крок 4:Синтаксис Jacascript

Рисунок 3.3 – Синтаксис JS

Код, який зображений на рис. 3.3, потрібно перенести у вкладку JS. Він має виглядати так:

$(document).ready(function() { //Thiscodewillrunafteryourpageloads });

Його синтаксис відрізняється від HTML і CSS синтаксису. У даному коді відбувається реалізація функції у JS – *document* є змінною в JavaScript, що дає доступ до об'єктів і змісту у веб-сторінці, яка переглядається у даний момент.

$(document).ready – це подія, яка відбувається в документі, коли він повністю завантажений і готовий до роботи. Досить часто відбувається очікування завантаження інтернет-сторінки перш ніж використовується JavaScript, щоб користуватися його вмістом. Ця подія дозволяє запустити код, коли він потрібен.

function() – це те, як визначається код підпрограми, яка може виконуватися в JavaScript. В даному випадку це знаменує собою початок того, що ми називаємо функцією зворотного виклику, що є функцією, яка викликається, коли подія відбувається.

Фігурні дужки {і} визначають початок і кінець функції, без них код не буде працювати правильно. Вони схожі на теги в HTML і фігурні дужки в CSS. Між цими фігурними дужками будемо писати код, за допомогою якого кнопка буде виконувати вказані їй дії.

Алгоритм функції схожий на вказівку – коли документ буде готовий (при повному завантаженні), виконати дії всередині дужок.

Ця функція запускається перед завантаженням веб-сторінки щоб бути готовою, коли це потрібно. Це ключ для всіх веб-проектів, які використовують Javascript, тому що це те, що допоможе коду працювати правильно.

Крок 5: Ознайомлення з JQuery

У програмуванні є багато різних способів, щоб отримати один і той же проект. Замість того, щоб писати все з нуля, можна використати простий і зручний інструмент під назвою JQuery, щоб допомогти створити кілька простих завдань.

JQuery – це те, що називається бібліотекою, а Codepen потрібно підключити до проекту у налаштуваннях до вкладки JS. Якщо запустити тег <head> і відкрити його, то можна побачити рядок коду, який автоматично завантажує бібліотеку JQuery:

Це виглядає так:

<scriptsrc="/static/common/js/jquery.min.js"></script>

Завантаження бібліотеки JQuery дає легкий доступ до таких корисних речей, як події, CSS-селектори та анімаційні ефекти, які дозволять виконувати різні дії простіше і набагато швидше, ніж написати це в JavaScript.

Знак $ (), який стоїть біля чогось, наприклад, біля документа, наведеного вище, означає JQuery. JQuery вставляє HTML-об'єкти з функціональністю, що дозволяє легко маніпулювати з їх кодом. Це можна зробити також без JQuery, але іноді це може бути складніше. У даній роботі будуть використовуватись функції JQuery, щоб допомогти зберегти невисоку складність Night&Day-проекту.

Крок 6: Створення чорного фону

- Усередині фігурних дужок потрібно створити нову функцію. Вона має виконувати такі дії, а саме: коли кнопка буде натиснута, додати чорний фон CSS в HTML-body.
- За допомогою синтаксису Javascript $('button').click(function) прописується дія того, що кнопка буде натиснута. Подія буде починатись з <button> як мішень всередині $ (). На відміну від документа, потрібно писати ім'я об'єкта і класів в лапках. JQuery має перелік таких подій, як, наприклад, .click; ця подія визначається для виявлення миші на цій кнопці.

Перший рядок має виглядати так.

$('button').click(function(){});

- Тепер необхідно реалізувати механізм, щоб при натисканні кнопки запускалася подія. Для цього потрібно визначити, що буде відбуватись при клацанні миші. В даному випадку – це зміна фону. Між *.click {}* потрібно додати новий рядок, який відповідає за зміну фону.
- За допомогою функції JQuery .addClass можна додати клас .black до фону.
- В кінці необхідно переконатись чи закінчені усі функції і чи закриті всі дужки (), {}, це ті речі, які говорять браузеру, що функція або оператор закінчені.

Код має виглядати таким чином у даний момент:

```
$( document ).ready( function() {
   $( 'button' ).click( function() {
      $( 'body' ).addClass( 'black' );     ); });
```

• Для перевірки роботи даного коду потрібно повернутися в режим попереднього перегляду, натиснути на кнопку та переконатись в тому, що фон почорнів.

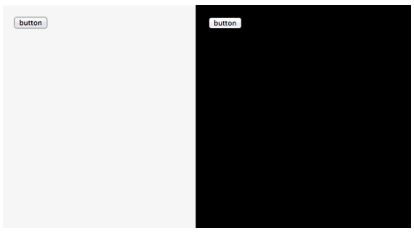

Рисунок 3.4 – Зміна фону

Крок 7: Змінні if/then/else

Зараз кнопка робить одну дію, потім зупиняється, це потрібно змінити. Javascript може бачити те, що відбувається на екрані, й використовувати змінні. Використаємо умовні змінні «if» і «else», щоб перетворити кнопку в перемикач, який може перемикатися між чорним і білим фонами. Для цього потрібно створити функцію, яка перевіряє, який фон стоїть на даний момент, і перемикати його на інший.

• Спочатку необхідно створити новий клас CSS, який буде використовуватись для перемикання – .white;

• На вкладці JS необхідно замінити «старий» код на $('body').addClass('black');

• Далі потрібно зробити так, щоб змінні if/the/else активувались за кліком миші.

Код має виглядати так:

```
$(document).ready( function() {
   $('button').click(function(){
if (thisThingIsTrue) {
doThisThing       }
else { doThisOtherThing       }    }); });
```

• «if» змінна буде створена для того, щоб код перевіряв, чи фон має клас .black, для цього буде використовуватись JQuery: .hasClass.

• Тепер потрібно визначити, що змінна «if» має робити, якщо підтверджується дія. Це «then» частина заяви, все, що в {} може бути прочитано *якщо так, «то».*

Наш код виглядає так:

```
$(document).ready( function() {
   $('button').click(function(){
if($('body').hasClass('black')) {
        $('body').removeClass('black').addClass('white');  }
else { doThisOtherThing       }    }); });
```

• Якщо залишити це в такому вигляді, то програма буде перемикати лише один раз. Щоб це змінити, потрібно створити «else» змінну, щоб вона перевіряла умову if кожен раз і виконувати дію альтернативно до неї. Тобто, це «в іншому випадку».

Остаточний код виглядає так:

```
$(document).ready(function() {
   $('button').click(function(){
if ($('body').hasClass('black')) {
        $('body').removeClass('black').addClass('white');      }
else {          $('body').removeClass('white').addClass('black');      }    }); });
```

Якщо це прочитати, то виглядатиме так: «Коли натиснута кнопка, відбувається перевірка, якщо тіло має чорний клас, то видаляємо цей клас і додаємо білий; в іншому випадку видаляємо білий клас і додаємо чорний».

Тепер необхідно повернутися в режим перегляду і подивитися, як кнопка перемикає фони.

Крок 8: Очищення неба і поява сонця (чи місяця)

Рисунок 3.5 – Місяць

Наступною задачею буде зміна ночі на день і навпаки. Можна скористатися прототипом з попередніх кроків, але тоді потрібно змінити деякі елементи для виконання завдання. З минулого завдання потрібно видалити кнопку та скористатись .

• Необхідно видалити <button> в HTML та створити кнопки, використовуючи ., що дасть більше контролю і свободи в їх оформленні.

• Також потрібно видалити клас .pagecontent на вкладці CSS і відповідний йому .

• На вкладці CSS потрібно зробити новий ідентифікатор для «кнопки.», вона буде перетворюватись в Сонце і Місяць, та назвати його #orb. Тепер потрібно для нього обрати висоту і ширину, однакового розміру, щоб зробити коло величиною 300px.

• Для того, щоб зробити у формі кола, потрібно додати радіус і повернути його на 100%. Завдяки цьому кнопка, як і раніше, має кути, але візуально вона виглядає як коло.

• Також слід позиціонувати наш #orbв центрі.

Після всіх дій, код має виглядати так:

```
#orb {
height: 300px;
```

width: 300px;
border-radius: 100%;
padding: 20px;
margin: auto;
position: absolute;
top: 0; left: 0; bottom: 0; right: 0;
background-color: #blue; }

• Тепер варто зробити новий CSS ID для неба і встановити його як небо на фоні під назвою #sky, зробити це окремо від thebody.

Код буде виглядати так:

#sky {
height: 100%;
width: 100%; }

• Необхідно налаштувати зв'язок в між #orb і #sky

HTML виглядає так:

<body>
* *
* *
</body>

Крок 9: Зробіть стилі для перемикання дня та ночі

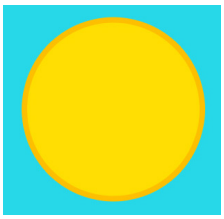

Рисунок 3.6 – Сонце

Тепер потрібно створити класи CSS, які перетворять #orb в .sun і .moon і #sky в .night і.day

• Для початку варто створити #orb – зробити два нових класи та назвати один .sun, а інший – .moon.

• Додати їм колір фону і межі, які відповідають їхнім іменам.

Після чого код виглядає так:

.sun { background-color: #ffdd00;
border: 10px solid #f1c40f; }
.moon { background-color: #bdc3c7;
border: 10px solid #eaeff2; }

• Тепер потрібно зробимо те ж саме для #sky – додати .night і клас, який змінить, залежно від доби, фон #sky від темного до світлого.

Тепер код виглядає так:

.night { background-color: #2c3e50; }
.day { background-color: #37d8e6; }

• Далі потрібно зайти в HTML і застосувати існуючі класи для їх зв'язку . Почати варто з класу, який працює в денний час, і зробити так, щоб він завантажувався при вмиканні сторінки.

Код має виглядати так:

<body>
* *
* *
</body>

Крок 10: Створення Hover

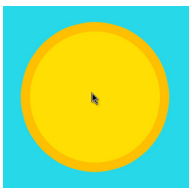

Рисунок 3.7 – Hover

Не всі інтерактивні елементи мають бути зроблені в JavaScript. CSS також може зробити кнопки з такими властивостями, як «hover» (коли курсор знаходиться на об'єкті), «active» (коли курсор натискає на об'єкт) і «visited» (коли об'єкт вже був натиснутий). Потрібно додати hover для .sun і .moon класів, щоб показати, що вони є інтерактивними.

• Потрібно свторити новий рядок в CSS: написати ім'я класу, в який буде додаватись hover, потім додати до тексту: hover {}.

