Лабораторна робота 1.

ТЕМА: Початкове налаштування web-проекту.

МЕТА: Опанувати практичними навичками початкового налаштування web-проекту з використанням таск-менеджера для автоматизації процесу web-розробки.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Gulp – це інструмент, який допомагає автоматизувати рутинні завдання веб-розробки. gulp призначений для вирішення таких завдань, як:

- Створення веб-сервера і автоматичне оновлення сторінки в браузері при збереженні коду, відслідковування змін в файлах проекту;
- Використання препроцесорів (Less, Sass, Stylus, CoffeeScript);
- Мініфікація оптимізація та конкатенація окремих файлів проекту в один;
- Автоматичне створення вендорних префіксів для CSS (префікси для нестандартних CSS властивостей, які додають виробники браузерів).
- Керування файлами і папками (створення, видалення, перейменування) в межах проекту;
- Запуск і контроль виконання зовнішніх команд операційної системи;
- Робота з зображеннями – створення спрайтів, стиснення, зміна розмірів;
- Робота з зовнішніми серверами по FTP, SFTP, Git.
- Використання в проекті Node.js, утиліт, програм і плагінів.
- Створення карт проекту і автоматизація ручної роботи.

Слід зазначити, що gulp і утиліти, створені для нього, забезпечують вирішення практично будь-якої задачі розробки web-проекту будь-якої складності. Незважаючи на широкий спектр застосування, gulp характеризується низьким порогом входження, тобто є простим в опануванні.

Будь-який gulp проект має в корені файл gulpfile.js, в якому описані інструкції для керування проектом. Особливістю переважної більшості інструкцій є їх однотипність і часто gulpfile.js може бути великих розмірів.

Встановлення gulp.

Для роботи з gulp вимагається наявність Node.js на комп'ютері. Процес встановлення Node.js для різних платформ досить простий: потрібно завантажити інсталятор для відповідної операційної системи і запустити його. Слід зазначити, що для Mac OS платформ Node.js встановлений за замовчуванням.

Після встановлення Node.js потрібно встановити gulp глобально. Для цього необхідно в командному рядку виконати наступну команду:

```
npm i gulp -g
```

Наведена вище команда запускає менеджер пакетів npm (Node Package Manager), який і встановлює gulp в операційну систему. Слід звернути увагу на ключ -g, який і вказує інсталятору виконати встановлення глобально.

Створення gulp проекта.

Спочатку потрібно створити директорію проекту, наприклад myProject. Далі відкрити термінал в папці проекту (myProject) і виконаємо ініціалізацію проекту, запустивши на виконання наступну команду:

```
npm init
```

Відповідно до інструкції, необхідно заповнити метадані про створений проект.

- package-name: назва проекту (необхідно назвати відповідно до рекомендацій у завданні);
- version: версія (можна залишити за замовчуванням 1.0.0);

- description: короткий опис проекту;
- entry point: (index.js), test command:, keywords: ці поля залишаємо за замовчуванням;
- git repository: потрібно вказати посилання на репозиторій проекту;
- author: ім’я автора проекту;
- license: залишити за замовчуванням;
- Is this ok? — потрібно ввести «yes» і натиснути Enter.

В результаті виконаних дій в папці створеного проекту згенерується файл package.json в якому буде розміщуватися вся інформація, яку було введено через термінал та про використані пакети, які будуть встановлюватися в проект. Наприклад, якщо встановити gulp проект з ключем -save-dev, то інформація про пакет (назва і версія) занесеться в package.json. Такий підхід дозволяє досить швидко розгортати проект на новому комп'ютері з використанням вже наявного package.json і встановлювати необхідні модулі з залежностями, які прописані в даному файлі.

Крім цього, згенерується папка node_modules, в якій розміщені встановлений gulp пакет і необхідні залежності. До цієї папки автоматично будуть додаватися всі модулі і залежності, які будуть встановлюватися в проект. Папок з залежностями може бути досить багато. Це пов’язано з тим, що на додаток до основних пакетів встановлюються програми, необхідні для коректної роботи основного пакета.

Формування структури каталогів gulp проекта.

Для продовження роботи над проектом необхідно задати структуру каталогів за всіма правилами хорошого тону web-розробки. В результаті повинна бути створена наступна структура в myProject (всі файли, які не згенерувалися, потрібно створити порожніми):

- app/
  - css/
  - fonts/
  - img/
  - js/
  - sass/
  - index.html
- dist/
- node_modules/
- gulpfile.js
- package.json

Розглянемо детальніше призначення кожної з директорій.

app/ – в даній директорії будуть зберігатися вихідні файли проекту. Взагалі, в дочірніх папках може бути досить багато файлів, в залежності від розміру проекту. В подальшому файли об'єднуються в один загальний файл відповідно до типу і копіюються в папку dist. У корені лежить файл index.html. Для HTML файлів можна створити окремий каталог, але в межах лабораторних робіт достатньо використання лише головного файлу.

css/ – директорія з файлами css-стилів.

fonts/ – директорія з файлами шрифтів.

img/ – неоптимізовані зображення та іконки проекту.

js/ – javaScript файли.

sass/ – файли препроцесорів SASS або SCSS.

dist/ – дана папка використовується для зберігання всіх готових файлів після компіляції. Внутрішня структура папки dist буде генеруватися автоматично при компіляції, тому задавати її структуру немає змісту.

node_modules/ – розміщуються всі модулі і gulp залежності.

gulpfile.js – це головний файл gulp проекту (розглянемо його далі).

package.json – опис даного файлу розглядався вище.

Представлена структура зустрічається досить часто, практично у всіх проектах, але це не аксіома і деякі проекти можуть мати взагалі іншу структуру.

gulpfile.js.

Це, по суті, головний файл без якого gulp проект працювати не буде. В цьому файлі записуються всі завдання для виконання (так звані таски (task)). Завдання створюються для забезпечення роботи додаткових плагінів. Без таска плагіни працювати не будуть. Розглянемо приклади створення тасків.

Відкрийте gulpfile.js в довільному текстовому редакторі і вставте в нього наступний код:

```
//Підключаємо gulp
var gulp = require ("gulp");

//Створюємо тестовий таск
gulp.task ('testTask', function () {
  console.log ('This is a test task!');
});

//Запуск тасків за замовчуванням
gulp.task ("default", ["testTask"]);
```

Розглянемо детальніше даний код.

```
var gulp = require ("gulp");
```

В цьому рядку підключається gulp, тобто створюється об'єкт з даними, з яким будемо працювати. Аналогічно підключаються всі додаткові gulp плагіни.

```
gulp.task ('testTask', function () {
  console.log ('This is a test task!');
});
```

Даний фрагмент gulpfile.js описує конкретний таск, де testTask – назва таску, а в функції представляється відповідна логіка (в даному випадку вивід стрічки в консоль).

```
gulp.task ("default", ["testTask"]);
```

Цей фрагмент дозволяє запустити за замовчуванням всі прописані в квадратних дужках таски. В даному випадку вказаний тільки один testTask. На виконання можна передавати декілька тасків у вигляді масиву назв. Наприклад, ["testTask1", "testTask2", "testTask3"]. Це дозволяє однією командою запускати на виконання відразу декілька тасків.

Для запуску на виконання створених тасків потрібно в консолі ввести команду gulp і натиснути кнопку Enter. В результаті в консолі виведеться повідомлення: This is a test task. В цьому випадку запустяться на виконання всі таски, які потрібно виконати за замовчуванням.

Для виконання конкретного таска необхідно після команди gulp ввести його назву: gulp testTask.

Приклади встановлення та використання gulp плагінів.

Розглянемо створення невеликого проекту для верстки сторінок. Спочатку необхідно встановити декілька плагінів, які спростять роботу. Для цього gulpfile.js потрібно доповнити наступним кодом (підключимо необхідні плагіни):

```
var gulp = require ("gulp");

//додаткові плагіни Gulp
var sass = require ("gulp-sass"), //конвертує SASS в CSS
    cssnano = require ("gulp-cssnano"), //мінімізація CSS
    autoprefixer = require ('gulp-autoprefixer'), //додавання префіксів в
                                                  //CSS для підтримки
                                                  //старих браузерів
    imagemin = require ('gulp-imagemin'), //стиснення зображень
```

```
    concat = require ("gulp-concat"), //об'єднання файлів - конкатенація
    uglify = require ("gulp-uglify"), //мінімізація javascript
    rename = require ("gulp-rename"); //перейменування файлів
```

Доданий код це, по суті, оголошення змінних. Але для повноцінного використання оголошених плагінів потрібно їх встановити локально, у папку проекту. Приклад команди для встановлення відповідного плагіну наведено нижче:

```
npm i pluginName --save-dev
```

При потребі можна встановити декілька плагінів одночасно. Для цього потрібно в команді перелічити їх назви, використовуючи пропуск, як розділювач.

```
npm i pluginName1 pluginName2 pluginName3 pluginName4 --save-dev
```

Після завершення встановлення файл package.json автоматично доповниться інформацією про плагіни. Їх можна знайти у секції devDependencies.

