ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра комп'ютерної інженерії

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА СИСТЕМ АВТОМАТИЗОАЦІЇ ПОБУДОВИ WEB СТОРІНОК З ВИКОРИСТАННЯМ PUG ТА SASS ПРЕПРОЦЕСОРІВ**»

Виконав: студент 4 курсу, групи ПД–41
спеціальності

123 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Жужков Д. І.
(прізвище та ініціали)

Керівник Бондарчук А. П.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра - Інженерії програмного забезпечення
Ступень вищої освіти - «Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач
кафедри
Інженерії
програмного
забезпечення

Негоденко О.В,

“___” 2022
року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Жужков Денис Ігорович
(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка систем автоматизації побудови web сторінок з використанням Pug та Sass препроцесорів

Керівник роботи: Бондарчук Андрій Петрович,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від « 16 » лютого 2022 року № 22 .
2. Строк подання студентам роботи
3. Вхідні дані до роботи:
3.1Стандарти написання коду;
3.2Препроцесори Sass Pug;
3.3Стандарти створення інтерфейсу;

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).
4.1 Розробка оточення для використання SASS(SCSS) та PUG прероцесорів.

4.2 Аналіз дизайн системи для створення помпонентів.
4.3 Розробка компонентів та побудова статичних сторінок.

5. Перелік демонстраційного матеріалу (назва основних слайдів)

6. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз існуючих технологій	01.03.22 – 03.03.22	
2	Розробка оточення з використання NPM пакет менеджера	04.03.22 – 05.03.22	
3	Розробка архітектури директорій	06.03.22 – 07.03.22	
4	Створення компонентів для автоматизації	08.03.22 – 09.03.22	
5	Створення сторінки використовуючи готові компоненти	10.03.22 – 12.03.22	
6	Висновки	29.03.22 – 30.03.22	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 62 с., 13 рис., 45 джерел.

Об'єкт дослідження (розробки) – Старення автоматизації для можливості перевикористання в подальшому .

Предмет дослідження (розробки) – компоненти Pug та SASS(SCSS) .

Мета роботи – розробка компонентів для системи автоматизації створення WEB сторінок

Методи дослідження(Розробки) – аналіз дизайн системи, на основі якого проєктуються компоненти.
Створення компонентів з використанням міксінів для подальшого перевикористання

Галузь використання – будь яка компанія, яка має відношення до розробки веб-сторінок.

BEM методологія, СТАНДАРТИ НАПИСАННЯ КОДУ, NPM, JAVASCRIPT, NODE JS, UI/UX, SASS(SCSS) PUG .

ЗМІСТ

ВСТУП

З кожним роком попит на створення сайтів все більшає, з ним і росте питання автоматизації процесів оскільки багато сайтів мають аналогічну структуру.

Створення веб сторінки можна поділити на декілька єтапів:

- Проєктування дизайну
- Створення статичної HTML сторінки
- Розробка функціоналу
- Архітектура бек-енду та побудова баз данних
- Відправка на сервер

Ми будемо розглядати побудову HTML сторінок та можливості автоматизації розробки для зменшення вокористованого нами часу на ствренння сторінок за допомогую перевикоритання побудованих нами рішень.

Побудова сторінок також має однакові частини які трохи змінні за дизайном але передають одну суть. Скільки б не було макетів різних сторінок продукту, вони всі мають кнопки, списки, таблиці, навігацию якусь текстову частину. В них може відрізнятися відступ, колір, наявність іконок, поведінка при взаємодії, розміри...
Але всі вони належать до конкретної сутності і в правильному дизайні вони будуть у певних випадках однаковими.

Наша головна ціль знайти ці закономірності спроєктувати архітектуру окремоі компоненти як цілисній сутності і в подальших просто вже користуватися

готовим рішенням, і при якихось глобальних змінах редагувати лише в одному місці не витрачаючи час ні інші стрінки. Для цього на щастья буди створені такі припроцесори як SASS(SCSS) для CSS стилів та PUG для HTML принцип іх роботи дуже між собою схожий. Вони можуть створювати окремі сутності вже готовий набір стилів та HTML розмітки.

Також ми будемо використовувати пакетний менеджер NPM для допосміжних сервісів які допоможуть в подальшій роботі. Налагодимо архрітектуру директорій та систему автомотичного перезбору та генерації HTML сторінок.

Основними критеріями для побудови окремих та самодостатніх компонентів буде аналіз дизайну для розподілу на окремі сутності та проєктування структури, перевірка на можливість вміщати одній компоненті в соб іншу компоненту які вдже зможуть створювати з себе окремі модулі.

Основою та принципом такоі системси відноситься BEM методологія

BEM – (блок, елемент модифікатор)

Із допоміжних систем нам також знадобиться текстовий редактор WebShtorm

Також основними технологіями для розробки такої системи є NodeJS на якому буде підійматися весь проєкт. Для MAC OS можна використовувати пакетний менеджер HOMEBREW

1 АНАЛІЗ ТЕХНОЛОГІЙ

1.1 Основні технології

Основні технології створення вебсторінок можна поділити на 3 частини:

- CSS in JS
- Post/Pre процесори
- фреймворки

1.2 Принципи CSS in JS

Основний принцип складається з того що створюються окремі модулі в яких одразу і функціональна частина і візуальна. Бібліотеки CSS in JS генерують унікальні імена класів. Така методика започаткована CSS модулями.

Усі стилі обмежені окремим компонентом, і здійснюють інкапсуляцію[1] не впиваючи на інші стилі. Така система вирішує проблему конфліктів стилів коли одні частини коду можуть впливати на інші.

І в результаті зображення сторінки може ламатися, бути не коректним чи не відповідним до дизайну. Також використовуються генерування автопрефіксів для окремих css параметрів для різних браузерів які особливо на них реагують.

Сама генерація проходить по окремому модулю і генерує свій css для нього одразу на html-сторінку.

1.2.1 Функціональність перевірки

Перевірка повинна включати в себе загальні положення з посібників JavaScript коду, а також положення, розроблені для фреймворку SAP UI5. Слід також мати можливість дезактивації контролю за певними правилами через специфіку конкретних проектів.

1.2.2 Розширюваність

У рамках проекту передбачена перевірка коду, що відповідає лише JavaScript, але має бути передбачена можливість розширення системи:

- динамічна перевірка коду;
- додаткові статичні перевірки.

Необхідно створити практичні інструменти для впровадження нових правил, характерних для нових версій SAP UI5 Framework

1.2.3 Веб-інтерфейс

Основне завдання прототипу розробленої системи - надати розробникам SAP можливість без додаткових витрат часу і зусиль проводити аналіз коду клієнта. У зв'язку з цим система повинна мати графічний веб-інтерфейс, який після завантаження коду проекті відображатиме статистичний звіт у таблицях, а також звіт про конкретні місця коду, які не пройшли перевірку.

1.2.4 Плюси підходу CSS in JS

Як сказано в основних принципах такого підходу, унікальність стилів є неоціненною функцією для розробки на основі компонентів

Не буде такої ситуації коли однакові імена класів різних сутностей можуть повпливати одне на одного.

1.2.5 Мінуси підходу CSS in JS

Унікальність стилів також можна вважати мінусом зі сторони рефакторингу та пошуку проблем на готовому продукту оскільки назви змінюються та шифруються. Їх читабельність стає важчою.

Також генерація css коду створює додаткове навантаження на сторінку і додатковий час для повної завантаження сторінки.