• Слід зробити так, щоб при наведенні збільшувались межі об'єкта.

Код виглядає так:

.sun:hover { border: 20px solid #f1c40f;

• Далі варто зберегти і перейти у вікно попереднього перегляду. Подивитись, що відбувається, коли курсор над Сонцем? Чи стають межі товщими?

• Тепер додати hover до класу .moon, використовуючи той же метод.

Крок 11:Зміна Сонця/Місяця

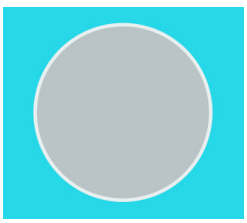

Рисунок 3.8 – Зміна Сонця на Місяць

Тепер варто повернутись до JavaScript. Потрібно змінити декілька окремих слів:

- Задати спочатку .click дію для #orb. Це перетворить його в кнопку.
- Замінити «button» текст всередині $ () на «#orb». Оскільки програма орієнтована на CSS ID, потрібно додати ще # між лапками.
- Наступна зміна торкнеться нового перемикача. Необхідно замінити body з «#orb».
- І, нарешті, замінити класи перемикання між .white і .black, тепер буде відбуватись перемикання між .sun і .moon.

Код має виглядати так:

```
$('#orb').click( function() {
if ($(this).hasClass('sun')) {          $(this).removeClass('sun').addClass('moon');       }
else {          $(this).removeClass('moon').addClass('sun');       }
```

- Варто зберегти код та перейти на попередній перегляд. Коли відбудеться подія кліку на Сонце, має відбутися його зміна Місяцем.

Крок 12: Перемикання День/Ніч

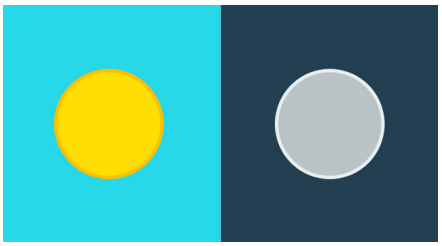

Рисунок 3.9 – Переключення з Дня на Ніч

Наступне завдання – створення нічного неба

- Безпосередньо під «#orb» if/ then /else необхідно додати нові змінні if/ then /else.
- Змінити мету, для «#sky» і класи, «day» і «night».

Код має виглядати так:

```
if ($('#sky').hasClass('day')) {    $('#sky').removeClass('day').addClass('night'); }
else {    $('#sky').removeClass('night').addClass('day'); }
```

- Варто зберегти код та перевірити його.

Крок 13: Додайте кратери на Місяць

Рисунок 3.10 – Кратери на Місяці

Останнє завдання – це додавання декілька місячних кратерів на Місяць.

Створення кратерів буде схожим на #orb, різниця тільки в тому, що в цей раз буде не перемикання між двома стилями, а буде просто їх додавання і видалення.

- Спочатку необхідно записати ідентифікатори кратерів в CSS. Всього буде створено три кратера, згідно з рис. 3.10. Крім ширини і висоти, всі вони матимуть ті ж властивості – 40 px, 80 px і 180 px в діаметрі.
- Ці кратери будуть встановлені всередині .

Перший код для додавання кратера виглядає так. Для того, щоб створити інші кратери, потрібно інкрементувати *moonspot1*:

```
#moonspot1 {
height: 40px;
width: 40px;
border-radius: 100%;
float:right;
margin: 20px; }
```

- Тепер потрібно створити клас, який буде робити кратери видимими, коли місяць на екрані. Все, що потрібно зробити, це додати колір фону, максимально схожий на колір Місяця.

Код виглядає так:

```
.visible { background-color: #eaeff2; }
```

- Тепер необхідно розмістити їх в в HTML.

HTML виглядає так:

```
<div id= "orb" class= "sun">
<div id= "moonspot1"></div>
<div id= "moonspot2"></div>
<div id= "moonspot3"></div>
</div>
```

- Останньою дією в рамках цього .click випадку потрібно змінити всі інші класи та додати ряд нових if / then / else наборів, які будуть додавати та видаляти видимий клас в потрібний час. Потрібно додати один для кожного з кратерів в .

Javascript виглядає так:

```
if ($('#moonspot1').hasClass('visible')) {
   $('#moonspot1').removeClass('visible');
} else {    $('#moonspot1').addClass('visible'); }
```

Контрольні запитання

1. Що таке прототипне спадкування?
2. Як можна використовувати JavaScript для підвищення доступності web-проектів?
3. Що таке спливання подій і чим воно відрізняється від перехоплення подій?
4. Як делегування подій покращує код на сайтах з великою кількістю інтерактивних елементів?
5. Що таке замикання і як вони можуть допомогти в організації коду?
6. Що означає рядок 'use strict' у верхній частині блока коду?
7. Що термін «підняття змінних» означає в застосуванні до JavaScript?
8. Чим стрілкові функції відрізняються від звичайних?
9. В яких ситуаціях слід використовувати ключові слова let і const?
10. Що таке функціональне програмування і які його особливості?

Лабораторна робота № 4

Тема: Вивчення технології AJAX.

Мета: створення веб-сторінки, що генерує випадкові цитати за технологією AJAX.

Теоретичні відомості

AJAX – це не самостійна технологія, а концепція використання декількох суміжних технологій. AJAX – це підхід до розробки, який призначений для користувачів інтерфейсів, комбінує кілька основних методів і прийомів:

- використання DHTML для динамічної зміни змісту сторінки;
- використання XMLHttpRequest для звернення до сервера, не перезавантажуючи всю сторінку повністю;
- альтернативний метод — динамічне підвантаження коду JavaScript в тег <SCRIPT> з використанням DOM, що здійснюється з використанням формату JSON;
- динамічне створення дочірніх фреймів.

Використання цих підходів дозволяє створювати набагато зручніші веб-інтерфейси користувача на тих сторінках сайтів, де необхідна активна взаємодія з ним. AJAX— асинхронний, тому користувач може переглядати далі контент сайту, поки сервер все ще обробляє запит. Браузер не перезавантажує web-сторінку і дані посилаються на сервер без візуального підтвердження (крім випадків, коли необхідно показати процес з'єднання з сервером). Використання AJAX стало популярним після того, як компанія Google почала активно використовувати його при створенні таких своїх сайтів, як Gmail, Google Maps і Google Suggest. Створення цих сайтів підтвердило ефективність використання даного підходу.

Хід роботи

Виконувати дану лабораторну роботу рекомендовано в онлайн редакторі Codepen – https://codepen.io/, – де необхідно безкоштовно зареєструватись для створення власних проектів – pens. Профіль викладача – https://codepen.io/barabanchik/.

Крок 1: Створіть основний HTML / CSS для проекту.

Зосередьте увагу на семантичному HTML та CSS, який допомагає візуалізувати елементи для правильного позиціонування.

Включіть тег <p>, який містить текст цитати, і тег <cite> для автора.

Для вертикальної орієнтації застосуйте flexbox до батьківського елемента та перевірте його, щоб побачити, як виглядає він з короткими та довгими цитатами.

Для розміщеняя елементів на однаковій відстані один від одного, необхідно, щоб кнопки розташовувались на однаковій вілстані. У статті про css-tricks пояснюється, як цього досягти з властивостями display:flex і justify-content:space-between, застосованими до елемента .

Результат роботи виглядає, як показано нижче:

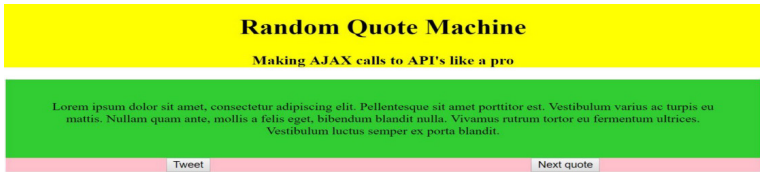

Рисунок 4.1 – Фрагмент сайту

Крок 2: Впровадження функціонала randomquote

Спершу потрібно запустити новий файл script.js і правильно під'єднати його до документа HTML. Перш ніж продовжувати, потрібно впевнитися, що скрипт підключено правильно, наприклад викликом сповіщення («Я правильно підключений»).

Далі необхідно підключити посилання JQuery CDN та перевірити його за допомогою $('body').append ("Jqueryindahouse");

Наступник кроком є створення $(document).ready(function () {// код }); що дозволить сценарію запускатися тільки після того, як весь DOM буде завантажений і готовий до маніпулювання.

Далі необхідно знайти API довільних цитат, який можна використати для свого проекту. Наприклад, існує ринок Mashape: https://market.mashape.com/, тому можна використати API Random Famous Quotes: https://market.mashape.com/andruxnet/random-famous-quotes/. Тим, хто раніше не використовував API та mashape, варто деякий час присвятити вивченню інформації про те, як зробити успішний запит (request). Для пошуку прикладу коду для JavaScript можна скористатись прикладами для node.js і стане зрозумілим, що API вимагає, щоб інформація заголовка (header information) передавалася разом із запитом (request). Перейдемо до запуску пошукової системи.

Інформацію про встановлення заголовків за допомогою .getJSON() можна знайти тут: How can I pass request headers with jQuery's getJSON() method? Як завжди, stackoverflow надає цінну інформацію; потрібно використати більш надійний метод .ajax(), в якому можна встановити інформацію заголовка. Створіть запит ajax, та перевірте його на виконання.

Наприклад, введіть даний запит в консоль:

```
$.ajax({
type: "POST",
url: "https://andruxnet-random-famous-quotes.p.mashape.com/?cat=famous",
dataType: "json",
success: function (response) {
    console.log(response);
```

```
    },
beforeSend: setHeader
  });
    functionsetHeader(xhr) {
xhr.setRequestHeader("X-Mashape-Key", "[myMashapeAPIkey]");
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xhr.setRequestHeader("Accept", "application/json");
}
```

Результат виглядає так.

Рисунок 4.2 – Результат виконання програми

Далі необхідно реалізувати функціональність отримання нової цитати при натисканні кнопки. Для цього необхідно додати виклик ajax у функцію getRandomQuote() і розмістити його як обробник події для кнопки «Наступна цитата»

```
$("#next_quote").click(getRandomQuote);
```

Відображення результату на сторінці легко зробити за допомогою JQuery. Наступна функція – успішний виклик зворотного зв’язку ajax:

```
functiondisplayQuote(response) {
   $("#quote").text(response.quote);
   $('#author').text(response.author);
}
```

Якщо виклик ajax невдалий, то потрібно використати функцію обробки помилок і перевірити, у чому помилка.