Після встановлення плагінів потрібно написати відповідні таски. Далі наведено декілька прикладів.

```
//копіювання HTML файлів в папку dist
gulp.task ( "html", function () {
    return gulp.src ( "src / *. html")
    .pipe (gulp.dest ( "dist"));
});

//об'єднання, компіляція Sass в CSS, додавання префіксів і подальша
мінімізація коду
gulp.task ( "sass", function () {
    return gulp.src ( "src / sass / *. sass")
        .pipe (concat ( 'styles.sass'))
        .pipe (sass ())
        .pipe (autoprefixer ({
            browsers: [ 'last 2 versions'],
            cascade: false
         }))
        .pipe (cssnano ())
        .pipe (rename ({suffix: '.min'}))
        .pipe (gulp.dest ( "dist / css"));
});

//об'єднання і стиснення JS-файлів
gulp.task ( "scripts", function () {
    return gulp.src ( "src / js / *. js") //вихідна директорія файлів
        .pipe (concat ( 'scripts.js')) // конкатенація js-файлів в один
        .pipe (uglify ()) //стиснення коду
        .pipe (rename ({suffix: '.min'})) //перейменування файлу з
                                          //приставкою .min
        .pipe (gulp.dest ( "dist / js")); // директорія продакшена
});

//стискання зображень
gulp.task ( 'imgs', function () {
    return gulp.src ( "src / images /*.+ (jpg | jpeg | png | gif)")
        .pipe (imagemin ({
            progressive: true,
            svgoPlugins: [{removeViewBox: false}],
            interlaced: true
        }))
        .pipe (gulp.dest ( "dist / images"))
});

//відстежування за змінами у файлах
gulp.task ( "watch", function () {
```

```
    gulp.watch ( "src / *. html", [ "html"]);
    gulp.watch ( "src / js / *. js", [ "scripts"]);
    gulp.watch ( "src / sass / *. sass", [ "sass"]);
    gulp.watch ( "src / images /*.+ (jpg | jpeg | png | gif)", [ "imgs"]);
});

//Запуск тасків за замовчуванням
gulp.task ( "default", [ "html", "sass", "scripts", "imgs", "watch"]);
```

Слід звернути увагу на таск з назвою watch. Для нього не потрібно встановлювати додаткових плагінів, оскільки це вбудована функція gulp. Даний таск дозволяє стежити за файлами, в яких були зроблені зміни. Як тільки в певних файлах відбулися зміни, тобто збереження змін, запускається вказаний таск.

Розглянемо складові частини таску.

```
gulp.task('taskname', function () {
    return gulp.src('source-files')
    .pipe(plugin-name())
    .pipe(gulp.dest('folder'))
});
```

taskname – назва таска;

source-files – директорія вихідних файлів з програмним кодом, необхідно вказувати повний шлях і розширення файлів, наприклад, шлях до всіх фалів з розширенням .js буде таким: templatename / js / *. js;

.pipe () – метод, який дозволяє працювати з встановленим плагіном;

plugin-name – назва плагіна;

dest(«filder-path») – шлях до папки продакшена, тобто директорія вивантаження результуючих файлів після відпрацювання плагіна.

Хотілося б відзначити, що після внесення змін в gulpfile.js необхідно перезапускати команду gulp, попередньо зупинивши поточне її виконання, для того, щоб зміни вступили в силу. Для зупинки просто два рази поспіль вводимо комбінацію клавіш ctrl+C, коли побачимо миготливий курсор, знову вводимо - gulp.

Розгортання готової збірки однією командою.

Після створення збірки з необхідними плагінами можна її використовувати як базу для довільного web-проекту. Для її використання достатньо скопіювати її в потрібне місце і запустити встановлення. При встановленні модулів варто встановлювати їх останні версії, включаючи сам gulp (при оновленні глобальної версії). Для цього необхідно відредагувати файл package.json. У блоці devDependencies, навпроти кожного плагіна, замінити назву його версії на ключове слово – latest. Після таких змін завжди будуть встановлюватися останні версії модулів.

Тепер для швидкого розгортання проекту необхідно зробити наступне: скопіювати всі файли проекту Gulp за винятком папки node_modules в іншу папку проекту; відкрити консоль в новій директорії проекту і запустити на виконання команду "npm i". Після завершення встановлення gulp і всіх залежностей можна приступати безпосередньо до розробки web-проекту.

Додаткові корисні gulp плагіни.

Створений тестовий проект включає найпростіший набір плагінів gulp. Його функціональність можна значно розширити, встановивши і налаштувавши додаткові плагіни. Далі наведено перелік найбільш поширених серед web-розробників плагінів.

Plumber – дає можливість продовжити роботу gulp проекту при помилку в коді. Якщо в коді (html, js, sass, css) буде помилка, то компілятор gulp виведе її в консоль і припинить роботу. Для подальшої роботи необхідно буде виправити помилку і знову запустити gulp.

Sourcemaps – створює карту підключень файлів css і js. Зазвичай в подібних проектах файли стилів і скриптів ділять на кілька частин для зручності. Для того, щоб потім орієнтуватися в якому файлі і в якому рядку знаходиться певний запис якраз і використовується карта підключень.

Tinypng – стиснення зображень. Працює аналогічно до imagemin, але стискає значно краще.

SvgSprite – збірка svg-спрайту іконок. Даний плагін варто використовувати для зменшення кількості http запитів до сервера шляхом групування іконок в спрайт.

Rigger – об'єднує html-файли в один. Необхідно, коли html-файли розділені на окремі незмінні блоки, наприклад, шапка сайту, футер і т.д.

BrowserSync – перезавантаження сторінки у браузері. Дуже корисне доповнення, тому що не потрібно витрачати час на оновлення браузера, плагін робить це автоматично при збереженні змінених файлів. В плагіні використовується вбудований простий локальний сервер.

Spritesmith – створення спрайтів картинок. Ще один плагін для збірки спрайтів, тільки в даному випадку іконок з розширенням png.

Rimraf – очищення папки dist. Буває, що доводиться очищати час від часу папку продакшена dist, тому що в ньому можуть зібратися невикористовувані файли. Наприклад, після перейменування файлу стилів в іншу назву, таким чином у папці dist будуть дві копії – стара і нова.

Контрольні запитання

1. Що таке Gulp?
2. Чи вимагається наявність на комп’ютері Node.js?
3. Яка структура папок у web-проекті є заданою за правилами хорошого тону web-розробки?
4. Охарактеризувати функціональне призначення файлу package.json?
5. Охарактеризувати функціональне призначення файлу gulpfile.js?
6. Описати поняття "gulp task".
7. Охарактеризувати складові частини gulp таска.
8. Запуск на виконання окремого gulp таска.
9. Розгортання готової gulp-збірки з використанням однієї команди.
10. Перелічити додаткові gulp плагіни та їх функціональне призначення.

ЗАВДАННЯ

Виконати початкове налаштування web-проекту з використанням Gulp-менеджера. Задати структуру каталогів web-проекту. Створити gulp-таски для збірки web-проекту, відслідковування змін у файлах (gulp watch) та автоматичного оновлення сторінок з використанням розширення Browser Sync. Розмістити вихідний код web-проекту на віддаленому репозиторії.

Шаблон назви проекту: pm_groupNumber_variantNumber (наприклад: pm_22_05).

Лабораторна робота 2.

ТЕМА: Формування базової розмітки web-сторінки.

МЕТА: Ознайомитися з принципами побудови, структурою та формуванням розмітки сторінки при розробці web-сайту з використанням відповідного фреймворку.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Будь-який фронтендер або верстальник рано чи пізно приходить до того, щоб створити свій невеличкий фреймворк. Складається він зазвичай з тих правил і функцій, які доводиться повторювати в кожному проекті. Зібравши їх один раз в одній бібліотеці, починаючи роботу над наступним проектом, досить буде просто підключити її і використовувати готові рішення. Це може бути сітка для колонок з контентом, стандартні правила спрайтів, відступи, заголовки і т.д.

У разі, коли над одним проектом працюють кілька фронтенд-фахівців, подібні фреймворки повинні бути стандартизовані. І перевага в такому випадку віддається загальноприйнятим фреймворками. Тут ми опиняємося перед вибором: який фреймворк використовувати?

Bootstrap

Створено в закутках компанії Twitter, спочатку використовувався для власних продуктів і називався "Twitter Bootstrap", а пізніше був випущений на волю. За це у нього забрали слово Twitter з назви.

Bootstrap – це CSS / HTML фреймворк для створення сайтів. Іншими словами, це набір інструментів для верстки. У ньому є ряд переваг, завдяки яким BS вважається найпопулярнішим з собі подібних. Переваги Bootstrap:

Швидкість роботи – завдяки безлічі готових елементів верстка з бутстрапа займає значно менше часу;

Масштабованість – додавання нових елементів не порушує загальну структуру;

Легкість налаштування – редагування стилів проводиться шляхом створення нових css-правил, які виконуються замість стандартних. При цьому не потрібно використовувати атрибути типу "! Important";

Велика кількість шаблонів;

Величезне співтовариство розробників;

Широка сфера застосування – Bootstrap використовується в створенні тем для практично будь-яких CMS (OpenCart, Prestashop, Magento, Joomla, Bitrix, WordPress і будь-які інші), в тому числі для односторінкових додатків.

Особливою популярністю користується Bootstrap для створення односторінкових сайтів або "Лендінгів" (landing page).

Шаблони в Bootstrap дозволяють змінювати вже змінені елементи під свої потреби. Безліч розробників пропонують свої шаблони (як платно, так і безкоштовно).

Підключити шаблон в Bootstrap дуже легко: після підключення Bootstrap потрібно тільки додати виклик CSS вашого шаблону.

Вибираючи Bootstrap, можна істотно заощадити час на верстку і розробку фронтенда завдяки безлічі компонентів. Далі буде описано основні з них, які потрібні більшості фронтенд-розробників.