1.3 Принципи фреймворків

Фреймворки в собі містять готові рішення і вже написаний інтерфейс зі

своїми стилями. Найпопулярнішим вважають Bootstrap також є аналоги: Foundation, UIkit, Sematic UI і т. д. Для використання достатньо знайти якийсь блок який вам підходить і скопіювати його собі у код

Bootstrap складається з інструментів для створення макетів класу стилізації базового контенту готових компонентів типу кнопок форм навігації слайдерів випадаючих списків акордеоні в якихось вікон модальних вікон. Основні переваги bootstrap це висока швидкість створення адаптивної статики, крос браузерність, Наявність великої кількості готових рішень

Основними недоліками вважається велика кількість лишнього коду, неможливість Створення сайтів зі специфічним дизайном однотипність готових рішень. Використання застарілих технологій.

1.4 Post/Pre процессори

До Post/Pre процесорів можна віднести Sass/Scss, Less Stylus PostCss. Вони або генерують код вже на сторінці або ще до створення.

Основні найпопулярніші препроцесори статичних сторінок це Pug та Sass. Які проблеми вирішують:

- Модульність
- Наслідування
- Форматування
- Постскриптум
- Плагіни

Препроцесори допомагають економити час на створення сторінок, дають можливість автоматизувати та побудувати зручні процеси для створення вебсторінок.

1.5 Аналіх технологій

На рисунку 1.1 зображенно популярність технологій та іх рівень

задоволеності де Sass(Scss) виграє по всім пунктам, а фреймворк будстрам буде з кожним днем втрачати популярність (рисунок 1.1).

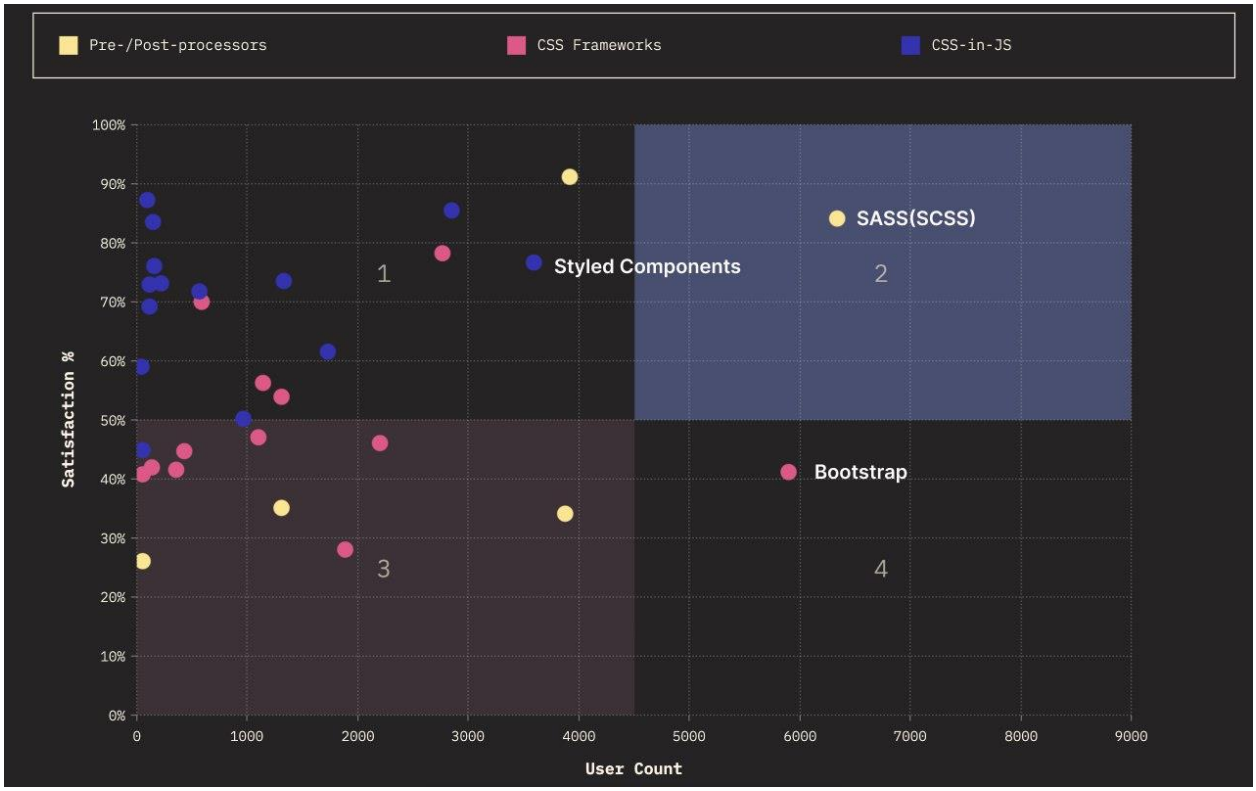


Рисунок 1.1 – відношення задоволеності до кількості костувачів

1.6 Можливості Pug препроцесорів

Основне використання препроцессору Pug:

- Анотація з відступом : шаблон Pug відрізняється від більшості шаблонів. Він має власні граматичні особливості. Він позначений відступом. Використання цього методу робить код більш зрозумілим.
- Немає хвостових тегів : цей синтаксис не вимагає тегів хвоста, як /html.

Наприклад:

Код для HTML:

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

```
    <title>Головна сторінка<html>
  </head>
</html>
<html>
```

Код для PUG:

```
html
  head
    title Головна сторінка
```

- Написання тексту : напишіть текст у формі + пробіл + вміст після мітки. Наприклад, веб-сайт, за яким у коді нижче h1, є текстом назви заголовка.
- Додавання атрибутів : запишіть атрибути з () дужками після мітки.
- Метод імпорту : Самостійний файл: використовуйте включення + ім’я файлу, щоб імпортувати файл.

Наприклад:

```
body
  h1 мій сайт
  include includes/header.pug
```

- Метод передачі параметрів : зокрема, метод передачі параметрів між внутрішнім кодом і механізмом шаблонів pug.
- Спеціальні символи :

  "|", символи після | будуть виведені як є.

  ".", . означає, що всі символи наступного рівня будуть виведені такими, як вони є, і більше не будуть **розпізнаватися**. Це оновлена

версія |, яка реалізує пакетну обробку.

Завдяки таким символам можна додавати звичайний HTML у своєму форматі.

- Метод extend: для використання готого лаяуту
- Метод блок: для створеня флагів привьязки елементу з унікальним імекнем
- Також викоористовються цикли each (Таблиця 1.1), for (Таблиця 1.2),

Таблиця 1.1 – цикли each

ul
  each val, index in {1:'перший',2:'другий',3:'третій'}
  li= index + ': ' + val
	
	    1: перший
	    2: другий
	    3: третій

Таблиця 1.2 – цикли for

- var nElement = 5;
ul
  while nElement < 9
  li= nElement ++
	
	    5
	    6
	    7
	    8

- Міксіни: можна сказати, що міксини є дуже потужною функцією pug, методом, який дозволяє повторно використовувати цілий блок коду в

Pug (Таблиця 1.3).

Таблиця 1.3– mixin

//- визначити
navList mixin
  ul
  li Головна
  li Магазин
  li Товари
//- використовувати
+ navList
+ navList
	
	   - Головна
	   - Магазин
	   - Товари
	
	   - Головна
	   - Магазин
	   - Товари

Його також можна використовувати як функцію (Таблиця 1.4).