Користувачеві програми не потрібно знати, звідки приходить цитата, а саме: це виклик ajax до API чи локальний масив об’єктів цитат. Крім того, маючи масив цитат, варто захопити першу цитату, яка з’являється при завантаженні сторінки, з масиву, і тим самим уникнути затримки, яка відбувається при поверненні запиту ajax. Тим не менш, необхідно імітувати поведінку API і отримувати випадковий показ цитування кожного разу, коли сторінка завантажується. Це можна досягнути з нижчевказаною функцією

```
      functiondisplayQuoteFromArray() {
varmyQuotes = [
        {
author: "AuthorName",
quote:"Veryfamousquote"
        }
        //+ morequoteobjectswiththesameformat
                ];
varrandom = Math.floor(Math.random() * x);
     //x isthenumberofquotesinthearray
displayQuote(myQuotes[random]);
  }
```

Викликайте дану функцію на початку скрипту, щоб отримати першу цитату з масиву. Тут також додано в функцію ajax параметр помилки, який вказує на цю функцію, тому, якщо запит не вдається, функція покаже цитату з масиву. На цьому перший етап роботи завершено, і настав час дослідити, як реалізувати функцію твітування (tweeting).

Документація Twitter допоможе з'ясувати, як попередньо заповнити твіт з цитатою. Виконайте функцію, яка створює URL-адресу твіту, який додається до тегу <a id="tweet">атрибуту href з кнопкою tweet:

```
      $("#tweet").attr("href", tweetQuote()):
      functiontweetQuote() {
vartwitterURL =
'https://twitter.com/intent/tweet?hashtags=quotes,freeCodeCamp&related=freecodecamp&tex
t="';
var quote = $("#quote").text();
var author = $('#author').text();
twitterURL +=quote +'" - '+ author;
      return twitterURL;
   }
```

Далі необхідно перевірити функцію Twitter. Після перевірки стає очевидно, що присутня помилка: tweetURL не оновлюється, коли натискається наступна цитата, відображається попередня цитата, а не поточна.

Це відбувається тому, що tweetQuote() запускається до завершення AJAX. Коли AJAX повертається, він просто оновлює текст, але не URL-адресу твіту.

Щоб виправити дану помилку, необхідно додати виклик до tweetQuote() всередині функції displayQuote. Ця функція буде викликатися AJAX-ом успішно і тоді, коли цитата завантажується з масиву (завантаження сторінки або помилка ajax). Тепер кнопка twitter працює так, як треба. Після натискання, з'являється попередньо заповнений твіт, що містить поточну цитату та автора цитати.

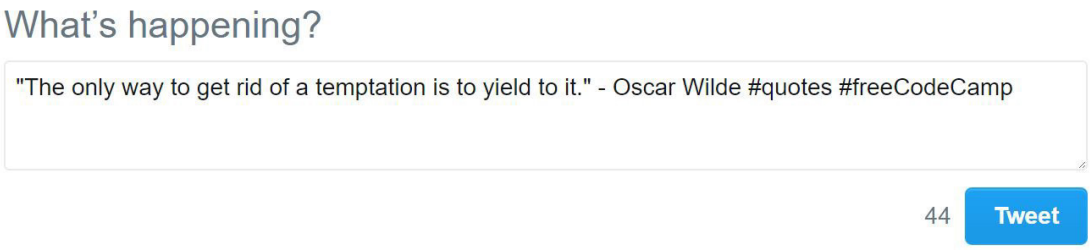

Рисунок 4.3 – Результат роботи програми

Крок 3: Додавання стилів

Використайте наведений нижче стиль для свого веб-додатка:
- Фоновий малюнок від Pixabay;
- Google шрифти: Merriweather&SpecialElite;
- Bootstrapдля стилізування кнопок;
- Шрифт Awesome для додавання піктограм.

Ваш веб-додаток може мати такий вигляд:

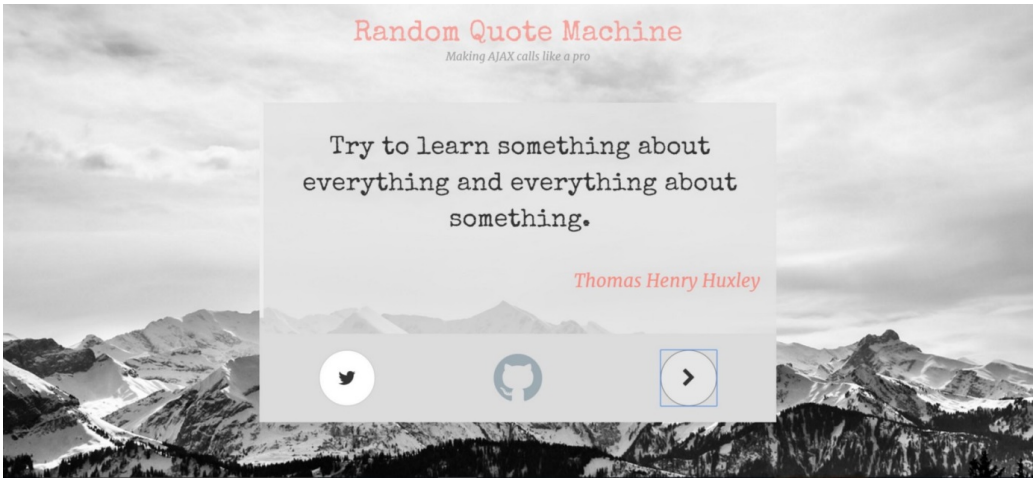

Рисунок 4.4 – Приклад вигляду веб-додатка

Контрольні запитання

1. Коли необхідно використовувати елементи _escaped_fragment_ чи #! в URL ресурсах, які використовують AJAX?
2. Чому необхідно використовувати елемент #! в URL ресурсах AJAX?
3. Скільки знімків HTML потрібно використовувати?
4. Чи провокує технологія AJAX використання маскування?
5. Чи допоможе AJAX підвищити індексованість файлів Flash і інших мультимедійних об'єктів?
6. Що можна зробити, якщо #! вже використовується в хеш-фрагментах?
7. Чи підтримує AJAX спеціальні можливості?
8. Як використовувати rel="canonical"?
9. Який URL рекомендується використовувати в Sitemap?

Лабораторна робота № 5

Тема: Створення першого проекту на React.

Мета: навчитися додавати бібліотеку React до веб-сторінки і створити перший проект на ній.

Теоретичні відомості

React.js – це всього лише бібліотека, а не фреймворк.

Як і будь-яка інша бібліотека, реакт додається на сторінку за допомогою тегу script.

Оскільки в сучасному js популярні модулі та різного роду перетворення/стискання і так далі, react відмінно працює з webpack, babel та іншим. Для простоти роботи, ми будемо працювати з реакт, як зі звичайною бібліотекою, наприклад, jQuery.

До речі, бібліотека jQuery для роботи реакта не потрібна. Проте потрібні деякі допоміжні бібліотеки, які поставляє сам реакт, або його аналоги в особі babel. [6]

Хід роботи

Насамперед завантажте react – https://reactjs.org/.

Далі необхідно створити структуру файлів і папок. Всі нескопійовані файли поки що залиште порожніми.

```
      +-- js
|   +-- react
|      +-- react-dom.js (скопіюйте з архіву з папки build)
|      +-- react.js (скопіюйте з архіву з папки build)
|   +-- app.js
+-- index.html
+-- package.json
+-- server.js
```

Для роботи можна використовувати localhost, або створити локальний сервер – server.js, щоб працювала вкладка в консолі ReactDeveloperTools, яка допоможе налагоджувати React-додаток.

Перш за все необхідно створити файл package.json, який є інструкцією/описом для проекту package.json

```
      {
"name":"react-ua-tutorial",
"version":"1.0.0",
"description":"React UAtutorial",
"main":"index.html",
"scripts":{
"start":"node server.js"
},
"author":"SerhiiBaraban",
```

```
"license":"MIT"
}
```

Далі створимо за допомогою Node.js* і express локальний сервер.

*Якщо у вас не встановлений Node.js, то необхідно скачати і встановити його з офіційного сайту (https://nodejs.org/en/). Якщо ви використовуєте Windows, для виконання команд нижче рекомендується консоль – ConEmu (https://www.fosshub.com/ConEmu.html).

Додамо express в проект

```
npminstallexpress --save-dev
```

При використанні прапорця --save-dev пакет express буде додано до списку залежностей проекту. Ви можете подивитися як змінився файл package.json, щоб переконатися в цьому.

server.js

```
varexpress=require('express');
varapp=express();

app.set('port',(process.env.PORT||3000));

app.use('/',express.static(__dirname));

app.listen(app.get('port'),function(){
  console.log('Server started: http://localhost:'+app.get('port')+'/');
});
```

Наступним кроком заповніть файл index.html

index.html

```
<!DOCTYPE html>
<html>
<head>
<title>React [RU] Tutorial</title>
</head>
<body>
<div id="root">Привіт, я #root</div>
</body>
</html>
```

Далі запустіть сервер node server.js.

Так само сервер можна запускати за допомогою команди npmstart, це можливо, тому що в файлі package.json є секція scripts, в якій вказана команда start. Такий варіант зручно використовувати, коли для запуску сервера доводиться писати набагато більше, наприклад:

```
node server.js -option1 -option2 CONST = qweqwe ...
```

Досить буде вказати повну команду в файлі-інструкції, і потім зручно використовувати її за допомогою скороченого варіанта. Так, насправді це alias (скорочення) для вашого проекту.

Переконайтеся, що у вас в браузері з'явився рядок «Привіт, я #root», і якщо це так, то далі підключимо react і почнемо його використовувати.