Потрібно зауважити, що Bootstrap – це скоріше набір з трьох фреймворків: css / html, js-компонентів і шрифтів.

Блокова система в Bootstrap – основа грамотної верстки, потужний інструмент для блочного каркасу блоків контенту і будь-яких вкладених елементів. За допомогою префіксів ми можемо вказувати, як потрібно змінювати відображення блоків в залежності від типу пристрою, на якому відображається сайт.

Наприклад, клас .col-xs- буде використаний для телефонів з шириною екрану менше 768 пікселів, а .col-lg- - для будь-яких пристроїв з екраном ширшим 1170 px. BS розбиває ширину батьківського блоку на 12 рівних частин, які ми можемо використовувати як завгодно. Частини можна об'єднувати, отримуючи, наприклад, три колонки: дві по 25% .col-lg-3 і одну на 50% .col-lg-6.

Крім оформлення блоків і структури сторінки, Bootstrap дозволяє оформити текст: абзаци, цитати, заголовки, підзаголовки, різні розміри тексту, вставки коду і так далі. У більшості випадків вам не доведеться міняти встановлені за замовчуванням параметри тексту, адже всі відступи, заголовки, між рядками відстані та інше прописані з точністю як в аптеці.

Чимало уваги приділено і семантиці: основний заголовок можна оформити тегом <h1> Заголовок </ h1>, а можна і Заголовок </ div> - виглядати обидва варіанти будуть однаково, зате другий можна використовувати скільки завгодно раз на сторінці.

Якщо ви хочете спробувати верстати на Bootstrap, але не хочете копатися в чомусь складному, почніть з малого: оформите текстову сторінку з будь-якої новиною. Розмежуйте заголовок, другорядні заголовки, вставте зображення, кілька цитат, списків і оціните, як легко цей фреймворк впорається з такими завданнями.

Чи потрібно користуватися Bootstrap, кожен вирішує сам, але будь-який поважаючий себе фронтенд-фахівець повинен вміти з ним працювати. Сфера застосування цього фреймворка дуже широка і він може стати в нагоді в будь-якій ситуації.

Посилання на офіційну документацію: http://getbootstrap.com/

Semantic UI

Якщо при розробці web-сайту потрібно створити складний прототип з великою кількістю нестандартних елементів управління, то одним з хороших рішень буде Semantic UI, який містить більше 3000 семантичних CSS класів, які легко застосовувати і призначені для оптимізації процесу розробки.

Кількість готових UI елементів з коробки просто вражає. Тут не тільки різні варіації стандартних елементів управління, але і такі речі як: зверстані картки профілів, заготовки для відображення зображень, різні види меню, різні візуальні ефекти та багато іншого.

Слід зазначити одну, досить важливу рису Semantic UI – можливість модифікації. Передбачається, що користувач (розробник web-сайту) легко можете змінювати кольорову гаму оформлення, створювати власні теми оформлення не ламаючи вихідний код фреймворка. До того ж, розробники даного ресурсу подбали про інтеграцію з популярними JS-фреймворками. Semantic UI прекрасно поєднується з React, Meteor, Ember і в тестовому режимі приймає з Angular.

Ще однією особливістю цього фреймворку є його налаштування. Наприклад, для виведення налагоджувальних повідомлень (для JS компонент) потрібно додати до методу єдину властивість – debug зі значенням true. Після цього відповідна інформація буде автоматично виводитися в консоль в легкому для читання форматі.

Semantic UI пропонує більше 20 різних моделей дизайну в стандартній версії. В цілому даний фреймворк є складнішим за структурою ніж Bootstrap. Крім основних файлів CSS і JavaScript основний пакет також має безліч шрифтів, менеджер пакетів PHP Composer, менеджер пакетів Bower, а також Gulp.

Також слід відзначити, що для новачків і користувачів, які приходять з інших систем, вхідний поріг в Semantic UI є вищим ніж в інші фреймворки і вимагає більше часу для освоєння. Зрештою, інвестування часу у вивчення Semantic UI окупитися, тому що HTML web-інтерфейсу буде більш інтуїтивно зрозумілий, ніж у випадку інших платформ.

Крім класичного CSS, Semantic UI також доступний у варіанті Less та Sass. З іншої сторони недоліком є те, що багато компонентів фреймворку залежать від JavaScript і не можуть працювати без мови сценаріїв.

Основний фокус Semantic UI полягає у тому, що практично все є «доступним з коробки». Якщо перед розробником web-ресурсу ставиться завдання притримуватися цього підходу, чи він є таким прихильником, то саме цей фреймворк можна брати у розробку.

Посилання на офіційну документацію: http://semantic-ui.com/

Material Design.

Сьогодні інтерфейси Google додатків виглядають і функціонують приблизно однаково. Але так було не завжди. Ще десять років тому додатки для Android, десктопний і мобільний вигляд пошти відрізнялися між собою.

У 2011 році в Google вирішили змінити концепцію і уніфікували свої продукти, створили єдиний стиль для додатків, але вони знову виявилися різними. В результаті користувачі і далі губилися при перемиканні між мобільним і десктопних інтерфейсами: виглядали вони по-різному, керувалися також.

До 2014 року проблему вдалося вирішити. Саме тоді на конференції I/O Google представили свою нову дизайн-систему Material Design. Компанія не просто представила посібник по візуальній стилізації, але і заявила про себе як про єдине цифрове середовище.

В основі Material Design лежать чотири принципи:

1. Тактильні поверхні.

Всі елементи інтерфейсу – це свого роду шари цифрового паперу, які розташовуються на різній висоті і відкидають тіні. Це допомагає користувачам відрізнити головні елементи від другорядних і робить інтерфейс інтуїтивно зрозумілим.

2. Поліграфічний дизайн.

Логічно, що на цифровому папері потрібно писати цифровим чорнилом. Все, що зображено і написано на шарах-елементах, підпорядковується законам друкованого дизайну. Так можна акцентувати увагу користувача на потрібному елементі і позначити ієрархію інтерфейсу.

3. Усвідомлена анімація.

Всі елементи, які є на екрані, не можуть просто так з’являтися і зникати, адже в реальному житті так не буває. Об’єкти плавно переходять один в інший і підказують користувачеві, як працює інтерфейс.

4. Адаптивний дизайн.

Все вище перераховане повинно працювати на будь-яких пристроях.

Слід зазначити, що анімація є однією з основ Material Design. І хоча деякі її критикують, шанувальників все ж більше. І ось чому.

Анімація в Material Design.

На відміну від Apple, у яких анімація несе переважно естетичну функцію, Google робить ставку на UX і функціональність. В їх рекомендаціях анімації приділено набагато більше уваги, а на конференціях раз у раз їм присвячуються доповіді.

Основна ідея анімації в Material Design – зробити користувальницький інтерфейс виразним і простим у використанні. Для цього вона повинна відповідати трьом принципам.

Інформативність.

Анімація показує просторові та ієрархічні зв’язки між елементами: які дії доступні користувачеві і що станеться, якщо він виконає одну з них.

Орієнтованість

Анімація фокусує увагу на тому, що важливо, і не відволікає від основного дії.

Виразність

Анімація висловлює характер, індивідуальність і стиль кожного продукту.

Таким чином, за допомогою анімації можна:

1. Визначити ієрархію.

Показати користувачеві, як елементи пов'язані один з одним.

2. Вчити користувача.

Показати, як виконувати різні дії.

3. Зробити вау-ефект.

Додати привабливості, щоб користувач знову захотів взаємодіяти з продуктом.

І це тільки верхівка айсберга. Google дійсно змусив світ переглянути ставлення до анімації і зробив її повноцінною частиною UX-дизайну. Можна шукати недоліки в рекомендаціях Material Design, але, не варто зовсім ігнорувати значення анімації сьогодні.

А от цікавий зауваження про одне з положень Material Design – про те, що всі предмети, що виходять з екрану, повинні прискорюватися. Провідний розробник Джон Шлеммер вважає, що неважливо, де саме вони зупиняться.

Завдяки Material Design анімація сьогодні – не просто ефектне доповнення дизайну, а повноцінна його частина.

Крім того, Material Design використовує принципи поліграфічного дизайну для ефектної розстановки акцентів (тобто фокусування уваги на потрібному елементі), спрощення навігації в інтерфейсі, інтуїтивної передачі сенсу його елементів. Для Material Design характерні насичені, рівні кольори, різкі, окреслені краї, велика типографіка і чималі відступи між елементами. У сукупності ці елементи не просто складаються в приємну для очей картинку, а створюють нову реальність з концептуальним змістом і безліччю функцій, які дарують користувачеві унікальний UX.

У випадку з цим видом дизайну дії користувача знаходяться в центрі уваги. Всі взаємодії відбуваються в одному оточенні, інтерактивні об'єкти без переривання послідовності переходять з однієї середу в іншу.

Material Design – це історія про функціональність, до якої прагне кожен елемент. Згідно з цим принципом, важливо зосередитися на основних точках уваги користувача, щоб направити його в потрібному напрямку. Концепція Material Design є більше, ніж просто елегантний користувальницький інтерфейс. Це передові технології, які здатні максимально спростити життя користувача і заощадити його час.