Таблиця 1.4– mixin як функція

mixin pet(name)
  li.pet= name
ul
  +pet('Кіт')
  +pet('Собака')
  +pet('Кінь')
	
	  <li class="pet">Кіт
	  <li class="pet">Собака
	  <li class="pet">Кінь

Звичайно, також можна змішувати блоки, що дуже зручно. Якщо треба повторно використати блок лише в поточному шаблоні, можна використовувати змішаний блок безпосередньо, не створюючи спадкування або включення шаблону (Таблиця 1.5).

Таблиця 1.5– mixin як функція з використанням блоку

mixin article(title)
  .article
  .article-wrapper
  h1= title
  if block
  block
  else
  p does not provide any content.
 	
+article('Привіт, світ')
 	
+article('Привіт, світ')
  p Це стаття, яку я написав випадково
	
	   Привіт, світ
	  Вміст не надано.
	
	   Привіт, світ
	  Це я
	  < p>Випадкова стаття

1.7 SASS препроцесор

Sass - це препроцесор CSS. Для CSS це те, що CoffeeScript для Javascript. Sass додає набір функцій до розмітки таблиць стилів, що знову робить написання стилів зручнішим.

Існує кілька способів компіляції Sass:

Оригінальний двійковий файл Ruby Sass. Встановіть його за допомогою gem install sass, і скомпілюйте його, запустивши sassc myfile.scss myfile.css.

Програма з графічним інтерфейсом, наприклад hammer.

libsass, який є надзвичайно швидким компілятором Sass, написаним на C. Ви також можна встановити libsass через NPM за допомогою node-sass (npm install node-sass).

тому Ruby Sass трохи повільний під час компіляції великих наборів

вихідних кодів. node-sass у своїй системі збірки, але libsass не відповідає 100% функцій Ruby Sass.

можна налаштувати їх на перегляд файлів scss, тому, коли ви їх редагуєте, вони будуть компілюватися автоматично

1.7.1 Яке відношення має .sass до .scss?

Коли Sass вперше вийшов, основний синтаксис помітно відрізнявся від CSS. Він використовував відступ замість дужок, не потребував крапки з комою та мав оператори скорочення.

У версії 3 Sass змінив його основний синтаксис на .scss. SCSS — це наднабір CSS, і в основному він написаний точно так само, але з усіма новими цікавими функціями Sass.

1.7.2 Навіщо використовувати Sass?

Sass полегшує написання підтримуваного CSS. Можна зробити більше, за меншу кількість коду, легше читати, за менший час.

1.7.3 Налаштування змінні

Sass переносить змінні в CSS.

Допустимі значення для змінних включають числа, рядки, кольори, null, списки та карти. Змінні в Sass визначаються за допомогою [$] символу.

```
$primaryColor: #eeacc
body {
        background-color : $primaryColor;
}
```

Sass має змінну область дії, якщо оголошувати змінну всередині селектора, вона буде в межах цього селектора.

Але що, якщо треба встановити змінну глобально з оголошення? Sass надає !global прапор, який приходить на допомогу.

Іншим корисним прапором, особливо під час написання міксинів, є !default

прапор. Це дозволяє переконатися, що для змінної існує значення за замовчуванням, якщо воно не надано. Якщо вказано значення, воно перезаписується.

Іншим корисним прапором, особливо під час написання міксинів, є !defaultпрапор. Це дозволяє нам переконатися, що для змінної існує значення за замовчуванням, якщо воно не надано. Якщо вказано значення, воно перезаписується.

Підтримувані оператори включають (Таблиця 1.6).

Таблиця 1.6– Підтримувані оператори

+	Додавання
-	Віднімання
/	Відділ
*	Множення
%	Модуль
==	Рівність
!=	Нерівність
+	Доповнення
-	Віднімання
/	Поділ
%	Модуль
*	Множення
==	Рівність
! =	Нерівність

Треба звернути увагу на дві потенційні «завади» математики Sass.

По-перше, оскільки /символ використовується у скорочених властивостях шрифтів CSS, як-от font: 14px/16px, якщо треба використовувати оператор поділу для не змінних значень, вам потрібно загорнути їх у дужки, наприклад: $fontRour:

(14px/16px)

По-друге, не можна змішувати одиниці значення:

$cont-width = 100% - 20px;

Наведений вище приклад не працюватиме. Замість цього для цього конкретного прикладу можна використовувати функцію css calc, оскільки вона повинна бути інтерпертована під час візуалізації.

1.7.4 Вкладення в SCSS

Однією з найбільш корисних, а також зловживаних функцій Sass є можливість вкладення декларацій.

Базове вкладення відноситься до можливості мати оголошення всередині оголошення. У звичайному CSS ми можемо написати:

```
.container {
  width: 100%;
  h1 {
  color: red;
  }
}
```

1.8 синтаксис мови JavaScript

Javascript є дуже популярною мовою сценаріїв серед розробників. Javascript - це мова програмування Інтернету. В останні роки Javascript став популярнішим, ніж будь-коли, завдяки багатьом функціям, які мають нові технології браузера. У браузерах, мобільних пристроях, настільних комп’ютерах, ігрових консолях використовується Javascript

Спочатку він був розроблений для взаємодії з кодом Html, коли він був дитиною. Також у той час існувала мова сценаріїв під назвою vb script. Vbscript був суперником Javascript, але Javascript вибив усі його альтернативи.

Javascript надає більшість сучасних мов програмування. Основний API Javascript дуже обмежений і може бути розширений за допомогою сторонніх бібліотек, таких як jQuery, Reactjs, AngularJS, Vue js тощо.

Javascript не зовсім об'єктно-орієнтований, але надає більшість функцій об'єктно-орієнтованого програмування. Тому ми можемо легко скористатися перевагами класів і успадкування.

Також він інтерпретується, що є протилежністю компіляції. Код Javascript не потрібно компілювати перед використанням. Це робить Javascript практичною мовою.

Зазвичай використовується для розробки програм на стороні клієнта. Що таке клієнтська сторона? сторона клієнта означає сторону споживача програми, тому зазвичай веб-додатки використовуються людьми, які просто переглядають їх у своєму браузері. Під час перегляду веб-сторінок коди JavaScript і програми також працюють у своїх браузерах .

У попередній частині ми називали Javascript стороною клієнта, але ми також називаємо його стороною сервера. Останнім часом Javascript використовується для програмування на стороні сервера. Програмування на стороні сервера Javascript стає популярним у NodeJS .

1.9 NodeJS

Кожна платформа має свою власну філософію: загальноприйнятий набір принципів і вказівок, спосіб виконання речей, який впливає на еволюцію платформи, а також те, як слід розробляти та проектувати програми. Деякі з цих принципів є похідними від самої технології, деякі активуються її екосистемою, деякі є тенденціями в суспільстві, а деякі інші еволюції різних ідеологій. Деякі фундаментальні принципи в Node.js походять безпосередньо від його творців - Райана Дала та інших, які зробили внесок у основну бібліотеку, деякі - від

харизматичних діячів у спільноті, а деякі успадковані від JavaScript. Походять від філософії UNIX або знаходяться під її впливом.

Жоден з цих принципів не нав’язується, вони завжди слідують здоровому глузду. Так чи інакше, вони виявилися дуже корисними, коли нам було потрібно джерело натхнення в процесі розробки програми.