Отож, відразу підключимо наш app.js, в якому будемо писати код.

index.html

```
<!DOCTYPE html>
<html>
<head>
<title>React [RU] Tutorial</title>
</head>
<body>
<div id="root">Привіт, я #root</div>

<scriptsrc="js/react/react.js"></script>
<scriptsrc="js/react/react-dom.js"></script>
<scriptsrc="js/app.js"></script>
</body>
</html>
```

js/app.js

```
console.log(React);
console.log(ReactDOM);
```

Перевіримо у браузері:

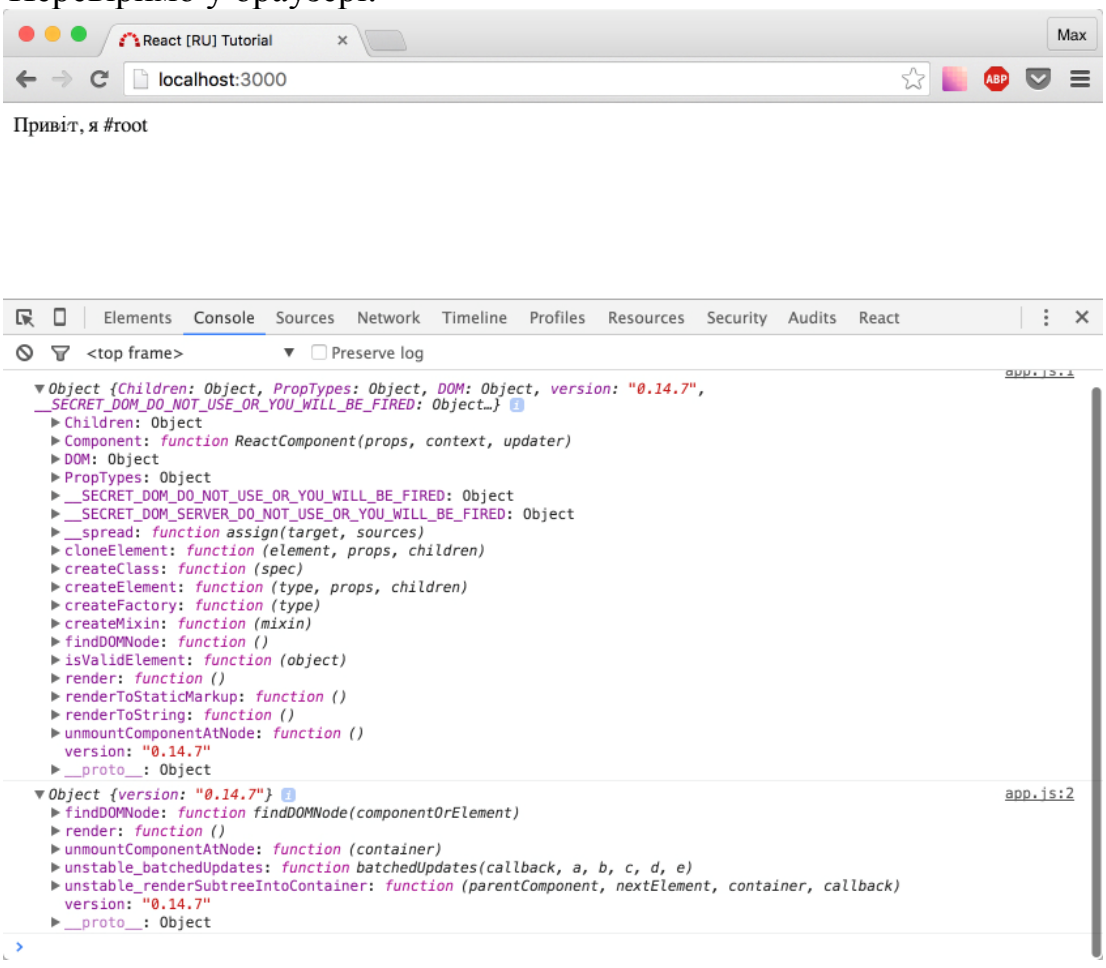

Рисунок 5.1 – Результат роботи в браузері

Спробуємо вивести що-небудь за допомогою react.

Змінимо app.js:

js / app.js

```
ReactDOM.render(
React.createElement('h1',null,'Привіт, Світ!'),
document.getElementById('root'));
```

Дана операція додає заголовок (<h1></ h1>) за допомогою react.

Тут була використана функція ReactDOM.render, яка приймає першим аргументом реакт-компонент, а другим – елемент DOM-дерева, куди необхідно додати react.

React необхідний, тому що при змінах DOM-дерева він використовує мінімально можливі впливи. React використовує «віртуальний DOM» (видаляє/змінює/додає елементи і т. д.) для того, щоб в реальний DOM за один раз додати всі зміни. Як відомо, операції з DOM-деревом найдорожчі. Тому «інтелектуальний» підхід реакту до них – це його особливість.

Далі потрібно переути у файл app.js

Команда розробників реакту вирішила писати розмітку в javascript-коді і назвала такий синтаксис JSX.

У такої техніки є затяті противники, але користі від використання реакту набагато більше, ніж від поділу на розмітку і скрипти.

Вдосконалимо наш код.

s / app.js

```
ReactDOM.render(
<h1>Hello,world!</h1>,
document.getElementById('root')
);
```

Оскільки React – javascript бібліотека, а ми пишемо JSX синтаксис всередині звичайного .js файлу, необхідно якось навчити браузер перетворювати

```
<h1>Hello, world!</h1>
```

```
React.createElement('h1', null, 'Привіт, Світ!')
```

Для цього існує babel. Тому необхідно додати ще один скрипт на сторінку https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.js Скачайте його, або підключіть через CDN. Збережіть в папку js/react.

Так само для скрипта потрібно вказати атрибут type = 'text/babel'

index.html

```
<!DOCTYPE html>
<html>
<head>
<title>React [RU] Tutorial</title>
</head>
<body>
<div id="root"></div>

<scriptsrc="js/react/react.js"></script>
<scriptsrc="js/react/react-dom.js"></script>
<scriptsrc="js/react/browser.min.js"></script>
<scripttype="text/babel"src="js/app.js"></script>
</body>
</html>
```

Отож, тепер можна писати звичну HTML-розмітку всередині «реакт-коду».

Важливо розуміти, що Babel перетворює HTML-розмітку в Javascript код.

Створення компонента

Нагадаємо, що ReactDOM.render приймає react-компонент і DOM-елемент, в який необхідно використати в додатку.

<H1>Hello, world! </ H1> – це примітивний компонент. Створимо примітивний компонент;

js/App.js

```
varApp=React.createClass({
render:function(){
return(
<div className="app">
Всім привіт, я компонент App!
</div>
);
}
});

ReactDOM.render(
<App/>,
document.getElementById('root'));
```

В браузері можна побачити розмітку, яку було вказано в методі Render, компонента App.

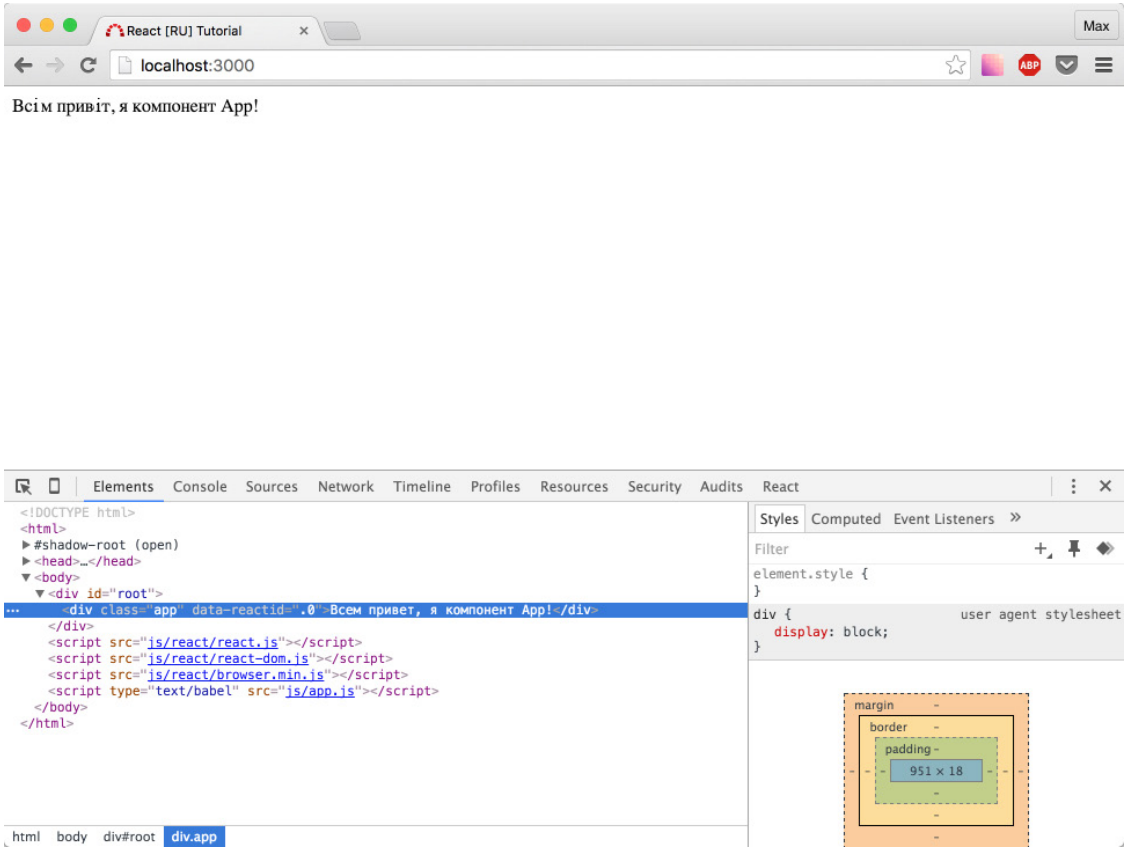

Рисунок 5.2 – Результат роботи в браузері

data-reactid = ". 0", додав уже сам реакт.

Слово class в javascript є зарезервованим, тому всередині JSX-синтаксису необхідно писати className.

У ReactDOM.render ми передали повністю наш компонент, без властивостей.

Запис <App /> еквівалентний<App></ App>.

Отож: у нас є компонент <App />

Необхідно розвинути ідею, і додати ще щось всередину App, і відобразимо це

js/App.js

```
varNews=React.createClass({
render:function(){
return(
<div className="news">
        На жаль, новин немає.
</div>
);
}
});

varApp=React.createClass({
render:function(){
```

```
return(
<div className="app">
Всім привіт, я компонент App! Я вмію відображати новини.
<News/>
</div>
);
}
});

ReactDOM.render(
<App/>,
document.getElementById('root')
);
```

По-перше – код всередині ReactDOM.render ніяк не змінився. Компонент <App /> містить зараз в собі інший компонент, але, при цьому, це ніяк не впливає на візуалізацію всього додатка.

По-друге, як уже було сказано, – компонент <App /> містить в собі компонент <News />. Так само, як якщо б це був просто дочірній елемент.

По-третє, компонент <News /> такий же примітивний, як і App, і містить всього один (обов'язковий) метод render.

Потрібно створити ще один компонент <Comments /> і зробити, щоб він відображався після новин. Текст компонента: «Немає новин – коментувати нічого.»