Посилання на офіційну документацію: https://material.io/develop/web

Контрольні запитання

1. Які особливості використання фреймворку Bootstrap?
2. Яка сфера застосування фреймворку Bootstrap?
3. Охарактеризувати складові компоненти фреймворку Bootstrap.
4. Охарактеризувати блокову структуру фреймворку Bootstrap.
5. Які особливості використання фреймворку Semantic UI?
6. На чому сфокусовано основну концепцію фреймворку Semantic UI?
7. Які компоненти включає базовий пакет Semantic UI?
8. В чому полягає основна концепція Material Design?
9. Означити основні принципи Material Design.
10. В чому полягає основна ідея анімації в Material Design?
11. Яким чином в Material Design реалізовано принципи поліграфічного дизайну?

ЗАВДАННЯ

Відповідно до варіанту завдання (заданого макету сторінки та фреймворку), розробити базову розмітку web-сторінки. При дослідженні особливостей використання фреймворку використовувати офіційну документацію до нього. Необхідно передбачити адаптивність верстки для desktop-екранів.

Лабораторна робота 3.

ТЕМА: Використання css препроцесорів при розробці web-сторінки.

МЕТА: Ознайомитися з основними css препроцесорами та їх використанням при розробці web-сайту.

ТЕОРЕТИЧНІ ВІДОМОСТІ

CSS-препроцесор (CSS preprocessor) – це програма з власним синтаксисом і може генерувати CSS код. Існує безліч препроцесорів. Більшість з них розширює функціонал чистого CSS, додаючи такі опції як: домішки, вкладені правила, селектори спадкування та ін. Ці особливості полегшують роботу з CSS: спрощують читання коду і його подальшу підтримку.

В загальному розумінні препроцесор є певною надбудовою над стандартним CSS, яка розширює стандартні можливості, додає деякі функції і дозволяє писати більш придатний для читання і зручний для розуміння код. В результаті компіляції на виході отримуємо звичний CSS. Використання подібного інструменту в сучасній розробці є досить важливим і часто використовується у промисловій розробці. Тому для майбутніх спеціалістів варто опанувати подібний інструментарій і набути практичний досвід з його використання.

На даний момент найбільш популярними CSS-препроцесорами є:

- Sass (SCSS);
- Less;
- Stylus.

Крім перелічених є й інші, але вони не користуються популярністю в розробників. Далі розглянемо особливості перелічених вище CSS-препроцесорів.

Як я вже зазначив вище, основні плюси - це читабельність коду, структурування і підвищення продуктивності.

Читабельність коду, його структурування і підвищення продуктивності розробки безперечно є позитивною стороною використання CSS-препроцесорів. Хоча для цього потрібно опанувати їх синтаксис, який може видатися складним і незрозумілим. Тому значна частина розробників вважає їх непотрібними. Але на це потрібно подивитися з іншої сторони.

Стандартний підхід з використанням CSS є доволі складним. Синтаксис без використання вкладеності, яка присутня у CSS-препроцесорів, є складним для зорового сприйняття. Крім того, потрібно пам’ятати ім’я батьківського селектора при вкладеності. Відсутність можливості використовувати змінні та функції перетворює класичний CSS-код в набір селекторів, які важко підтримувати чи модифікувати.

Практично для кожного популярного CSS-препроцесора, зокрема і для представлених вище, доступна актуальна документація. З її використанням в досить стислі терміни і з мінімальними витратами сил можна опанувати синтаксисом і практично відразу його застосовувати у web-розробці.

Використовувати препроцесори стало простіше, ніж раніше. Для цього потрібно лише встановити програму, яка буде стежити за файлами, призначеними для препроцесора, і при їх зміні буде компілювати вміст цих файлів в чистий CSS-код. Зокрема, таку можливість надають спеціальні збирачі проектів, які пропонують повний контроль над усіма файлами проекту, а також додаткові параметри, які є корисними під час розробки.

Найпопулярнішим і досить зручним функціоналом будь-якого CSS-препроцесора є можливість вкладати селектори один в одного. Узагальнену модель вкладеності наведено нижче.

```
1. Батьківський селектор
    1.1. Дочірній (вкладений) селектор
    1.2. Дочірній (вкладений) селектор
        1.2.1. Дочірній (вкладений) селектор
    1.3. Дочірній (вкладений) селектор
```

Практично в кожному CSS-препроцесорі використовується таке поняття, як домішки (анг. mixins), що дозволяє перевикористовувати певний блок коду в потрібних місцях. Домішки це свого роду функції, в яких визначаються певні CSS властивості. Це дозволяє суттєво скоротити кількість коду, який дублюється в різних частинах CSS файла. Коли домішка викликається в певному CSS селекторі, препроцесор розпізнає аргумент домішки, і застосовує означені в ній стилі до конкретного селектора.

Ще одним позитивним аспектом використання CSS-препроцесорів є можливість вкладати файли у файли, тобто виконувати конкатенацію файлів в заданій послідовності. Хоча це можна і реалізувати на чистому CSS, але в поєднанні з іншими можливостями виходить дуже потужний інструмент. При такому підході надається можливість використовувати вже готові модулі (бібліотеки домішок), які створили інші розробники.

Розглянемо далі деякі особливості використання найпопулярніших CSS-препроцесорів (Sass (SCSS), Less, Stylus).

Змінні в CSS-препроцесорах.

Змінні можуть оголошуватися і використовуватися у всій таблиці стилів. Вони можуть приймати будь-яке значення, яке є доступним в CSS (наприклад, колір, число чи текстове значення), і на них можна посилатися у всіх таблицях стилів проекту.

Sass:

Назви змінних в Sass починаються зі знаку $, а значення та ім'я розділяються двокрапкою, як і властивості CSS.

```
$mainColor: #AA82C1;
$pageWidth: 1200px;
$borderStyle: dotted;

body {
  color: $mainColor;
  border: 1px $borderStyle $mainColor;
  max-width: $pageWidth;
}
```

Less:

Назви змінних в Less і їх оголошення подібно до Sass. Єдиною відмінністю є те, що імена змінних починаються з символа @.

```
@mainColor: #AA82C1;
@pageWidth: 1200px;
@borderStyle: dotted;

body {
  color: @mainColor;
  border: 1px @borderStyle @mainColor;
  max-width: @pageWidth;
}
```

Stylus:

Оголошення змінних в Stylus не вимагає додавання якихось специфічних символів, хоча допускається символ $. В кінці рядка крапка з комою не вимагається, але присвоєння значень змінній відбувається з використанням «=». Потрібно звертати особливу увагу на відступи, які формують межі блоків. Приклад оголошення змінних і їх використання наведено нижче. В прикладі можна звернути увагу на відступи, які в скомпільованому css файлі будуть формувати відповідний блок.

```
mainColor = #AA82C1;
pageWidth = 1200px;
$borderStyle = dotted;

body
```

```
  color mainColor;
  border 1px @borderStyle mainColor;
  max-width pageWidth;
```

Скомпільований CSS:

Кожен з вищенаведених файлів буде скомпільований в ідентичний CSS. На даному прикладі можна побачити наскільки корисними і зручними є змінні. У випадку використання відпадає потреба змінювати в багатьох місцях значення, наприклад кольору, а тільки присвоїти нове значення змінній. На виході після компіляції отримаємо наступний CSS.

```
body {
  color: #AA82C1;
  border: 1px dotted #AA82C1;
  max-width: 1200px;
}
```

Вкладеність в CSS-препроцесорах.

Якщо виникає потреба посилатися на кілька елементів з одним і тим же батьківським селектором. Така необхідність в класичному CSS приводить до значного дублювання коду. Наприклад:

```
section {
  margin: 12px;
}
section nav {
  height: 24px;
}
section nav a {
  color: #AA82C1;
}
section nav a:hover {
  text-decoration: underline;
}
```

Вирішити проблему дублювання можна з використанням препроцесорів шляхом розміщення дочірніх селекторів в середині батьківського, огорнувши їх фігурними дужками. Слід зазначити, що символ & позначає батьківський селектор у всіх препроцесорах.

Sass, Less і Stylus:

Синтаксис вкладеності селекторів у препроцесорах Sass, Less і Stylus є однаковим. Далі наведено приклад вкладеності.

```
section {
  margin: 12px;
  nav {
    height: 24px;
    a {
      color: #AA82C1;
      &amp;:hover {
        text-decoration: underline;
      }
    }
  }
}
```

Скомпільований CSS:

Далі наведено приклад скомпільованого CSS. Це скомпільований CSS з наведеного вище коду. І в цьому випадку досить зручним є використання препроцесорів.

```
section {
  margin: 12px;
```

```
}
section nav {
  height: 24px;
}
section nav a {
  color: #AA82C1;
}
section nav a:hover {
  text-decoration: underline;
}
```

Mixins в CSS-препроцесорах.

Mixins – це функції, які забезпечують повторне використання властивості чи групи властивостей у всій таблиці стилів проекту. Подібно до змінних, можна внести зміни в одному місці без дубляжу. Це може бути корисним у випадку для конкретного стилю елемента. Коли mixins викликається в певному селекторі CSS, аргументи mixin розпізнаються, а стилі, описані в mixin, застосовуються до селектора.