Власна основна бібліотека Node.js побудована на кількох принципах. Одна з них — це мінімальний набір функцій, решта залишена для так званої платформи користувача (або простору користувача), екосистема модулів існує поза основною бібліотекою. Цей принцип має величезний вплив на культуру Node.js, оскільки дає спільноті свободу експериментувати та швидко використовувати широкий спектр рішень у межах користувацьких модулів, замість того, щоб бути обмеженим жорстко контрольованим і стабільним ядром, побудованим на Slowly розвиток рішень на основі бібліотек. Збереження мінімального набору основних функцій не тільки сприяє ремонтопридатності, але й має позитивний культурний вплив на еволюцію всієї екосистеми.

Node.js використовує концепцію модулів як фундаментальний спосіб структурувати програмний код. Це будівельний блок для створення програм і багаторазових бібліотек, також відомих як пакети (пакет також часто називають модулем, зазвичай він має один модуль як точку входу). У Node.js одним з найважливіших принципів є розробка невеликих модулів, що стосується не тільки розміру коду, а й розміру області.

Цей принцип випливає з філософії UNIX, зокрема з двох її принципів, а саме:

- «Маленьке – це красиво»
- «Змусьте програму робити одну справу добре»

Node.js виводить ці концепції на новий рівень. За допомогою NPM (офіційного менеджера пакетів) Node.js може допомогти вирішити проблему пекла залежностей, гарантуючи, що кожен встановлений пакет має власний набір

залежностей, так що програма залежить від багатьох пакунків без конфліктів. Насправді, цей підхід Node вимагає надзвичайно високого рівня повторного використання, тобто програма складається з великої кількості невеликих, але добре централізованих залежностей. Хоча це вважається непрактичним або навіть зовсім нездійсненним на інших платформах, це заохочується в Node.js. Як наслідок, часто можна побачити пакети NPM, які містять менше 100 рядків коду або надають лише один метод.

Крім очевидних переваг у багаторазовому використанні, можна також розглянути невеликі модулі:

- Легкий для розуміння та використання
- Простий у тестуванні та обслуговуванні
- Ідеально підходить для обміну з браузерами

Даючи кожному можливість спільно використовувати або повторно використовувати менші та централізовані модулі або навіть найменші шматки коду, виводить принцип «Не повторюй себе» (DRY) на новий рівень.

На додаток до невеликих розмірів і обсягу модулі Node.js часто мають функції, які надають лише мінімальний набір функціональних можливостей. Основна перевага цього полягає в тому, що воно підвищує зручність використання API, що означає, що використання API стає більш зрозумілим і менш схильним до неправильного використання. У більшості випадків користувачів компонента насправді цікавить лише дуже обмежений і цілеспрямований набір функціональних можливостей без необхідності розширювати функціональність або копати глибше.

На додаток до невеликих розмірів і обсягу модулі Node.js часто мають функції, які надають лише мінімальний набір функціональних можливостей. Основна перевага цього полягає в тому, що воно підвищує зручність використання API, що означає, що використання API стає більш зрозумілим і менш схильним до неправильного використання. У більшості випадків

користувачів компонента насправді цікавить лише дуже обмежений і цілеспрямований набір функціональних можливостей без необхідності розширювати функціональність або копати глибше.

Особливістю багатьох модулів Node.js є те, що вони створені для прямого використання, а не для розширень. Блокування внутрішньої частини модуля шляхом заборони будь-якої можливості розширення може здатися негнучким, але має перевагу, що зменшує випадки використання, спрощує впровадження, простоту обслуговування та покращує зручність використання .Принцип Keep It Simple простота - це найвища витонченість.

Дизайн повинен бути простим як в реалізації, так і в інтерфейсі. Що ще важливіше, реалізація простіше, ніж інтерфейс. Простота є найважливішим фактором в дизайні.

Хорошою практикою є розробка простого, а не ідеального, повністю функціонального програмного забезпечення. Є кілька причин: простіше впроваджувати; дозволяє швидше передавати з меншими ресурсами; легше адаптувати, підтримувати та розуміти. Внаслідок цих факторів було сприяно низці зусиль громади, що також сприяло розробці та вдосконаленню самого програмного забезпечення.

Хорошою практикою є розробка простого, а не ідеального, повністю функціонального програмного забезпечення. Є кілька причин: простіше впроваджувати; дозволяє швидше передавати з меншими ресурсами; легше адаптувати, підтримувати та розуміти. Внаслідок цих факторів було сприяно низці зусиль громади, що також сприяло розробці та вдосконаленню самого програмного забезпечення.

2. РОЗРОБКА ОТОЧЕННЯ

2.1 Розробка NPM оточення

NPM це менеджер пакунків, означає Node Package Manager. NPM потрібно в першу чергу NodeJS в ній вже буде існувати NPM його не потрібно встановлювати окремо.

NPM дозволяє розробникам JavaScript легко ділитися та повторно використовувати код, а також оновлювати код, яким ви надаєте спільний доступ.

NPM - це інструмент керування пакетами та розповсюдження для nodejs. Це полегшує розробникам javascript обмін кодом і фрагментами коду, а керувати спільним кодом за допомогою npm швидко й легко.

Офіційний сайт: https://docs.npmjs.com/

Завантаження Node.js: https://nodejs.org/en/download/

Встановіть Nodejs та оновіть npm. Якщо використуєтеся Windows або Mac, найкращий спосіб встановити nodejs — це зайти на офіційний веб-сайт, щоб завантажити інсталяційний пакет.

Після завершення інсталяції введіть у консолі node -v Якщо номер версії вузла повернуто правильно, установка пройшла успішно.

Після встановлення nodejs вже є npm, але оскільки швидкість оновлення nodejs нижча за npm, то загалом, щоб оновити npm до останньої версії, введіть таку команду:

npm install npm –g

Якщо це linux , вам може знадобитися запустити його з правами root через sudo.

Коли закінчите, запустіть npm -v, щоб побачити поточний номер версії npm.

2.2 Система управління пакетами

Менеджер пакетів або система управління пакетами являє собою набір програмних засобів, автоматизує процес установки, оновлення, налаштування і видалення комп'ютерних програм для комп'ютера та операційної системи послідовним чином.

Менеджер пакетів угод з пакетами, розподілу програмного забезпечення і даних в архівних файлах. Варіанти комплектації включають метадані, такі як ім'я програмного забезпечення, описи його призначення, номер версії, постачальник, контрольну суму, і список залежностей, необхідних для програмного забезпечення для правильного запуску. Після установки, метадані зберігаються в локальній базі даних пакета. Менеджери пакетів, як правило, підтримують базу даних залежностей програмного забезпечення та інформацію про версії програмного забезпечення для запобігання невідповідностей і відсутніх передумов. Вони тісно співпрацюють з репозиторіями, бінарними менеджерами сховищ і магазинами додатків

Менеджери пакетів призначені для усунення необхідності установки і оновлення власноруч. Це може бути особливо корисно для великих підприємств, чиї операційні системи засновані на Linux та інших Unix-подібних системах, як правило, складаються з сотень або навіть десятків тисяч різних програмних пакетів.

Програмний пакет являє собою архівний файл, який містить комп'ютерну програму, а також необхідні метадані для його розгортання. Комп'ютерна програма може бути в вихідному коді, який повинен бути складений і побудований першим. Метадані пакету включають в себе опис пакета, пакет версії і залежності (інші пакети, які повинні бути встановлені заздалегідь).