Рішення

js / App.js

```
varNews=React.createClass({
render:function(){
return(
<div className="news">
      На жаль, новин немає.
</div>
);
}
});

varComments=React.createClass({
render:function(){
return(
<div className="comments">
Немає новин – коментувати нічого.
</div>
);
}
});
varApp=React.createClass({
```

```
render:function(){
return(
<div className="app">
Всім привіт, я компонент App! Я вмію відображати новини.
<News/>
<Comments/>
</div>
);
}
});

ReactDOM.render(
<App/>,
document.getElementById('root')
);
```

Відкрийте браузер і перевірте розмітку сторінки.

Щоб не використовувати стандартний спосіб перегляду в консолі, необхідно встановити reactdevtools (плагін для хрому – https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi).

Після встановлення, відкрийте вкладку React в консолі розробника.

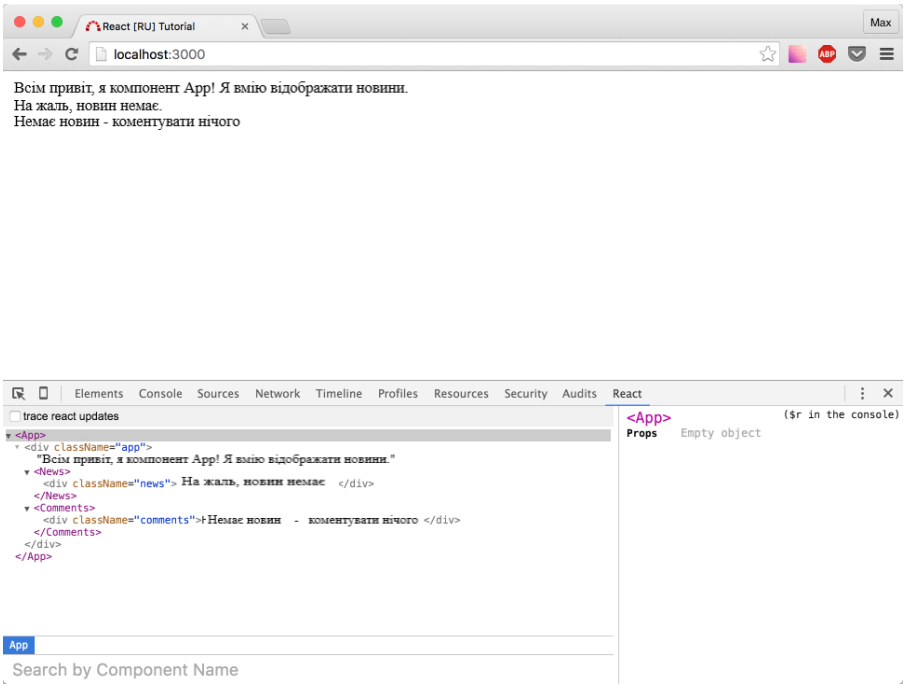

Рисунок 5.3 – Консоль розробника

У кожного компонента можуть бути властивості (вікно Props). Вони зберігаються в this.props і передаються компоненту як атрибути.

Загальний вигляд:

```
var value1 ={name:Garry,surname:Potter};
```

```
<MyComponentdata={value1}eshe_odno_svoistvo={[1,2,3,4,5]}/>
```

У властивість можна передати будь-який javascript-примітив, об’єкт, змінну і навіть вираз. Значення властивості має бути взято у фігурні дужки.

Значення доступні через this.props.IM’Я_ВЛАСТИВОСТІ

У нашому випадку, ми отримаємо:

this.props.data – об’єкт {name: Garry, surname: Potter}

this.props.eshe_odno_svoistvo – масив [1,2,3,4,5]

this.props використовуються тільки для читання.

Створимо кілька новин для нашого додатка.

Додайте на початок файлу js / app.js

```
varmy_news=[
{
author:'СашаПєчкін',
text:'В четвер, четвертого числа...'
},
{
author:'Просто Вася',
text:'Вважаю, що $ має коштувати 25 гривень!'
},
{
author:'Гість',
text:'Безкоштовно. Скачати. Кращий сайт - http://localhost:3000'
}
];
```

І змініть рядок з підключенням компонента News таким чином:

```
var App =React.createClass({
render:function(){
return(
<divclassName="app">
      Всім привіт, я компонент App! Я вмію відображати новини.
<Newsdata={my_news}/>{/*додали властивість data */}
<Comments/>
</div>
);
}
});
```

Зверніть увагу, коментарі всередині JSX пишуться у фігурних дужках: {/ * текст коментаря * /}

Так само прошу зауважити, JSX це не весь js-код, який міститься в App.js, грубо кажучи JSX – це HTML-розмітка + змінні/вирази. Тому в інших місцях можна писати коментарі звичним способом (// ... або /*...*/).

Відкрийте в консолі вкладку react (попередньо обновивши сторінку).

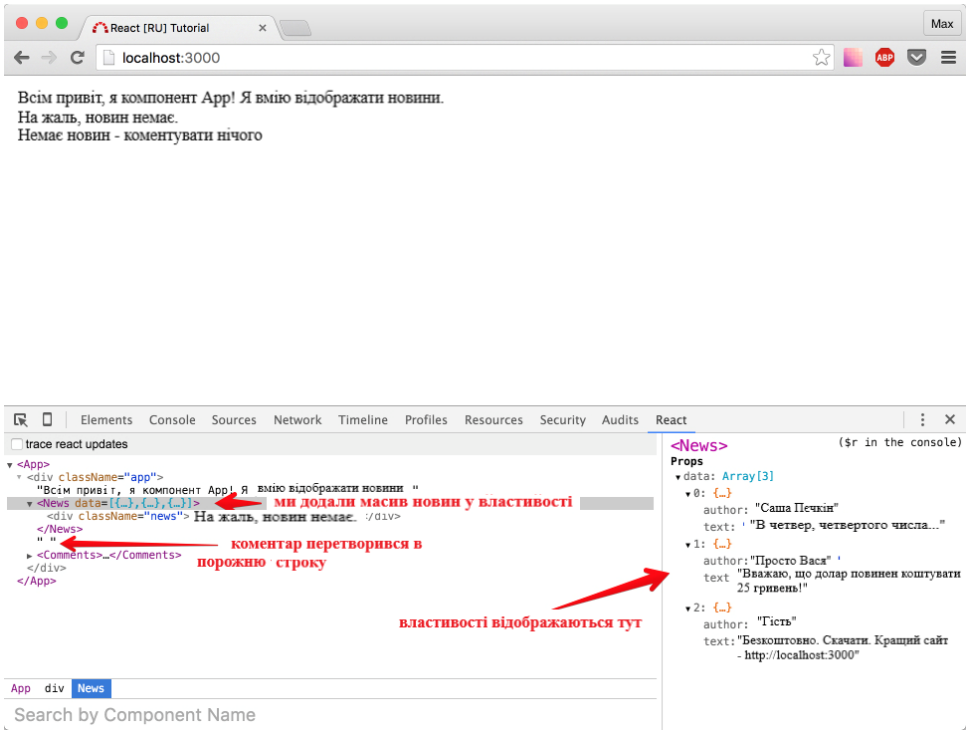

Рисунок 5.4 – Сторінка в консолі

В даний момент, ми додали властивість data в компонент <News />. Необов'язково було називати властивість так, можна було написати, наприклад:

<Newsposledine_novosti = {my_news} />

Тоді в консолі розробника, це виглядало б так:

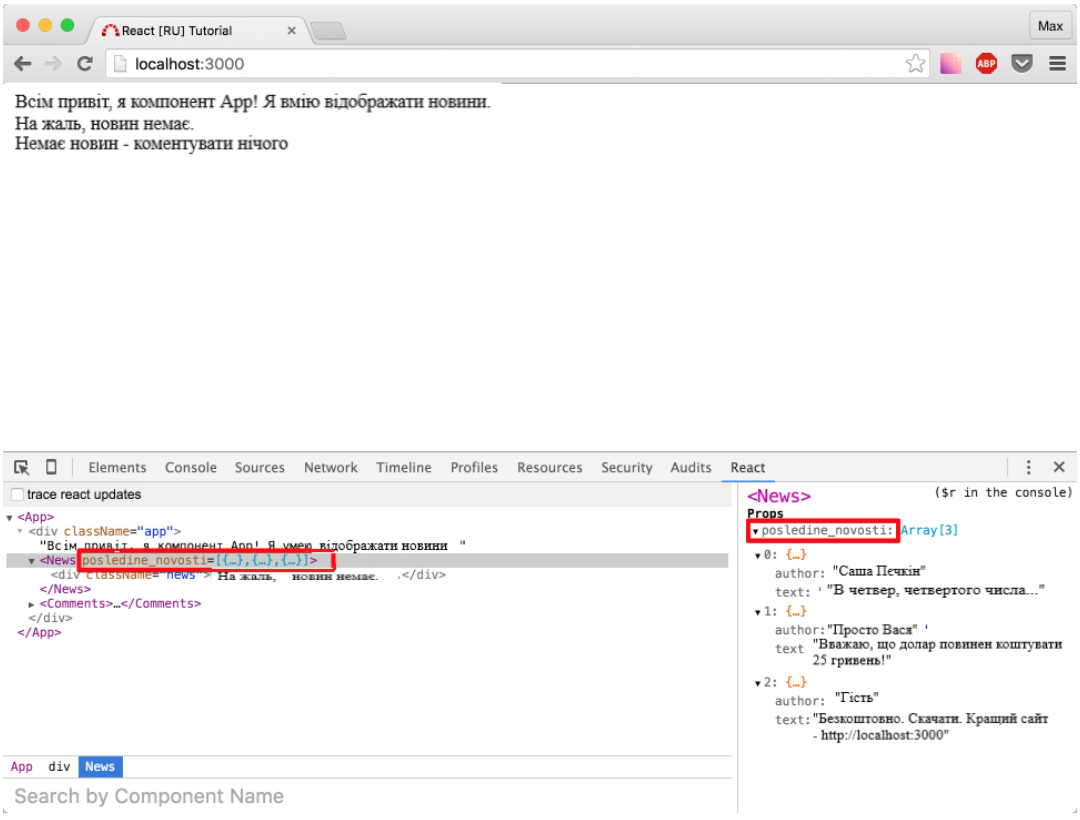

Рисунок 5.5 – Консоль розробника

Зараз у компонента є властивість, в якій знаходяться новини, але компонент не вміє їх відображати. Це легко виправити.