Sass:

```
/* Sass mixin error with (optional) argument $borderWidth which defaults
to 2px if not specified */
@mixin error($borderWidth: 2px) {
  border: $borderWidth solid #F00;
  color: #F00;
}

.generic-error {
  padding: 20px;
  margin: 4px;
  @include error(); /* Applies styles from mixin error */
}
.login-error {
  left: 12px;
  position: absolute;
  top: 20px;
  @include error(5px); /* Applies styles from mixin error with argument
$borderWidth equal to 5px*/
}
```

Less:

```
/* LESS mixin error with (optional) argument @borderWidth which defaults
to 2px if not specified */
.error(@borderWidth: 2px) {
  border: @borderWidth solid #F00;
  color: #F00;
}

.generic-error {
  padding: 20px;
  margin: 4px;
  .error(); /* Applies styles from mixin error */
}
.login-error {
  left: 12px;
  position: absolute;
  top: 20px;
  .error(5px); /* Applies styles from mixin error with argument
@borderWidth equal to 5px */
}
```

Stylus:

```
/* Stylus mixin error with (optional) argument borderWidth which defaults
to 2px if not specified */
error(borderWidth= 2px) {
  border: borderWidth solid #F00;
  color: #F00;
}

.generic-error {
  padding: 20px;
  margin: 4px;
  error(); /* Applies styles from mixin error */
}
.login-error {
  left: 12px;
  position: absolute;
  top: 20px;
  error(5px); /* Applies styles from mixin error with argument borderWidth
equal to 5px */
}
```

Скомпільований CSS:

```
.generic-error {
  padding: 20px;
  margin: 4px;
  border: 2px solid #f00;
  color: #f00;
}
.login-error {
  left: 12px;
  position: absolute;
  top: 20px;
  border: 5px solid #f00;
  color: #f00;
}
```

Наслідування в CSS-препроцесорах.

При написанні CSS класичним способом часто виникає потреба одночасного застосування певних стилів до декількох елементів. Наприклад:

```
p,
ul,
ol {
  /* styles here */
}
```

Такий підхід відмінно працює, але при необхідності додаткової стилізації окремо взятих елементів, для кожного з них потрібно створювати додатковий селектор, що в результаті призведе до об'ємності коду і складності в обслуговуванні. Щоб уникнути таких проблем варто використовувати наслідування при побудові CSS-стилів.

Sass & Stylus:

```
.block {
  margin: 10px 5px;
  padding: 2px;
}

p {
  @extend .block; /* Inherit styles from '.block' */
  border: 1px solid #EEE;
}
ul, ol {
```

```
  @extend .block; /* Inherit styles from '.block' */
  color: #333;
  text-transform: uppercase;
}
```

Скомпільований CSS (Sass & Stylus):

```
.block, p, ul, ol {
  margin: 10px 5px;
  padding: 2px;
}
p {
  border: 1px solid #EEE;
}
ul, ol {
  color: #333;
  text-transform: uppercase;
}
```

На відміну від Sass і Stylus в препроцесора LESS є певна особливість наслідування дійсно не буде наслідувати стилі Sass і Stylus. В LESS наслідування відбувається подібно до використання mixin, тобто відбувається імпорт властивостей у селектор. Недоліком такого підходу є дублювання властивостей у скомпільованій таблиці стилів. Далі наведено приклад наслідування в препроцесорі LESS та результат компіляції в CSS.

Less:

```
.block {
  margin: 10px 5px;
  padding: 2px;
}

p {
  .block; /* Inherit styles from '.block' */
  border: 1px solid #EEE;
}
ul, ol {
  .block; /* Inherit styles from '.block' */
  color: #333;
  text-transform: uppercase;
}
```

Скомпільований CSS (Less):

```
.block {
  margin: 10px 5px;
  padding: 2px;
}
p {
  margin: 10px 5px;
  padding: 2px;
  border: 1px solid #EEE;
}
ul,
ol {
  margin: 10px 5px;
  padding: 2px;
  color: #333;
  text-transform: uppercase;
}
```

Варто звернути увагу на стилі з класу «.block», які були вставлені у селектор «p». Слід зазначити, що в даному випадку можливі проблеми з пріоритетністю стилів. Тому на це варто звертати увагу.

Імпорт файлів у CSS-препроцесорах.

Більшість розробників не схвалюють імпорт CSS, оскільки для цього необхідно робити декілька HTTP-запитів. Проте, імпорт в препроцесорах працює по-іншому. Якщо використовується імпорт файлу в Sass, Less чи Stylus, то цей процес виконається на стадії компіляції, створюючи тільки один файл.

Sass, Less & Stylus:

```
/* file.{type} */
body {
  background: #EEE;
}
```

```
@import "file.{type}";

p {
  background: #0982C1;
}
```

Скомпільований CSS:

```
body {
  background: #EEE;
}
p {
  background: #0982C1;
}
```

Можливості роботи з кольором у CSS-препроцесорах.

В CSS-препроцесорах передбачені вбудовані функції роботи з кольорами, які виконують відповідні маніпуляції з кольором при компіляції. Це може бути надзвичайно корисним для створення градієнтів, більш темних кольорів і багато іншого.

Sass:

```
lighten($color, 10%); /* returns a color 10% lighter than $color */
darken($color, 10%);  /* returns a color 10% darker than $color */

saturate($color, 10%);   /* returns a color 10% more saturated than $color
*/
desaturate($color, 10%); /* returns a color 10% less saturated than $color
*/

grayscale($color);  /* returns grayscale of $color */
complement($color); /* returns complement color of $color */
invert($color);     /* returns inverted color of $color */

mix($color1, $color2, 50%); /* mix $color1 with $color2 with a weight of
50% */
```

Це лиш короткий список доступних функцій роботи з кольорами в Sass. Повний список доступних функцій Sass можна отримати в офіційній документації Sass.

Функції роботи з кольорами можна використовувати в будь-якому місці, де використовується колір. Наприклад:

```
$color: #0982C1;

h1 {
  background: $color;
  border: 3px solid darken($color, 50%);
```

```
}
```

Less:

```
lighten(@color, 10%); /* returns a color 10% lighter than @color */
darken(@color, 10%);  /* returns a color 10% darker than @color */

saturate(@color, 10%);   /* returns a color 10% more saturated than @color
*/
desaturate(@color, 10%); /* returns a color 10% less saturated than @color
*/

spin(@color, 10);  /* returns a color with a 10 degree larger in hue than
@color */
spin(@color, -10); /* returns a color with a 10 degree smaller hue than
@color */

mix(@color1, @color2); /* return a mix of @color1 and @color2 */
```

Список всіх функцій Less можна знайти в офіційній документації.
Далі наведено приклад використання функції роботи з кольорами в Less:

```
@color: #0982C1;

h1 {
  background: @color;
  border: 3px solid darken(@color, 50%);
}
```

Stylus:

```
lighten(color, 10%); /* returns a color 10% lighter than 'color' */
darken(color, 10%);  /* returns a color 10% darker than 'color' */

saturate(color, 10%);   /* returns a color 10% more saturated than 'color'
*/
desaturate(color, 10%); /* returns a color 10% less saturated than 'color'
*/
```

Список всіх функцій Stylus можна знайти в офіційній документації.
Далі наведено приклад використання функції роботи з кольорами в Stylus:

```
color = #0982C1

h1
  background color
  border 3px solid darken(color, 50%)
```

Математичні операції у CSS-препроцесорах.

Використання математичних операцій в CSS-препроцесорах є досить корисним і доступним. Далі наведено приклади використання математичних операцій при заданні значень властивостей.

```
body {
  margin: (14px/2);
  top: 50px + 100px;
  right: 100px - 50px;
  left: 10 * 10;
}
```

Використання вендорних префіксів (Vendor Prefixes)

В процесі розробки коду часто виникають моменти, які приводять до необхідності використання вендорних префіксів, тобто специфічних префіксів, які додаються до CSS-селекторів в залежності від браузера. В даному випадку використання препроцесорів є досить доречним і актуальним, так як економить достатньо часу і зусиль. В цьому випадку використовуються mixin для обробки відповідних префіксів. Далі це продемонстровано на прикладі.

Sass:

```
@mixin border-radius($values) {
  -webkit-border-radius: $values;
     -moz-border-radius: $values;
          border-radius: $values;
}

div {
  @include border-radius(10px);
}
```

Less:

```
.border-radius(@values) {
  -webkit-border-radius: @values;
     -moz-border-radius: @values;
          border-radius: @values;
}

div {
  .border-radius(10px);
}
```

Stylus:

```
border-radius(values) {
  -webkit-border-radius: values;
     -moz-border-radius: values;
          border-radius: values;
}

div {
  border-radius(10px);
}
```

Скомпільований CSS:

```
div {
  -webkit-border-radius: 10px;
     -moz-border-radius: 10px;
          border-radius: 10px;
}
```

Деякі особливості використання CSS-препроцесорів.

Розглянемо найбільш поширені особливості використання CSS-препроцесорів, зокрема, повідомлення про помилки та коментарі до коду.

Повідомлення про помилки.

Досить часто буває, що розробка CSS коду розтягується на великий проміжок часу і в процесі можуть виникати помилки, які впливають на загальний вигляд web-сторінок. І досить часто пошук місць, де були зроблені помилки, займає багато часу і зусиль. В цьому випадку CSS-препроцесори мають значну перевагу над класичним CSS, оскільки здатні відслідковувати такі місця і повідомляти розробника про це та чому виникає помилка.

Коментарі.

В процесі компіляції CSS, однорядкові коментарі, оголошені «//», видаляються, а коментарі, оголошені «/*» залишаються. Тому цю особливість потрібно враховувати при розробці коду. Також, слід зазначити, що при компіляції мінімізованих CSS файлів всі коментарі будуть видалені.

Підсумовуючи розгляд CSS-препроцесорів, варто зазначити, що кожен з них характеризується своїми певними можливостями, які забезпечують чистоту коду, передбачають використання корисних функцій, які недоступні в класичному CSS, є сумісними з усіма популярними браузерами та багато іншого. Тому слід у роботі використовувати CSS-препроцесори.