Менеджери пакетів створенні з завданням пошуку, установки, обслуговування або видалення пакетів програмного забезпечення по команді користувача. Типові функції системи управління пакетами включають в себе:

- робота з файлами архіваторів для вилучення архівів пакетів;
- забезпечення цілісності та автентичності пакета шляхом перевірки їх цифрових сертифікатів і контрольних сум;
- завантаження, установка або оновлення існуючого програмного забезпечення зі сховищ або магазину додатків;
- угруповання пакетів функції для зменшення непорозумінь;
- управління залежностей для забезпечення встановлення пакета з усіма пакетами, таким чином, уникнути «пекла залежностей».

Комп'ютерні системи, які покладаються на бібліотеки динамічного компонування, замість статичної бібліотеки зв'язків, спільно виконуваних бібліотек машинних команд через пакети та програми. У цих системах, складні відносини між різними пакетами вимагають різних версій бібліотек результатів або «пекло залежностей». У Microsoft Windows системах, це також називається «DLL пекло» при роботі з динамічно підключуванними бібліотеками. Добре управління пакетом має життєво важливе значення в цих системах. Рамкова система з OPENSTEP це була спроба вирішити цю проблему, дозволяючи існування кількох версій бібліотек, які будуть встановлені одночасно.

Системні адміністратори можуть встановлювати і підтримувати програмне забезпечення з використанням іншого, відмінного від програмного забезпечення інструмента управління пакетами. Наприклад, локальний адміністратор може завантажити неупакований вихідний код, скомпілювати його, і встановити його. Це може привести стан локальної системи до збою синхронізації зі станом менеджера пакетів в базі даних. Локальний адміністратор повинен вжити додаткові заходи, такі як управління вручну деякими залежності або інтегрувати зміни в менеджері пакетів.

Є інструменти, доступні для того, щоб локально скомпільовані пакети інтегровані з керуючими пакетами. Для розподілів на основі .deb і .rpm файлів, а також Slackware Linux, є CheckInstall, і для інструкцій на основі систем, такі як Gentoo Linux і гібридні системи, такі як Arch Linux, можна написати першу інструкцію яка потім забезпечує те, що пакет поміщається в локальну базу даних

пакетів.

Особливо клопітною роботою з оновленням програмного забезпечення є оновлення конфігураційних файлів. Оскільки менеджери пакетів на системах Unix виникають як розширення архівних файлів утиліт, зазвичай вони можуть тільки або перезаписати або зберегти файли конфігурації, а не застосовувати правила до них. Є винятки з цього правила, які зазвичай застосовуються до ядру конфігурації (яке, якщо порушення, буде надавати комп'ютер непридатний для використання після перезавантаження). Проблеми можуть виникнути, з форматом конфігураційних файлів змін; наприклад, якщо у старому файлі конфігурації не відкрито відключення нових опцій, які повинні бути відключені. Деякі менеджери пакетів, такі як Debian або DPKG, дозволяють налаштовувати конфігурації під час установки. В інших ситуаціях, бажано встановлювати пакети з настройками за замовчуванням, а потім перезаписувати цю конфігурацію, наприклад, в безголових установках з великою кількістю комп'ютерів. Цей вид попередньо сконфігурованої установки також підтримується DPKG.

Для того, щоб дати користувачам більше контролю над видами програмного забезпечення, вони дозволяють встановити на свою систему (а іноді з - за юридичних або цілодобових причин на стороні дистриб'юторів), програмне забезпечення часто завантажуються з декількох репозитаріїв.

Коли користувач взаємодіє з програмним забезпеченням управлінням пакетів, щоб викликати оновлення, прийнято представляти користувачеві список дій, які будуть виконуватися (як правило, список пакетів, які будуть оновлені, і, можливо, даючи старі і нові номери версій) і дозволяє користувачеві або прийняти оновлення навалом, або вибрати окремі пакети для оновлення. Багато менеджерів пакетів можуть бути налаштовані, щоб ніколи не оновити деякі пакети, або оновити їх тільки тоді, коли критичні уразливості або нестабільності знайдені в попередній версії, яка визначена пакувальником програмного забезпечення. Цей процес іноді називається версія піннінга.

2.3 Встановлення пакєтів Mac OS/Linux

Для систем MAC OS та Linux існує опенсорсний менеджер який допомагає швидко завантажити потрібні продукти які не може запропронувати Apple.
На офіцій сторінці homebrew заватажуємо через консольну команду яку він пропонує

Код для консолі:

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Після встановлення потрібно встановити NodeJs

Код для консолі:

```
brew install node
```

Встановиться остання стабільна версія Nodejs. Але є одна деталь, оскільни технологі застарівають виникає проблема із сумісністю пакетів. Різні версії пакетів працюють на різних версіях NodeJs
І дуже часто така проблема виникає оскільки на різних проєктах використовуються різні версії ноди
Для вирішення такоі проблеми знадобться встановити NVM
NVM дозволяє змінювати в окрємій дерікторії версію Node JS

2.4 Встановлення пакєтів Windows

Для системи Windows достатнь зайти на офіційний сайт NodeJS[2] та встановити потрібну версію

2.5 Ініціалізація

Далі ствлюємо директорію в якій буде наш проєкт.

Консольна команда: npm init

Він робить ініціаліацію проєкту та задасть пару питань де в дужкак будуть по дефолтні значення

- name
- version
- description
- main
- scripts
- repository
- author
- license
- bugs
- homepage

Після цього генерує package.json файл який повністю описує проєкт

2.6 Встановлення залежностей

Після ініціалізації ми можемо встановлювати потрібні пакети. Їх можна вставновлювати по церзі наприклад:

Консольна команда: npm i gulp

Або створити розділ в package.json [див. додаток А] з назвою devDependencies и виконати npm i.

NPM завантажить всі потрібні пакети для подальшої роботи і створить node_modules в якому з`явиться всі залежності

Для проэкту будуть потрібни такі пакети:

- "autoprefixer": "9.7.6",
- "browser-sync": "2.27.4",
- "filehound": "1.17.4",

- "gulp": "4.0.2",
- "gulp-clean": "0.4.0",
- "gulp-csso": "^4.0.1",
- "gulp-modernizr": "3.4.0",
- "gulp-plumber": "1.2.1",
- "gulp-postcss": "8.0.0",
- "gulp-pug": "^4.0.1",
- "gulp-rename": "^2.0.0",
- "gulp-sass": "4.1.0",
- "gulp-svg-sprite": "1.5.0",
- "path": "0.12.7",
- "through2": "^3.0.1"

Дуже важливо слідкувати за версіями оскильки вони можуть бути не сумісні з іншими або працювати не коректно.

2.7 Налаштування архитектури

Після встановлення необхідно продумати архітектуру папок та налаштувати збірку для цього кожний всановленний пакет виконує свою функцію

3 РОЗРОБКА КОМПОНЕНТІВ

3.1 Дизайн

Розробка компонентів неможлива без гарно спланованого дизайну (рисунок 3.1).