Уявімо HTML-розмітку для наших новин

```
<div class="news">
<p class="news__author">Саша Пєчкін:</p>
<p class="news__text">В четвер, четвертого числа...</p>
</div>
```

Існує розмітка для одного елемента даних, є дані повністю (масив my_news), для відображення цих даних потрібно створити шаблон, пройтися по всіх змінних в масиві з новинами і підставити значення.

Коли потрібно відобразити змінну в шаблоні розмітки – вона так само береться в фігурні дужки. На практиці це виглядає простіше, ніж в теорії, тому уявімо, як може виглядати наша JSX-розмітка

```
this.props.data.map(function(item, index){
return(
<div key={index}>
<p className="news__author">{item.author}:</p>
<p className="news__text">{item.text}</p>
</div>
)
})
```

Ми використовували метод масивів Map. Якщо ви не знайомі з ним, прочитайте документацію –

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Після обгортання розмітки всередині return в кореневий елемент можна її обгорнути в будь-який інший елемент, головне, що потрібно запам’ятати, – всередину return завжди повинен повертатися DOM-вузол (тобто, що завгодно, обгорнуте в батьківський тег).

Ми використовували у батьківському елементі атрибут key ( ). Якщо пояснити гранично просто: реакту потрібна унікальність, щоб всі його механізми працювали коректно. За «ключем» він буде розуміти, з яким саме дочірнім вузлом зараз хтось працює і якому батькові чи матері він належить.

В шаблоні використовувались значення змінних + текст, наприклад, <p className = "news__author"> {item.author}: </ p> могло б бути подано в нативному js-коді як <p className = "news__author"> " + item.author + ':' </ p> (порожній рядок + значення змінної + двокрапка).

В результаті роботи функції map було отримано новий масив, що складається з необхідних нам реакт-елементів. Це і є розв’язок задачі, залишилося лише зберегти цей масив у змінній, наприклад newsTemplate, і в render функції компонента <News /> повернути розмітку + "змінну-шаблон".

js/app.js

```
...
var News =React.createClass({
render:function(){
var data =this.props.data;
varnewsTemplate=data.map(function(item, index){
return(
<div key={index}>
<p className="news__author">{item.author}:</p>
<p className="news__text">{item.text}</p>
</div>
)
})

return(
<div className="news">
{newsTemplate}
</div>
);
}
});
...
```

Результат виглядає так:

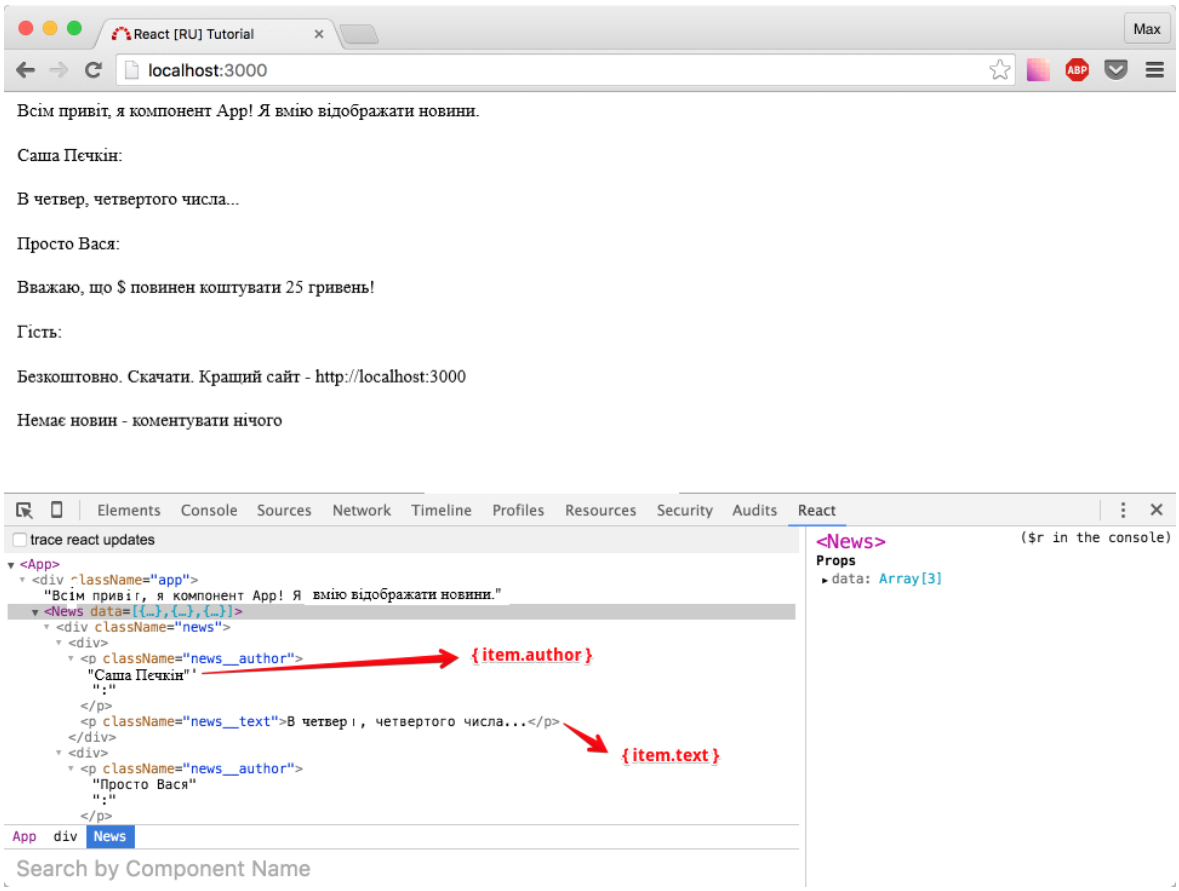

Рисунок 5.6 – Результат програми

Додайте console.log перед return'ом методу render компонента <News />.

js/app.js

```
var News =React.createClass({
render:function(){
var data =this.props.data;
varnewsTemplate=data.map(function(item, index){
return(
<div key={index}>
<p className="news__author">{item.author}:</p>
<p className="news__text">{item.text}</p>
</div>
)
})
console.log(newsTemplate);
return(
<div className="news">
{newsTemplate}
</div>
);
}
});
```

Подивимось у консоль:

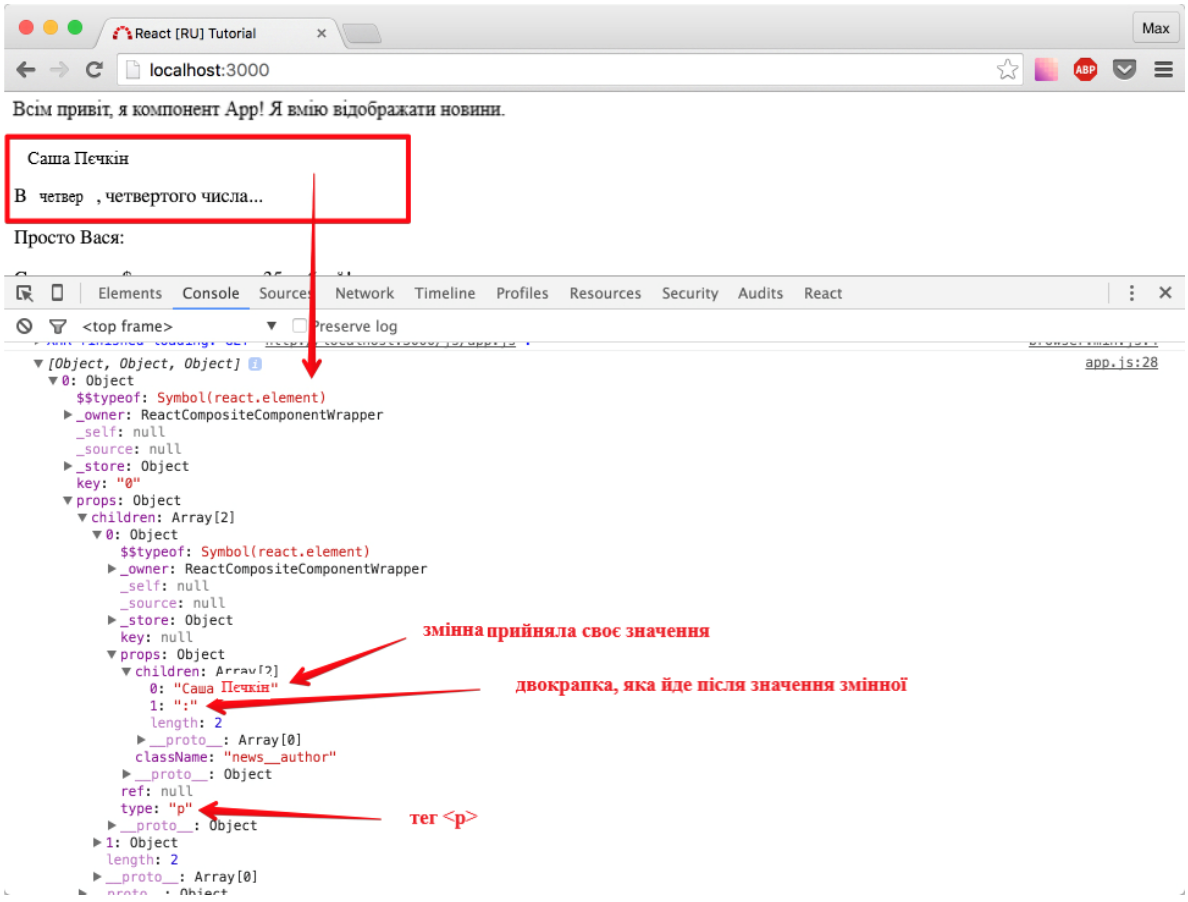

Рисунок 5.7 – Консоль програми

Об’єкт, у об’єкта властивості, все як вимагає javascript.

У підсумку ми навчилися відображати властивості компонента. Тепер необхідно видалити console.log.

В даному завданні для відображення масиву новин використовується key = {index}. Зверніть увагу на таку гілку коментарів – https://habrahabr.ru/post/279249/#comment_8807873

Контрольні запитання

1. Яка різниця між Елементом і Компонентом в React?
2. Що відбувається, коли Ви викликаєте setState?
3. Коли необхідно використовувати Class Component замість Functional Component?
4. Що таке refs в React і в чому їх особливість?
5. Що таке keys в React і в чому їх особливість?
6. В чому різниця між controlled і uncontrolled компонентами?
7. Що робить і для чого потрібен shouldComponentUpdate?
8. Опишіть, як в React обробляються події?
9. В чому різниця між createElement і cloneElement?

Лабораторна робота № 6

Тема: Створення Vue.js компонента.