Контрольні запитання

1. Означити поняття CSS-препроцесора.
2. Охарактеризувати використання змінних у CSS-препроцесорах.
3. Охарактеризувати використання вкладеностей у CSS-препроцесорах.
4. Означити поняття «mixin» в CSS-препроцесорах.
5. Означити процес наслідування в CSS-препроцесорах.
6. Імпорт файлів у CSS-препроцесорах.
7. Математичні операції в CSS-препроцесорах.
8. Робота з кольорами в CSS-препроцесорах.
9. Використання вендорних префіксів у CSS-препроцесорах.
10. Особливості опрацювання помилок в коді та коментування коду.

ЗАВДАННЯ

Використовуючи результати попередньої лабораторної роботи (базову розмітку) та макет web-сторінки, розробити відповідні css-стилі. Необхідно використати css-препроцесор згідно варіанту завдання.

Лабораторна робота 4.

ТЕМА: Використання засобів JavaScript при розробці web-сайту.

МЕТА: Ознайомитися з основними засобами JavaScript при розробці web-сайту та їх практичне застосування.

ТЕОРЕТИЧНІ ВІДОМОСТІ

JavaScript – мова програмування, яка застосовується при створенні HTML-сторінок для збільшення їх функціональності та можливостей взаємодії з користувачем. JavaScript є однією із складових динамічного HTML. Ця мова програмування була створена фірмами Netscape та Sun Microsystems на базі мови програмування Sun's Java. На сьогодні є декілька версій JavaScript. За допомогою JavaScript на HTML-сторінці можна зробити те, що не можливо за допомогою стандартних тегів HTML.

Код програми JavaScript розміщується або у середині HTML-сторінки, або у текстовому файлі, що пов'язаний за допомогою спеціальних команд з HTML-сторінкою. Цей код, як правило, розміщується у середині тегу HTML та завантажується у браузер разом з кодом HTML-сторінки. Програма JavaScript не може існувати самостійно, тобто без HTML-сторінки.

Виконання програми JavaScript відбувається при перегляді HTML-сторінки у браузері, звісно, тільки у тому випадку, коли браузер містить інтерпретатор JavaScript. Практично всі сучасні популярні браузери оснащені таким інтерпретатором. Зазначимо, що, крім JavaScript, на HTML-сторінках можливо використовувати інші мови програмування. Наприклад, VBScript або JScript, яка є варіантом JavaScript від фірми Microsoft. Але виконання програм VBScript та JScript гарантовано коректне тільки при перегляді HTML-сторінки за допомогою браузера Microsoft Internet Explorer. Тому у більшості випадків використання JavaScript доцільніше, хоча функціональність програм VBScript та JScript дещо краща.

Досить часто програму JavaScript називають ***скриптом*** або ***сценарієм***. Скрипти виконуються у результаті того, що відбулась деяка подія, пов'язана з HTML-сторінкою. У багатьох випадках виконання вказаних подій ініціюється діями користувача.

Скрипт може бути пов'язаний з HTML-сторінкою двома способами:

- за допомогою парного тегу script;
- як оброблювач події, що стосується конкретного тегу HTML.

Сценарій, вбудований у HTML-сторінку з використанням тегу script, має такий формат:

```
<script>
  // Код програми
</script>
```

Усе, що розміщується між тегами <script> та </script>, інтерпретується як код програми на мові JavaScript. Обсяг вказаного коду не обмежений. Інколи скрипти розміщують у середині HTML-коментаря. Це роблять для того, щоб код JavaScript не розглядався старими браузерами, які не мають інтерпретатора JavaScript. У цьому випадку сценарій має формат:

```
<script>
  <!--
    // Код програми
  -->
</script>
```

Тег script має декілька необов'язкових параметрів. Найчастіше використовуються параметри ***language*** та ***src***. Параметр ***language*** дозволяє визначити мову та версію мови сценарію. Параметр **src** дозволяє задати файл з кодом сценарію. Для кращого розуміння використання параметрів тегу script розглянемо задачу.

Задача. Необхідно для HTML-сторінки hi.html створити сценарій на мові JavaScript для показу на екрані вікна повідомлення з текстом "Привіт!".

Для показу на екрані вікна з повідомленням використаємо функцію alert.

Для ілюстрації можливостей інтегрування скриптів в HTML-код розв'яжемо задачу двома способами.

Спосіб 1. Оголошення сценарію безпосередньо на HTML-сторінці hi.html

```
<html>
  <head>
    <title>Використання JavaScript</title>
  </head>
  <body>
    <script language="JavaScript">
      alert('Привіт!');
    </script>
  </body>
</html>
```

Спосіб 2. Оголошення сценарію у файлі a.js та його підключення до HTML-сторінки hi.html за допомогою параметра src тегу script. Код HTML-сторінки hi.html буде виглядати наступним чином:

```
<html>
  <head>
    <title>Використання JavaScript</title>
    <script language="JavaScript" src="a.js"></script>
  </head>
  <body>
  </body>
</html>
```

Програмний код, записаний у файлі a.js буде мати наступний вигляд:

```
alert('hi')
```

Змінні та вирази JavaScript можна використовувати як значення параметрів тегів HTML. У цьому випадку елементи JavaScript розміщуються між амперсандом (&) та крапкою з комою (;), але повинні бути обмежені фігурними дужками {} і використовуватись тільки як значення параметрів тегів.

Наприклад, нехай визначена змінна *c,* якій присвоєно значення *green*. Приклад застосування змінної в якості значення параметра тегу HTML наведено нижче.

```
<font color="&{c};">Текст зеленого кольору</font>
```

Зазначимо, що мінімальним комплектом програмного забезпеченням для розробки та тестування програм JavaScript є текстовий редактор та браузер з підтримкою JavaScript.

Оголошення та ініціалізація змінних.

У JavaScript передбачено оголошення змінних з використанням трьох операторів:

- var;
- let;
- const.

Слід зазначити, що оператори let і const були введені в JavaScript у специфікації ES2015 (ES6).

Є певні вимоги до назв змінних в JavaScript, тобто вона може починатися з букви латинського алфавіту або символів нижнього підкреслення "_" і знаку долара"$". А наступні символи можуть бути і цифрами. Слід зазначити, що в JavaScript потрібно враховувати регістр букви. Наприклад, variable та Variable це дві різні змінні. Заборонено використовувати для змінних імена, що збігаються з ключовими словами JavaScript.

При оголошенні змінної не обов'язково присвоювати їй значення і тип, а можна це зробити в необхідному місці.

```
let variable;
variable = 'variable value';
```

Також в JavaScript тип змінної може змінюватися.

```
let variable;
variable = 'variable value'; // тип змінної string;
variable = 123; // тип змінної number;
variable = {}; // тип змінної Object;
```

Крім цього, в JavaScript змінній можна присвоювати функцію.

```
let func = function(x, y) {
  return x*y;
};
func(3, 4);
```

Вирази та оператори.

Вираз – це комбінація змінних, літералів та операторів, у результаті обчислення яких можливо отримати тільки одне значення, яке може бути числовим, рядковим або булевим.

Для реалізації обчислень у JavaScript використовуються арифметичні, рядкові, логічні вирази та декілька типів операторів.

Арифметичні вирази повертають число, наприклад a = 7+5;

Рядкові вирази повертають рядок символів, наприклад "Джон" або "234".

Логічні вирази повертають true (істина) або false (хибна).

Оператор присвоєвання (=) присвоює значення лівому операнду, базуючись на значенні правого операнда. Наприклад, для присвоєння змінній *a* значення числа 5 необхідно записати: a = 5

До стандартних арифметичних операторів належать: оператори додавання (+), віднімання (-), множення (*), ділення (/), остача від ділення чисел (%), інкремент (++), декремент (--).

Зазначимо, що оператор додавання можна використовувати не тільки для чисел, але й для додавання (контрактації / конкатенації) текстових рядків.

Для створення логічних виразів використовуються логічні оператори та оператори порівняння.

До логічних операторів належать: логічне І (&&), логічне АБО (||), логічне НІ (!).

Оператори порівняння не відрізняються від таких операторів у інших мовах програмування. До операторів порівняння належать (==, >, >, =<, <=,!=).

Умовні оператори

Керування послідовністю дій, при виконанні сценарію, здійснюються за допомогою операторів. В JavaScript присутній стандартний набір операторів, який унаслідувався від C++ і Java. Зокрема:

- умовний оператор if…else;
- оператор вибору switch;
- оператори циклу for, while, do…while, break и continue;
- оператор ітерації for…in;
- оператор вказання об'єкту with.

В JavaScript будь-який вираз є також оператором.

Умовний оператор if...else.

Даний оператор передбачає перевірку певної умови і, в залежності від результату, виконати визначену послідовність операторів. Є дві форми представлення цього оператора:

```
if (condition) operator1;
```

```
if (condition) operator1 else operator2;
```

В даному випадку умова представляє собою вираз, значення якого приводиться до логічного типу, а operator1 та operator2 можуть представляти будь-яку групу JavaScript операторів. У випадку, коли група містить більше ніж один оператор, то вони розміщаються у фігурних дужках.

```
if (condition) {
  operator1;
  operator2;
  operator3;
}
```

Оператор вибору switch.