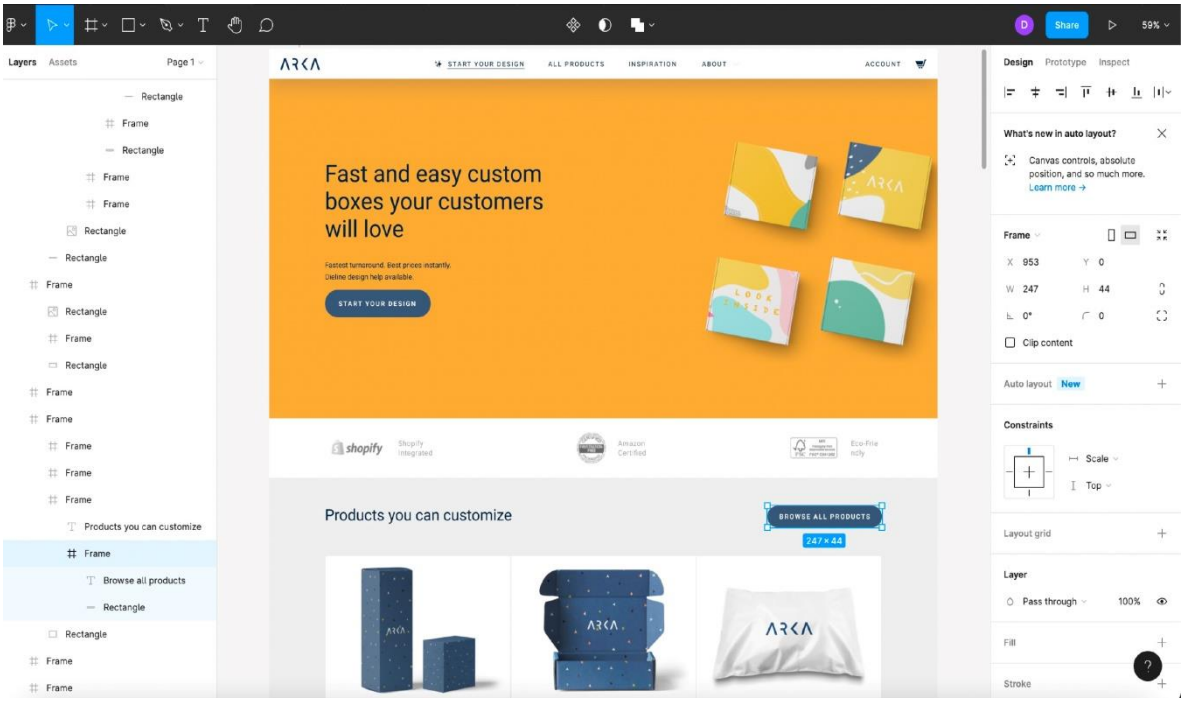


Рисунок 3.1 – Приклад дизайну

Дизайн в свою чергу створюється на основі компонентів їх ще називають UI Kit системою(Рисунок 3.2).

Рисунок 3.2 – Приклад дизайну UI Kit системи

Для створення такої системи треба зважувати на схожість між різними елементами дизайну. Їх можна поділити на такі частини:

- Навігація
- Кнопки
- Поля для заповнювання
- Текстові поля
- Чекбокси
- Радіо кнопки
- Текстова частина поділяється на заголовки, параграфи, системні меседжі та підзаголовки

Кожні з них мають спільні ознаки за потребою та трохи відмінні за виглядом. Наприклад кнопки можуть бути головні та другорядні. Зазвичай головні кнопки використовуються для підтвердження якихось дій а другорядні для відміни, кнопки червоного кольору зазвичай використовуються для видалення.Основні ознаки модальних вікон полягають в олному вигляді але вони можуть бути різні за розміром. У них є певні границі по висоті.

Кожен дизайн складається з верхньої частини тобто header

Основної частини тобто main

Та нижньої тобто footer

Також на сторінці присутня панель навігації вона може дублюватися у нижню частину сторінки. На сторінці можуть бути присутні сайт бари з лівої та правої сторони у них зазвичай додають контент який не зовсім відноситься до основної частини сторінки

3.2 Розробка автоматизації додавання SVG

SVG Це векторний тип зображень зазвичай використовується для створення іконок та векторної графіки. Основні його переваги це вміння масштабуватись залежно від розміру,малий розмір пам’яті, можливість змінювати частини елементу за допомогою css, та додавати анімацію до частин svg елементу. Використовує мову XML.

3.2.1 Автоматизація створення svg спрайтів.

Для автоматизації ми будемо додавати окремі SVG файли, Та найменувати їх структурно наприклад 16-arrow-down або для кольорових colored-16-arrow-down.

Напишемо скрипт де по назві елементу він буде створювати унікальний ID і змінювати тег <svg> у <symbol>. Тепер з усіх елементів створюється один SVG з усіма символами зі своїм ID. А далі створюємо mixin для SVG в ярому нам достатньо буде вказувати назву та розмір іконки.

Приклад коду х з поясненням :

```
let gulp = require('gulp'),
svgSprite = require('gulp-svg-sprite');
const { iconsMove } = require('./copy-icon');
const SpriteConfig = {
```

```
  mode: {
    symbol: { // Режим символів для створення SVG
      dest: '.', // Тека призначення
      sprite: './sprite.svg', // Ім'я спрайта
      example: false // Створити зразок сторінки
    }
  },
  svg: {
    xmlDeclaration: false, // видалити атрибут XML
    doctypeDeclaration: false // не включає декларацію !DOCTYPE
  }
};
svgToSprite =  async function svgToSprite() {
  return gulp.src('dev/icons/**/*.svg')
    .pipe(svgSprite(SpriteConfig))
    .pipe(gulp.dest('./public/icons/'))
    .pipe(gulp.dest('./dev/icon/'));
};
svgWatch = async function svgWatch() {
  let watcherSvg = gulp.watch('./dev/icons/**/*.svg');
  watcherSvg.on('change', function () {
    svgToSprite()
    iconsMove()
  });
};
```

3.2.2 Створення міксіну іконок

Для створення іконки нам потрібно для неї прописати параметри такі як назва іконки та розміри іконки. Якщо в наступному нам будуть потрібно додавати інші параметри моїх також додомо.

```
mixin icon(dataObject)
  if dataObject
    -
      var className = 'icon',
      modifiersArr = [],
      modifiers,
      setModifiers = function () {
      modifiersArr.push(className);
      if (dataObject.size)
        modifiersArr.push(className + '--size--' + dataObject.size);
        modifiers = modifiersArr.join(' ');
        return modifiers;
      };
    //<component: icon>
    span(class=setModifiers(), aria-hidden='true')
      if dataObject.name
        svg.icon__pic(style='fill:currentColor')
          use(xlink:href='#' + dataObject.name)
      else
        - throw new Error('Initial Error: {'+ className +'} Required property "name"')
```

3.3 Створення міксіну кнопок

Важливо пам’ятати різницю між кнопкою та силкою. Кнопки використовується для виконання якихось дій на сайті наприклад відправка форми, додавання якогось товару в корзину, налаштувати фільтр і таке інше.

А лінка для переходу на іншу сторінку але буває що візуально вона схожа

на кнопку тому треба передбачити цей момент.

За якими ознаками треба поділяти кнопки? Розмір, колір, його в свою чергу можна поділити на основний та додатковий також може бути прозорий і при наведеній він змінює свій колір, або з границею. Окрім цього в ньому може міститися текст або іконки якщо кнопка тільки з іконкою вона може мати фіксований розмір. Також кнопкам може бути різних станів таких як активний та неактивний.