Мета: створити систему оцінювання за кількістю зірок, використовуючи концепції Vue.js.

Теоретичні відомості

Vue (вимовляється / vju: /, приблизно як view) - це прогресивний фреймворк для створення користувацьких інтерфейсів. На відміну від фреймворків-монолітів, Vue створений придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня уявлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторінкових додатків (SPA, Single-Page Applications), якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками.[7]

Хід роботи

Vue.js має можливість запуску за допомогою звичайного додавання js скрипту, але ситуація кардинально змінилась з однофайловими компонентами. Тепер не потрібно створювати компоненти у такий спосіб. Можна обійтись глобальним компонентом з Vue.component. Проблема у тому, що з'являються побічні ефекти: необхідність використання шаблонів рядків, відсутність підтримки локального CSS, а також етапу збирання. Проте необхідно зануритись і навчитись створювати справжній компонент, який можна використовувати в реальному проекті. З цих причин ми будемо працювати з реальними налаштуваннями на базі Webpack. [10]

Використаємо шаблон vue-cli та Vue.js, а також шаблон webpack-simple.

Для початку потрібно встановити vue-cli глобально. Потрібно запустити консоль і ввести

npm install -g vue-**cli.**

Тепер можна створювати готові до використання Vue.js шаблони у кілька кліків. Далі потрібно ввести

vue init webpack-simple path/**to**/**my**-project.

Вам буде запропоновано декілька варіантів. Виберіть для всіх параметри за замовчуванням, за винятком «Use sass», на який слід відповідати yes (y). Тоді vue-cli ініціалізує проект і створить файл package.json. Коли це буде зроблено, ви зможете перейти до каталога, встановити залежності та запустити проект

```
cd path/to/my-project
npm install
npm run dev
```

Webpack почне показувати ваш проект на порту 8080 (якщо є) і запустить його у вашому браузері. Якщо все пройшло добре, ви маєте побачити привітання, подібне до цього.

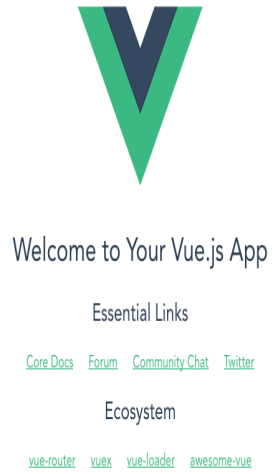

Рисунок 6.1 – Привітання

Щоб належним чином налаштувати компонент Vue.js, потрібні правильні інструменти. Потрібно перейти й встановити розширення браузера Vue.js Devtools (Firefox /Chrome/Safari).

Перший компонент

Однією з найкращих функцій Vue.js є однофайлові компоненти (SFC). Вони дозволяють визначати структуру, стиль та поведінку компонента в одному файлі без стандартних недоліків змішування HTML, CSS та JavaScript.

SFC мають розширення .vue і таку структуру:

```
<template>
<!-- Ваш HTML -->
</template>

<script>
  /* Ваш JS */
</script>

<style>
  /* Ваш CSS */
</style>
```

Створимо компонент:

Для цього створіть Rating.vue у /src/components та скопіюйте/вставте фрагмент коду вище. Відкрийте /src/main.js і адаптуйте наявний код

```
import Vue from 'vue'
import Rating from './components/Rating'

new Vue({
  el: '#app',
```

```
  template: '<Rating/>',
  components: { Rating }
})
```

Додайте трохи HTML у свій Rating.vue:

```
<template>
<ul>
<li>One</li>
<li>Two</li>
<li>Three</li>
</ul>
</template>
```

Тепер подивіться на сторінку у вашому браузері, там ви маєте побачити список. Vue.js додав ваш компонент <Rating> до елемента #app в index.html. Якщо ви перевіряєте HTML, ви не маєте бачити жодного знака елемента #app: Vue.js замінив його компонентом.

Webpack vue-loader постачається з функцією hot reload. На відміну від живого перезавантаження чи синхронізації браузера hot reload не оновлює сторінку щоразу, коли ви змінюєте файл. Замість цього він стежить за змінами компонентів і лише оновлює їх, не змінюючи стан.

Шаблон

Використовуємо vue-awesome, компонент SVG іконок для Vue.js, побудований за допомогою Font Awesome [8]. Це дозволяє завантажувати лише потрібні іконки. Встановіть його за допомогою npm (або Yarn)

```
npm install vue-awesome
```

Відредагуйте таким чином

```
<template>
<div>
<ul>
<li><icon name="star"/></li>
<li><icon name="star"/></li>
<li><icon name="star"/></li>
<li><icon name="star-o"/></li>
<li><icon name="star-o"/></li>
</ul>
<span>3 of 5</span>
</div>
</template>

<script>
import 'vue-awesome/icons/star'
import 'vue-awesome/icons/star-o'

import Icon from 'vue-awesome/components/Icon'

exportdefault {
   components: { Icon }
  }
</script>
```

Vue.js використовує нативні модулі ES6 для обробки залежностей та експорту компонентів. Перші два рядки блока script імпортують іконки самостійно, тому ви не отримаєте у остаточному пакеті іконок, які не потрібні. Третій імпортує Icon-компонент з vue-awesome, щоб була можливість використовувати його у створеному середовищі.

Icon – це Vue.js SFC. Якщо відкрити файл, то можна побачити, що він має точно таку ж структуру.

Export default блок експортує об'єкт як модель подання компонента. Компонент Icon був зареєстрований у властивостях components, щоб потім мати можливість його використовувати.

Після залучення компонента Icon у HTMLtemplate потрібно передати йому властивість name, щоб визначити, яка іконка буде реалізовуватись наступною. Компоненти можуть використовуватись як спеціальні теги HTML, перетворюючи їх у шаблонні набори (наприклад: MyComponent стає my-component). Не потрібно вставляти щось усередині компонента, тому використовуємо тег, що закривається самостійно.

Потрібно додати div, тому що ми приєднали лічильник до span на кореневому рівні, а шаблони компонентів у Vue.js приймають лише один кореневий елемент. за цим потрібно слідкувати, інакше можна отримати помилку під час компіляції.

Стиль

React має стилізовані компоненти (styled components), Vue.js має **компонентно-локальний CSS**. Це дозволяє писати компонентно-специфічний CSS. Ви пишете звичайний CSS з іменами «звичайних» класів, а Vue.js обробляє область визначення, присвоюючи атрибути даних елементам HTML і додаючи його до скомпонованих стилів.

Додамо кілька простих класів до компонента

```
<template>
<div class="rating">
<ul class="list">
<li class="star active"><icon name="star"/></li>
<li class="star active"><icon name="star"/></li>
<li class="star active"><icon name="star"/></li>
<li class="star"><icon name="star-o"/></li>
<li class="star"><icon name="star-o"/></li>
</ul>
<span>3 of 5</span>
</div>
</template>
```

і стилізуємо їх

```
<style scoped>
  .rating {
    font-family: 'Avenir', Helvetica, Arial, sans-serif;
    font-size: 14px;
```

```
  color: #a7a8a8;
 }
 .list {
  margin: 0 0 5px 0;
  padding: 0;
  list-style-type: none;
 }
 .list:hover .star {
  color: #f3d23e;
 }
 .star {
  display: inline-block;
  cursor: pointer;
 }
 .star:hover ~ .star:not(.active) {
  color: inherit;
 }
 .active {
  color: #f3d23e;
 }
</style>.
```

Тут відсутні локальні атрибути, ось що робить Vue.js задля масштабованості стилів. Якщо ви копіюєте/вставляєте HTML-код прямо в index.html, ваші стилі не застосовуються: це пов'язано з тим, що вони орієнтовані на компонент.

Робота з препроцесорами

Vue.js робить простим перехід від звичайного CSS до препроцесора. Все, що вам потрібно, – правильний Webpack-завантажувач і простий атрибут в блоці style. Під час створення проекту було зазначено, що «yes» для «Use sass», тому для нас вже встановлено та налаштовано sass-cli. Тепер все, що потрібно зробити, це додати lang = "scss" до відкривного тегу style.

Далі Sass можна використовувати, щоб писати стилі компонентів, імпортувати змінні, визначати кольори або міксини. Якщо хтось віддає перевагу синтаксису (або «sass»), можна просто змінити scss на sass в атрибуті lang.[9]

Поведінка

Тепер, коли компонент виглядає добре, настав час змусити його працювати. Зараз існує шаблон. Потрібно створити початковий макет і налаштувати шаблон таким чином, щоб він відображався

```
<script>
 ...
 export default {
  components: { Icon },
  data() {
```

```
      return {
        stars: 3,
        maxStars: 5
      }
    }
  }
</script>
<template>
<div class="rating">
<ul class="list">
<li v-for="star in maxStars" :class="{ 'active': star <= stars }" class="star">
<icon :name="star <= stars ? 'star' : 'star-o'"/>
</li>
</ul>
<span>3 of 5</span>
</div>
</template>
```

Vue data використовується для встановлення стану компонентів. Кожна властивість, яка визначається в data, стає реактивною: якщо вона змінюється, то це буде зображено у поданні.

Оскільки створюється багаторазовий компонент, то data має бути фабричною функцією, а не літералом об'єкта. Таким чином, отримуємо новий об'єкт, а не посилання на чинний, який буде розділений кількома компонентами.

Дана фабрика даних (factory data) повертає дві властивості: stars – поточна кількість «активних» зірок і maxStars – загальна кількість зірок для компонента. З цих причин шаблон потрібно адаптувати так, щоб він показував стан фактичного компонента. Vue.js поставляється з набором директив, які дозволяють додавати логіку презентації у шаблон, не змішуючи його з простим JavaScript. V-for проходять циклом над будь-яким ітерабельним об'єктом (масиви, об'єкти тощо). Він також може приймати номер як діапазон, який буде повторюватися *x* разів. Ось що ми зробили з v-for = "star in maxStars", тому маємо li для кожної зірки в компоненті.

Деякі властивості мають префікс двокрапкою: це скорочення для директиви v-bind, яка динамічно «прив'язує» атрибути до виразу. Це можна було б записати у довгій формі, v-bind: class.

Коли зірка активна, потрібно додати active клас до елементів li – це означає, що кожен li, чий індекс менший, ніж stars, має мати active клас.

Робота з лічильником

Тепер, коли список зірок пов'язаний з фактичними даними, настав час зробити те ж саме для лічильника. Найпростіший спосіб зробити це – використовувати текстову інтерполяцію з таким синтаксисом

```
<span>{{ stars }} of {{ maxStars }}</span>
```

Якщо потрібен більш складний вираз JavaScript, то краще абстрагувати його в обчислених властивостях.

```
export default {
 ...
 computed: {
  counter() {
   return `${this.stars} of ${this.maxStars}`
  }
 }
}
<span>{{ counter }}</span>
```

Ось це надмірність.