Даний оператор призначений для виконання тої чи іншої послідовності операторів, що залежить від значення виразу. Структура оператора:

```
if (expression) {
  case value:
    operators…
    break;
  case value:
    operators…
    break;
  case value:
    operators…
    break;
  default:
    operators…
}
```

В даному випадку вираз (expression) – довільний вираз, значення (value) – можливе значення виразу. Принцип роботи даного оператора полягає в порівнянні значення виразу зі значенням в кожній з case міток. При співпадінні виконується відповідний набір операторів. В іншому випадку виконається набір операторів з міткою default.

Оператори циклу for, while, do…while, break и continue.

Цикл – це послідовність операторів, які виконуються при певній умові. Якщо умова не виконується, припиняється і виконання операторів. В JavaScript є три оператори циклу: for, while і do ... while. Крім цього є ще оператори break і continue, які використовуються безпосередньо в тілі циклів. Також варто відзначити ще один оператор for ... in, який використовується при роботі з об'єктами і його також можна віднести до операторів циклу.

Інколи необхідно закінчити цикл не через умови, що задана у його заголовку, а у результаті виконання рівної умови у тілі циклу. Для цього використовуються оператори ***break*** та ***continue.*** Оператор ***break*** завершує цикл while або for та передає керування програмою першому оператора після циклу. Оператор continue передає управління оператора перевірки істинності умови у циклі while та оператора оновлення значення лічильника у циклі for і продовжує виконання циклу.

Контрольні запитання

1. Навіщо використовується мова JavaScript?
2. Як визначити у HTML- документі код JavaScript?
3. Синтаксис запису змінних у JavaScript.
4. Синтаксис запису виразів у JavaScript.
5. Типи даних JavaScript.
6. Оператори JavaScript.
7. Оператори вибору JavaScript.
8. Оператори циклів JavaScript.

9. Як достроково закінчити цикл?
10. Як достроково перейти на наступну ітерацію циклу?

ЗАВДАННЯ

Використовуючи результати попередніх лабораторних робіт, розробити сценарії поведінки декількох компонентів web-сторінки на мові JavaScript.

Лабораторна робота 5.

ТЕМА: Використання бібліотеки JQuery при створенні web-сайту.

МЕТА: Ознайомитися з основними можливостями бібліотеки JQuery та її використання на практиці.

ТЕОРЕТИЧНІ ВІДОМОСТІ

jQuery – популярна JavaScript-бібліотека з відкритим кодом. Вона була представлена у січні 2006 року у BarCamp NYC Джоном Ресігом (John Resig). Згідно з дослідженнями організації W3Techs, JQuery використовується понад половиною від мільйона найвідвідуваніших сайтів. jQuery є найпопулярнішою бібліотекою JavaScript, яка використовується на сьогоднішній день.

jQuery є вільним програмним забезпеченням під ліцензією MIT (до вересня 2012 було подвійне ліцензування під MIT та GNU General Public License другої версії.

Синтаксис jQuery розроблений, щоб забезпечити орієнтування у навігації зручнішим завдяки вибору елементів DOM, створенню анімації, обробки подій, і розробки AJAX-застосунків. jQuery також надає можливості для розробників при створенні плагінів. Це сприяє створенню потужних і динамічних веб-сторінок.

Доцільність використання бібліотеки jQuery.

Динаміку й інтерактивність сайтів реалізують зазвичай з допомогою сценарію, як правило записаного мовою JavaScript.

Щоб керувати об’єктами, наприклад міняти властивості стилю (колір, розмір символів тощо), потрібно одержати доступ до цих об’єктів. Для цього кожному такому об’єкту надають назву (ідентифікатор) і у відповідний метод (функцію) передають цю назву (ідентифікатор). Якщо потрібно однакові дії виконати над кількома об’єктами, то потрібно створити колекцію об’єктів і проводити з об’єктами цієї колекції відповідні дії. Створити колекцію об’єктів інколи не просто. Для спрощення цієї роботи й використовують бібліотеку jQuery.

Для використання бібліотеки jQuery потрібно підключити відповідний файл. Є два шляхи підключення:

з використанням CDN:

```
<head>
  <script type="text/javascript"
  src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.min.js">
  </script>
</head>
```

або скопіювати цей файл у теку зі сценаріями Java Script (зазвичай з назвою *js*) даного сайту та підключити його наступним чином:

```
<head>
  <script type="text/javascript" src="js/jquery.js">
  </script>
</head>
```

Структура сценарію

Код сценарію зазвичай пишуть чи підключають у голові веб документу. Але цей код, часто, потрібно виконати лише після завантаження *всіх* об’єктів документа. Якщо написали сценарій і розмістили його в голові документа, не вказавши, що його потрібно виконати лише після завантаження документа, його буде виконано для порожнього документа: відповідні теги ще не завантажено. У цьому випадку сценарій потрібно записати таким чином:

```
jQuery(document).ready( сценарій );
```

Круглі дужки позначають початок і кінець сценарію або чи фрагменту (блоку) сценарію. Запис сценарію завершують крапкою з комою. Сценарій реалізують функціями:

```
function( список параметрів ){ тіло функції };
```

У записі функцій для початку та кінця функції чи її фрагменту використовують не круглі, а фігурні дужки. Функції та вказівки в них розділяють крапкою з комою. Після останньої функції jQuery запиту крапку з комою не ставлять. Використовують два типи коментарів:

```
/* коментар всередині рядка чи на кілька рядків */
// коментар до кінця рядка.
```

Напишемо, наприклад, сценарій заміни кольору символів в усіх параграфах (теги p) на синій. Потрібно передбачити очікування на завантаження всього документа до виконання функції пошуку всіх тегів p і заміни у них у стилі "css" кольору "color" на синій "blue":

```
<script>
  jQuery(document).ready(
    function()
    {
      jQuery("p").css("color",  "blue");
    }
  );
</script>
```

Запис `jQuery` можна замінити символом $. Тобто даний сценарій можна переписати таким чином:

```
<script>
  $(document).ready(
    function(){
      $("p").css("color",  "blue");
    }
  );
</script>
```

Пошук та редагування вмісту сайту.

Пошук можна робити за простою умовою за таким зразком:

```
$("h2")       // вибір всіх тегів h2
$("#content") // вибір елементів за ідентифікатором: id=content
$(".wrapper") // вибір елементів за класом: class=wrapper
```

Пошук можна робити й за складною умовою, складаною з кількох простих, об'єднаних логічними (вказівками):

- *і* — при відсутності знаків пунктуації;
- *або* — при розділенні умов комою.

Наведемо приклад використання *або:*

```
$("h1, h2")         // вибір всіх тегів h1 і h2
$(".wrapper, .box") // вибір елементів з класів class=wrapper і class=box
```

та *і:*

```
$("div.wrapper")  // вибір тегів div з класу class=wrapper
$("div#content")  // вибір тегів div з ідентифікатором id=content
```

Зауважимо: ідентифікатор в одному файлі може бути тільки в одному тегові, тому остання умова є доцільною лише при пошуку в кількох файлах.

Можна шукати об’єкти, які розташовані всередині вказаних об’єктів. У запиті на пошук потрібно вказати, в яких об’єктах шукати, а потім після пропуску — ті об’єкти, які шукати. Наприклад:

```
$("div  h2")  // вибір всіх тегів h2 всередині тегів div
$("div p h2") // вибір всіх тегів h2 всередині тегів p, розташованих у
тегах div
```

Ці приклади можна записати більш наочно з використанням методу find(" "), перед записом якого ставлять крапку, щоб вказати, що цей метод викликають для вказаних чи знайдених раніше об’єктів:

```
$("div").find("h2")
$("div").find("p").find("h2")
```

Можна вибирати елементи, які розташовані перед чи після вказаних у пошуку об’єктів. Вибирати можна також всіх прямих попередників по ієрархії та підлеглих об’єктів, як безпосередніх так і до кінця ієрархії.

Зазвичай об’єкти шукають з метою здійснити з ними певні перетворення. Наприклад, змінити елементи стилю, використовуючи метод css():

```
css( властивість ) — одержання величини властивості;
css( властивість, величина ) — надати властивості величину;
css( властивість1: величина1, властивість2: величина2, … ) — надати
величини кільком властивостям.
```

Приклад:

```
$("#my").css('color')        // одержати величину кольору шрифту
                             // об’єкту з ідентифікатором my
$("div").css('color', 'red') // змінити колір шрифту блоків div на
червоний
$("div").css({                    // у блоках div змінити:
              'color':'red',      // колір шрифту на червоний
              'font-size':'14pt', // розмір шрифту на 14 пунктів
              'margin-left':'10px' // ширину поля зліва на 10 пікселів
             })
```

Апострофи в назвах, а не величинах, можна опустити. Але тоді назви потрібно записати за правилами мови JavaScript. Наприклад, 'font-size' потрібно замінити на fontSize, а 'margin-left' — на marginLeft.

Події.