Для створення міксин у кнопки нам треба зауважити що у кнопці можуть бути присутні іконки тому треба їх також додати. І на виході у нас буде міксін кнопки який в свою чергу використовує міксін іконки

```
include ../_mixins/icon


mixin button(options)
  - var buttonText = options.buttonText
  - var buttonBodyClass = options.buttonBodyClass || ""
  - var href = options.href
  - var ...

if href
  a(class="button" href=href role="button" aria-
label=ariaText)&attributes(buttonAttributes)
    span.button__body
      if icon
        span.button__icon(class=svgIconModes)
          +icon({
            name: icon,
          })
```

Для кнопки можуть бути додані додаткові атрібути, треба також додати можливість

```
else
  button(class="button" aria-label=ariaText)&attributes(buttonAttributes)
    span.button__body(class=buttonBodyClass)
      if icon
        span.button__icon(class=svgIconModes)
          +icon({
             name: icon,
          })

      if buttonText
        span.button__text!=buttonText
```

3.4 Створення міксіну для полів форми

Поля форми можуть бути 2 видів це input та textaria. В основному їх поділяють за розміром та наявністю іконок також вони можуть містити в собі Допоміжний текст при заповненні тобто placeholder, лейбл, і також мають стани як неактивний та помилковий коли значення не відповідає валідності

```
mixin input(data)
  - var iconBefore = data.iconBefore ? 'input--icon-left' : false
  - var iconAfter = data.iconAfter ? 'input--icon-right' : false
  .input-wrapper
    if iconBefore
      .input-wrapper__icon.input-wrapper__icon--align--left
        +icon({
          name: iconBefore.name,
          size: iconBefore.size
```

```
    })
  input.input(class=[iconBefore,iconAfter])
  if iconAfter
    .input-wrapper__icon.input-wrapper__icon--align--right
      +icon({
        name: iconAfter.name,
        size: iconAfter.size
      })
```

3.5 Створення модулів

Модулі це практично такі ж міксіни Але вони відмінні тим що зосереджують у собі групу різних міксінів. Це потрібно для створення однакових великих частин сторінок наприклад форми відправки даних в якій може міститися поля для емейл, імені, прізвища, номера телефону та якогось опису.

ВИСНОВОК

У роботі проведено аналіз існуючих технологій. Проведено аналіз існуючих рішень. Виявлено сильні сторони та недоліки кожного існуючого рішення, та на основі цього аналізу обрання за основу найбільш якісних існуючих рішень для розробки власної системи створення статичних стиорінок.

Проведено збір рекомендацій щодо створення інтерфейсу, який буде найбільш зрозумілий та зручний для стровення компонентів, що сприятиме в майбутньому швидку роботу.

Розроблена архітектура системи автоматизації створення єлементів там будування сторінок з можливістю її розширення, що дасть змогу швидко створювати нові шаблони до будь яких вимог дизайну.

Реалізовано технологіі для автомавтизації, що включає такі компоненти:

- Основні лаяути;
- Компоненти кнопок, навігації, полів форм, ;
- Створення збору іконок в один спрайт та компонент іконок.
- Створення міксінів та модулів для подальгошо перевикористання.

Така система в свою черну гарантує значне збільшення швидкості розробки. Можливість розширення компонентів та перевикористання в майбутньому.

ПЕРЕЛІК ПОСИЛАНЬ

1. PUG – препроцессор для HTML — URL: https://pugjs.org l (online;accessed: 13.04.2022).
2. SASS(SCSS) препроцесор для CSS. — URL: https://sass-lang.com/ (online;accessed: 26.04.2022).
3. JetBrains. WebStorm IDE. — URL: https://www.jetbrains.com/webstorm/ (online; accessed:13.04.2018).
4. Letter case. — URL: https://goo.gl/1AEW8n (online; accessed:7.05.2018).
5. Ltd Sublime HQ Pty. Sublime. — URL: https://www.sublimetext.com/ (online; accessed: 13.04.2018).
6. NPM https://www.npmjs.com/
7. Morelli Brandon. Different between ES6, ES8, ES 2017,ECMAScript. — URL: https://goo.gl/hNwTZt(online; accessed:24.04.2018).
8. Reid. JSLint. — URL: https://github.com/reid/node-jslint(online; accessed: 13.04.2018).
9. SAP. Web IDE. — URL: https://www.sap.com/developer/topics/sap-webide.html (online; accessed:10.04.2018).
10. SAP. Фреймворк SAPUI5. — URL: https://sapui5.hana.ondemand.com/ (online; accessed:10.04.2018).
11. SAP Fiori apps. — URL: https://www.sap.com/products/fiori.htmls (online; accessed: 26.04.2018).
12. Static program analysis. — URL: https://goo.gl/vSo9JG (online;accessed: 26.04.2018).
13. danielstjules. Jsinspect. — URL: github.com/danielstjules/jsinspect (online; accessed: 10.04.2018).
14. jonlabelle. JsPrettier. — URL: github.com/jonlabelle/SublimeJsPrettier (online; accessed:10.04.2018).

15. Алгоритм обхода в глубину. — URL: http://e-maxx.ru/algo/dfs(online; accessed: 8.05.2018).
16. Алгоритм обхода в ширину. — URL: http://e-maxx.ru/algo/bfs(online; accessed: 8.05.2018).
17. Дэвид Флэнаган. 13.4.1. Букмарклеты // JavaScript. Подробное руководство = JavaScript. The Definite Guide / Перевод А. Киселева. — 5-е изд. — СПб.: «Символ-Плюс», 2008. — С. 267. — ISBN 5-93286-103-7.
18. User javascript (англ.). Opera tutorial. Opera Software. Дата обращения 27 ноября 2009. Архивировано 22 августа 2011 года.
19. UJS Manager for Opera makes it easy to manage userscripts (англ.) (25 November 2009). Дата обращения 27 ноября 2009. Архивировано 22 августа 2011 года.
20. User Scripts (англ.). The Chromium Projects. Дата обращения 4 июня 2010. Архивировано 22 августа 2011 года.
21. Java™ SE 6 Release Notes (англ.). Sun Microsystems, Inc.. — Анализ исходного кода Mozilla Firefox. Дата обращения 19 ноября 2009. Архивировано 22 августа 2011 года.

Додаток А

Державний університет телекомунікацій

Навчально-науковий інститут ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

РОЗРОБКА СИСТЕМ АВТОМАТИЗАЦІЇ ПОБУДОВИ WEB СТОРІНОК З ВИКОРИСТАННЯМ PUG ТА SASS ПРЕПРОЦЕСОРІВ

Виконав студент 4 курсу
Групи ПД-41 Жужков Д.І.
Керівник роботи: Старший викладач Бондарчук А.П.

Київ – 2022

Мета, об’єкт та предмет дослідження

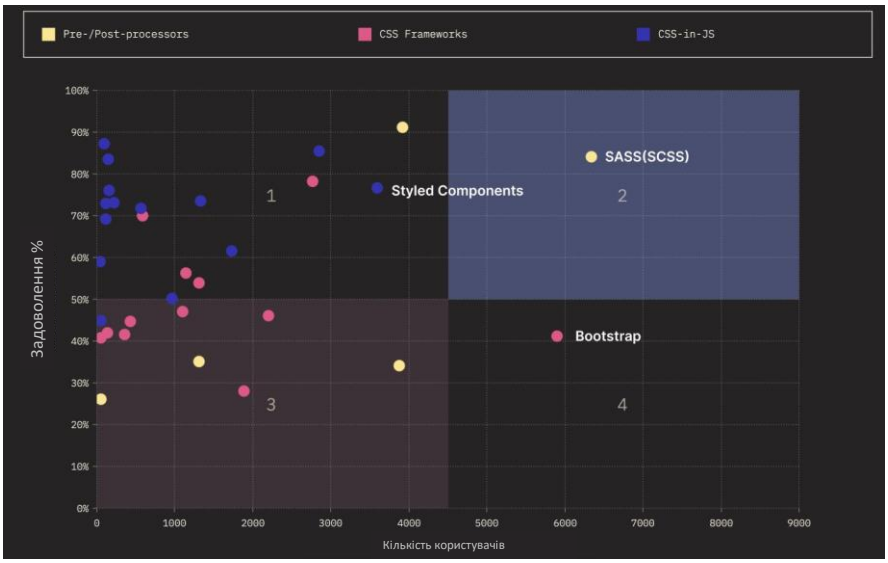
Об’єкт дослідження: Аналіз актуальних технологій створення автоматизації для можливості перевикористання в подальшому.