Інша річ, яку треба зробити, це забезпечити спосіб для приховання лічильника. Найпростіший шлях – використовувати директиву v-if з boolean.

```
<span v-if="hasCounter">{{ stars }} of {{ maxStars }}</span>
export default {
 ...
 data() {
  return {
   stars: 3,
   maxStars: 5,
   hasCounter: true
  }
 }
}
```

Інтерактивність

Використаємо v-on, директиву Vue.js, яка обробляє події та methods, властивість Vue.js, на яку ви можете приєднати всі свої методи.

```
<template>
 ...
<li @click="rate(star)" ...>
 ...
</template>
export default {
 ...
 methods: {
  rate(star) {
   // щось робиться
  }
 }
}
```

Далі потрібно додати атрибут @click до li, який є скороченням для v-on:click. Ця директива містить виклик rate-методу, який було визначено в methods-властивостях компонента.

Попри те, що синтаксис виглядає так само, як onclick, порівнювати їх буде помилкою. Створюючи компонент Vue.js, ви не маєте думати про

нього як про окремий HTML/CSS/JS, а як один компонент, який використовує декілька мов. Коли проект подається у браузері або компілюється для продакшену, всі HTML-документи та директиви складаються в простий JavaScript. Якщо ви перевіряєте зрендерований HTML, ви не побачите жодних ознак своїх директив, а також атрибутів onclick. Vue.js склав цей компонент і створив відповідні біндинги. Це також означає, що ви маєте доступ до контексту компонента прямо з шаблона: оскільки директиви пов'язані з моделлю перегляду. Всупереч традиційному проекту з окремим HTML, шаблон є невід'ємною частиною компонента.

Повернімося до \ rate методу. Тут потрібно змінювати stars до індексу натиснутого елемента, тому ми передаємо індекс з директиви @click таким чином:

```
export default {
  ...
  methods: {
    rate(star) {
      this.stars = star
    }
  }
}
```

Перевірте сторінку в своєму браузері та спробуйте натиснути зірки.

Якщо ви відкриєте панель Vue у браузерній консолі розробника і виберете компонент Rating, то можна побачити зміну даних, при натисканні на зірочки. Це означає, що властивість stars є реактивною: коли її змінють, вона відправляє свої зміни на перегляд. Дана концепція називається зв'язуванням даних (data-binding). Різниця полягає в тому, що Vue.js, як і React, робить це лише в одному напрямку, що називається одностороннім зв'язуванням даних.

Зараз не можна встановити нульову оцінку, тому що натискання на зірку встановлює значення до свого індексу. Краще повторно натиснути на одну і ту ж зірку і перемкнути її поточний стан, а не залишати активною.

```
export default {
  ...
  methods: {
    rate(star) {
      this.stars = this.stars === star ? star - 1 : star
    }
  }
}
```

Тепер, якщо індекс кліка зірки дорівнює поточному значенню stars, зменшуємо його значення. В іншому випадку потрібно призначити йому значення star.

Для більш ретельного підходу необхідно додати рівень контролю, щоб переконатися, що для stars ніколи не присвоюється безглузде значення. Потрібно переконатися, що stars ніколи не стає менша за 0, є не більшою, ніж maxStars, і є правильним числом.

```
export default {
  ...
  methods: {
    rate(star) {
if (typeof star === 'number' &&star<= this.maxStars &&star>= 0) {
        this.stars = this.stars === star ? star - 1 : star
      }
    }
  }
}
```

Передавання властивостей

На даний час інформація про компонент прихована в data-властивостях. Якщо необхідно, щоб компонент використовувався, ми маємо мати можливість передавати йому спеціальні дані з його екземплярів. У Vue.js це робиться за допомогою props

```
exportdefault {
  props: ['grade', 'maxStars', 'hasCounter'],
  data() {
    return {
      stars: this.grade
    }
  },
  ...
}
```

І в main.js:

```
new Vue({
  el: '#app',
  template: '<Rating :grade="3" :maxStars="5" :hasCounter="true"/>',
  components: { Rating }
})
```

Можна виділити три речі:

По-перше, ми використали v-bind для передачі властивостей з екземпляра компонента: це Vue.js називає динамічним синтаксисом. Це не потрібно, коли ви хочете передавати значення рядка, для якого буде працювати буквальний синтаксис (звичайний атрибут без v-bind). Але в цьому випадку, оскільки передаються числа та логічні вирази, це дуже важливо.

Props та data об’єднуються під час компіляції, тому не потрібно змінювати спосіб, яким називаються властивості як у моделі подання, так і у шаблоні. Але з цієї ж причини не можна використовувати однакові імена для props і data.

Ми визначили grade prop і передали як значення для stars у data-властивість: оскільки значення буде змінено, коли змінюється оцінка. У Vue.js, властивості передаються від предків до нащадків, а не навпаки. Це дозволить протистояти принципу потоку даних в одну сторону та ускладнити налагодження. Ось чому не потрібно змінювати підтримку всередині дочірнього компонента. Замість цього визначте локальну властивість data, яка використовує початкове значення властивостей як власне.

Завершення

Vue.js дозволяє контролювати властивості, перш ніж вони передаються у компонент. Ви можете виконати чотири основні речі: перевірити тип, вимагати визначення властивостей, встановити значення за замовчуванням та виконати користувальницьку перевірку.

```
export default {
  props: {
    grade: {
      type: Number,
      required: true
    },
    maxStars: {
      type: Number,
      default: 5
    },
    hasCounter: {
      type: Boolean,
      default: true
    }
  },
  ...
}
```

Ми використовували перевірку типу, щоб переконатися, що потрібний тип даних передається компоненту. Це буде особливо корисним, якщо ми забудемо використовувати динамічний синтаксис для передачі нестандартних значень. Також переконалися, що значення grade було передано. Для інших властивостей було визначено значення за замовчуванням, тому компонент працює, навіть якщо не передано жодних спеціальних даних.

Тепер потрібно проаналізувати компонент, виконавши нижченаведене.

```
<Rating :grade="3"/>
```

На цьому все, перший компонент Vue.js створено, а також досліджено багато концепцій, також генерацію шаблонного проекту з vue-cli, однофайлові компоненти, імпортування компонентів, локальні стилі,

директиви, обробники подій, обчислені властивості, спеціальні методи, односторонній потік даних, властивості та перевірка властивостей.

Контрольні запитання

1. На якому шаблоні проектування заснований Vue?
2. Назвіть хуки життєвого циклу компонента в Vue.js?
3. Як оновити стан компонента в Vue ?
4. Що таке обчислювальні властивості і коли їх потрібно використовувати?
5. Як підключити зовнішній css файл в Vue ?
6. Як підключити jQuery-плагін?
7. Як зареєструвати компонент в Vue.js ?
8. Що таке Vuex?
9. Як створити умовний рендерінг компонента?
10. Чи існує в Vue.js підтримка data binding? Якщо так, то як її використовувати?
11. Як реалізувати маршрутизацію на стороні клієнта в Vue?
12. Як програмно зробити редірект в Vue Router?

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. 30 CSS Best Practices for Beginners [Електронний ресурс]. – Режим доступу: https://code.tutsplus.com/tutorials/30-css-best-practices-for-beginners--net-6741 (дата звернення 26.03.2019) – Назва з екрана.

2. Methods to Organize CSS [Електронний ресурс]. – Режим доступу: https://css-tricks.com/methods-organize-css/ (дата звернення 26.03.2019) – Назва з екрана.

3. Хорошие и плохие CSS-практики для начинающих [Електронний ресурс]. – Режим доступу: https://medium.com/@ABatickaya/%D1%85%D0%BE%D1%80%D0%BE%D1%88%D0%B8%D0%B5-%D0%B8-%D0%BF%D0%BB%D0%BE%D1%85%D0%B8%D0%B5-css-%D0%BF%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B8-%D0%B4%D0%BB%D1%8F-%D0%BD%D0%B0%D1%87%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B8%D1%85-619289ce8bae (дата звернення 26.03.2019) – Назва з екрана.

4. Efficiently Rendering CSS [Електронний ресурс]. – Режим доступу: https://css-tricks.com/efficiently-rendering-css/ (дата звернення 26.03.2019) – Назва з екрана.

5. Методологія BEM [Електронний ресурс]. – Режим доступу: https://ru.bem.info/methodology/css/ (дата звернення 26.03.2019) – Назва з екрана.

6. React – A JavaScript library for building user interfaces [Електронний ресурс]. – Режим доступу: https://reactjs.org/ (дата звернення 26.03.2019) – Назва з екрана.

7. Vue.js [Електронний ресурс]. – Режим доступу: https://vuejs.org/ (дата звернення 26.03.2019) – Назва з екрана.

8. Рефакторинг.Гуру Шаблони проектування [Електронний ресурс]. – Режим доступу: https://refactoring.guru/uk (дата звернення 26.03.2019) – Назва з екрана.

9. Coder Projects [Електронний ресурс]. – Режим доступу: https://googlecreativelab.github.io/coder-projects/ (дата звернення 26.03.2019) – Назва з екрана.

10. W3Schools Online Web Tutorials [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/ (дата звернення 26.03.2019) – Назва з екрана.

Навчальне видання

Методичні вказівки
до виконання лабораторних робіт
з курсу «Web-технології» для студентів спеціальностей
124 – «Системний аналіз»,
126 – «Інформаційні системи та технології»,
151 – «Автоматизація та комп'ютерно-інтегровані технології»,
152 – «Метрологія та інформаційно-вимірювальна техніка»

Укладачі: *Барабан Марія Володимирівна*
Барабан Сергій Володимировмч
Бевз Олександр Миколайович

Рукопис оформлено *М. Барабан*

Редактор *В. Дружиніна*

Оригінал-макет підготовлено *О. Ткачуком*

Підписано до друку 18.09.2019 р.
Формат 29,7×42¼. Папір офсетний.
Гарнітура Times New Roman.
Друк різографічний. Ум. друк. арк. 4,02.
Наклад 40 (1-й запуск 1–20) пр. Зам. № 2019-122.

Видавець та виготовлювач
Вінницький національний технічний університет,
інформаційний редакційно-видавничий центр.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95,
м. Вінниця, 21021.
Тел. (0432) 65-18-06.
press.vntu.edu.ua;
E-mail: kivc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.