У мові JavaScript є кілька типів (груп) подій:

- роботи з мишею;
- роботи з клавішами;
- роботи з формою;
- завантаження / вивантаження документу або його об’єктів

та інші. З опрацюванням події завантаження документу ми вже познайомилися при розгляді структури сценарію. Розглянемо використання подій роботи з мишею (подано у хронологічному порядку, ЛКМ — ліва клавіша миші):

- *mouseover* – входження вказівника в об’єкт;
- *mousedown* – натиснення ЛКМ, коли вказівник у даному об’єкті;
- *mouseup* – відпускання ЛКМ, коли вказівник у даному об’єкті;
- *click* – натискання й відпускання ЛКМ, коли вказівник у даному об’єкті;
- *dblclick* – подвійне натискання ЛКМ, коли вказівник у даному об’єкті;
- *mouseout* – вихід вказівника з об’єкту;

Розглянемо частину сценарію з використанням події *click*.

```
$(document).ready(function(){  $(".button").click(
/*при натисненні на будь який блок класу button -
  виклик такої функції зміни параметрів блоків: */
  function(){   /* зміна параметрів усіх блоків */
    $("div").css({color:'red',
        backgroundColor:'00ff00',
                border:'2px solid 000000'})
    /* потім - зміна параметрів блоків класу "button" */
    $(".button").css({color:'ffff00',
           backgroundColor:'0000ff',
                   border:'3px solid ff0000'})
    /* наприкінці зміна параметрів блоку з id=d1 */
    $("#d1").css({color:'00ffff',
        backgroundColor:'ff00ff',
                border:'3px solid 00ff00'})
  })
});
```

Об’єкти WEB сторінки набувають вигляду з урахуванням початкових налаштувань і всіх здійснених над кожним об’єктом дій. Якщо, наприклад, у жодній дії з блоками ми не змінювали розмір шрифту, то кожен блок має свій розмір шрифту, вказаний у стилі цього блоку. Якби у поданому вище сценарії ми змінили колір символів (атрибут стилю color), а для блоків класу "button" та блоку з id=d1 колір символів не змінювали, то колір символів в усіх блоках був би однаковий, одержаний при діях з блоками.

Згортання й розгортання об'єктів.

Як бачимо, використання бібліотеки jQuery спрощує написання сценаріїв мовою JavaScript. Це стосується також і поки не розглянутих нами питань: пошуку й заміни тексту, опрацювання вмісту форм, створення анімації об’єктів.

Не всі браузери мають однакові методи пошуку та заміни тексту. Анімацію об’єктів у браузері Internet Explorer реалізовано з допомогою динамічних і статичних фільтрів. А в інших браузерах або зовсім не реалізовано, або реалізовано інакше.

У бібліотеці jQuery анімацію реалізовано подібно до динамічних фільтрів у браузері Internet Explorer: для вказаних часу та типу переходу з одного стану в інший: «показати» / «сховати». Час переходу задають у мілісекундах. Тип і стан вказують разом. Наприклад, при переході *hide* відбувається згортання об’єкта *у точку* догори, а при переході *show* — обернене розгортання *з точки* в об’єкт. У поданому далі коді при події *click* у одному з об'єктів класу *button*, викликається функція, що згортає, а потім негайно розгортає цей самий об'єкт.

```
<script type="text/javascript">
$(document).ready(function(){
$(".button").click(function(){$("#box").hide(1000);
                               $("#box").show(1000);
                              });
                             });
</script>
```

Переходи *slideUp* і *slideDown* вказують на те, що згортання відбувається знизу вгору у *відрізок*, а розгортання — навпаки.

Як бачимо, бібліотеки jQuery не лише спрощує написання сценаріїв мовою JavaScript, а й дозволяє не враховувати особливості реалізації сценаріїв сучасними браузерами. Надалі доцільно розглянути використання бібліотеки jQuery для:

- пошуку та заміни тексту;
- реалізації динамічних меню;
- опрацювання даних форм;
- анімації об’єктів.

Контрольні запитання

1. Навіщо використовується бібліотека jQuery?
2. Обґрунтувати доцільність використання бібліотеки jQuery?
3. Розкрити структуру сценарію jQuery.
4. Розкрити суть пошуку та редагування вмісту сайту з використанням jQuery.
5. Охарактеризувати роботу з типами JavaScript подій з використанням jQuery.

ЗАВДАННЯ

Використовуючи результати попередніх лабораторних робіт, розробити сценарії поведінки декількох компонентів web-сторінки, застосувавши бібліотеку JQuery.

Лабораторна робота 6.

ТЕМА: Використання AJAX підходу до побудови користувацьких інтерфейсів web-сайту.
МЕТА: Ознайомитися з основними можливостями AJAX підходу до побудови користувацьких інтерфейсів web-сайту та його практичне застосування.

ТЕОРЕТИЧНІ ВІДОМОСТІ

$.ajax() функція.

В jQuery $.ajax() функція використовується для виконання асинхронного запиту HTTP. Вона була додана в бібліотеку давно, присутня з версії 1.0. Параметри цієї функції показано нижче:

```
$.ajax(url[, options])
$.ajax([options])
```

Параметр **url** – це рядок, що містить URL, який ви хочете досягти з допомогою Ajax-запиту, а **options** – це об'єкт, що містить налаштування для Ajax-запиту.

В своїй першій формі, функція виконує Ajax-запит за допомогою параметра url і параметрів, зазначених в налаштуваннях. У другій формі, URL, зазначений у параметрі options, може бути пропущений, у цьому випадку, запит виконується до поточної сторінки.

Перелік опцій цієї функції дуже довгий. Тож ми будемо намагатися скоротити опис. У разі, якщо ви хочете поглиблено вивчати їх сенс, ви можете звернутися до **офіційної документації** $.ajax().

Параметр option.

Є багато варіантів властивостей для $.ajax (). У списку нижче, Ви можете знайти їх імена і опис в алфавітному порядку:

- **accepts**: Тип вмісту, відправляється в заголовку запиту. Вказує серверу, яку відповідь він буде приймати по поверненню.
- **async**: встановіть значення *false*, щоб виконати синхронний запит.
- **beforeSend**: pre-request функції зворотного виклику, який може бути використаний, щоб змінити об'єкт *jqXHR* перед відправкою.
- **cache**: встановіть опції значення *false*, щоб змусити запитувані сторінки не кешуватися браузером.
- **complete**: функція викликається, коли закінчується запит (після *success* і *error* виконуються зворотні виклики).
- **contents**: об'єкт, який визначає, яка бібліотека буде аналізувати відповідь.
- **contentType**: тип вмісту, що відправляється на сервер.
- **context**: об'єкт для використання в якості контексту для всіх Ajax-пов'язаних зворотніх викликів.
- **converters**: об'єкт, що містить “тип даних в тип даних” для перетворювачів.
- **crossDomain**: встановити цю властивість в *true* для кросдоменного запиту (наприклад, JSONP) на тому ж домені.
- **data**: дані для відправки на сервер при виконанні Ajax-запиту.
- **dataFilter**: функція використовується для обробки raw-даних XMLHttpRequest.
- **dataType**: тип очікуваних даних з сервера.
- **error**: функція, яка буде викликана, якщо запит не буде виконано.
- **global**: Надати можливість викликати глобальні обробники подій Ajax для цього запиту.
- **headers**: додаткові заголовки для відправки на сервер.
- **ifModified**: встановіть цей параметр як *true*, якщо ви хочете вважати запит успішним лише у випадку, якщо відповідь змінилася з моменту останнього запиту.
- **isLocal**: встановіть цей параметр як *true*, якщо ви хочете змусити jQuery визнати поточне середовище локальним.
- **jsonp**: рядок для зміни імені функції зворотного виклику в JSONP запиті.

- **jsonpCallback**: задає ім'я функції зворотного виклику для JSONP запиту.
- **mimeType**: рядок, який вказує тип mime для перевизначення XHR mime.
- **password**: пароль з XMLHttpRequest у відповідь на запит HTTP аутентифікації.
- **processData**: встановіть цей параметр як *false*, якщо Ви не хочете, щоб дані в опції **data** (якщо вони не є рядком) були оброблені і перетворені в рядок запиту.
- **scriptCharset**: встановлює атрибут charset для тегуі script, що використовується в запиті, але застосовується тільки тоді, коли відбувається транспортування сценарію.
- **statusCode**: об'єкт числових кодів HTTP і функції, що викликаються при відповіді на відповідний код.
- **success**: функція, яка буде викликана, якщо запит успішно виконано.
- **timeout**: число, яке задає час очікування (в мілісекундах) для запиту.
- **traditional**: встановити як *true*, якщо ви хочете використовувати традиційний стиль для сериалізації параметрів.
- **type**: тип запиту, який може бути або "POST" або "GET".
- **url**: рядок, що містить URL-адресу, на яку надсилається запит.
- **username**: ім'я користувача для використання з XMLHttpRequest у відповідь на запит HTTP аутентифікації.
- **xhr**: зворотний виклик для створення об'єкта XMLHttpRequest.
- **xhrFields**: об'єкт, щоб встановити нативний XHR об'єкт.

Приклад формування ajax запитів.

```
$.ajax({
    type: "POST",
    url: "request_url",
    data: data,
    success: function(response) {
        code_for_success_response;
    },
    error: function() {
        code_for_error_response;
    }
});
```

Контрольні запитання

1. Дати означення $.ajax() функції.
2. Означити основні параметри $.ajax() функції.
3. Який з параметрів дозволяє створити синхронний запит?
4. Яка функція виконається після успішного запиту?
5. Яка функція виконається, якщо запит буде неуспішним?
6. Які типи запитів передбачені в $.ajax() функції?
7. Яка з властивостей відповідає за передачу даних?
8. Яка з властивостей відповідає за URL-адресу, на яку надсилається запит?

ЗАВДАННЯ

Створити файл data.json, розробити структуру об’єкта та заповнити всі його поля інформацією, яка буде відображатися на web-сторінці. Використовуючи результати попередніх лабораторних робіт, створити асинхронні AJAX запити (методи GET та POST), які будуть використовувати дані з файлу data.json.