Предмет дослідження: компоненти Pug та SASS(SCSS).

Мета роботи: розробка компонентів для системи автоматизації для створення WEB сторінок.

Методи дослідження: аналіз дизайн системи, на основі якого проєктуються компоненти. Створення компонентів з використанням міксінів для подальшого перевикористання.

Аналіз технологій



4

Основні препроцессори

Для HTML

Для Css

5

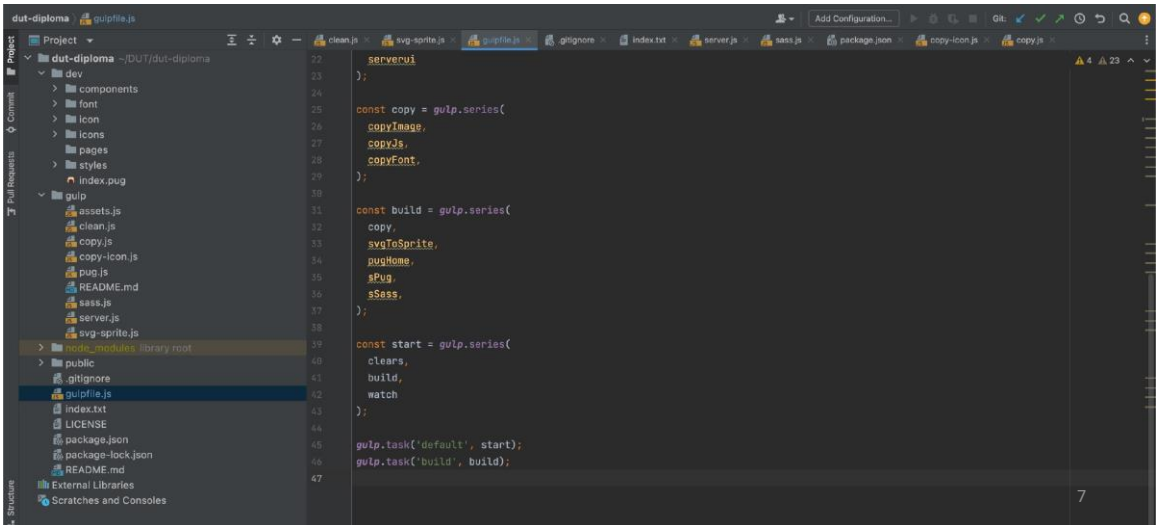
Розробка оточення

Вибранні основні пакети для оточення

- gulp
- browser-sync
- path
- gulp-clean
- gulp-sass
- autoprefixer
- gulp-pug

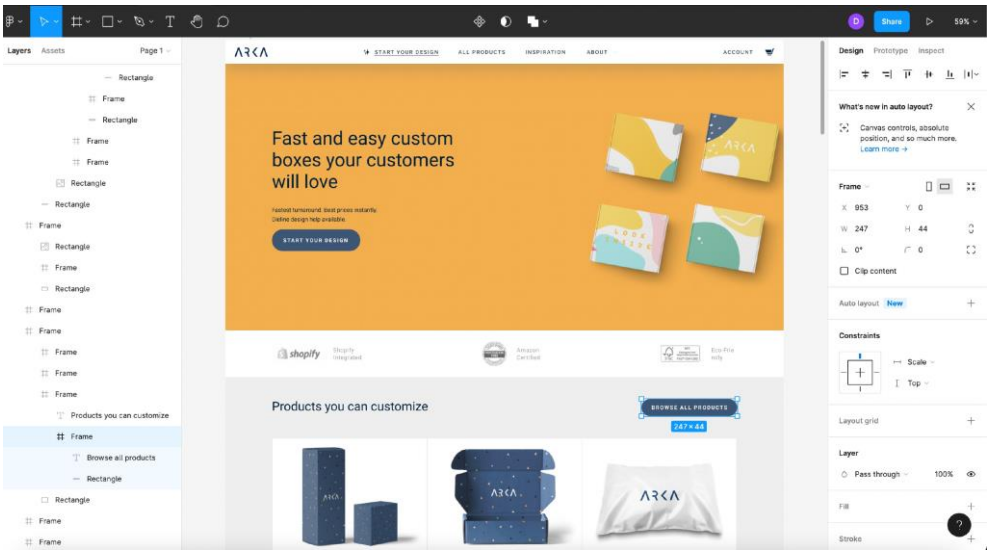
6

Розробка оточення



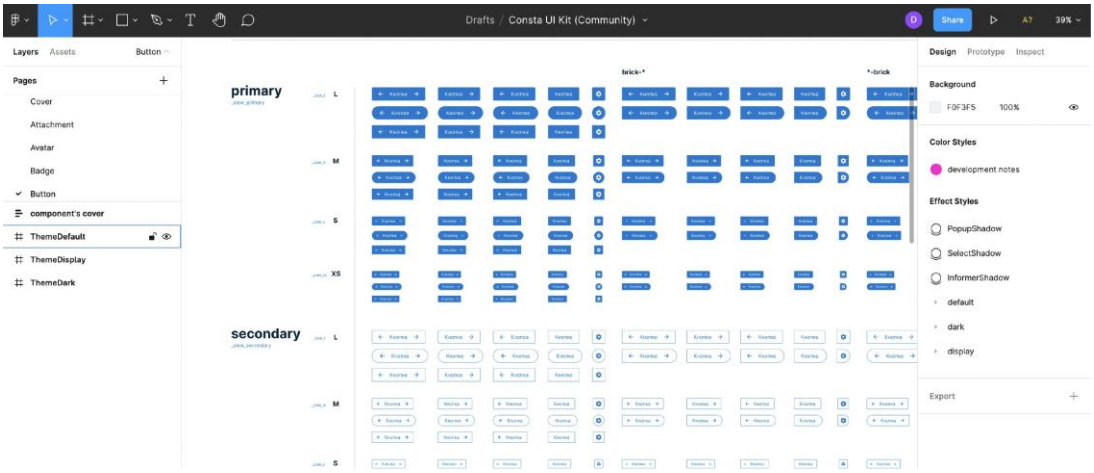
7

Дизайн



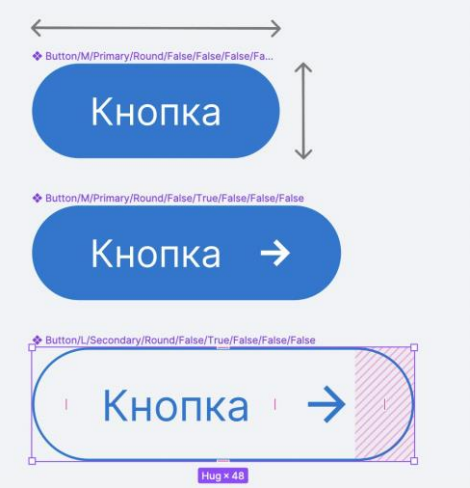
8

UI KIT



9

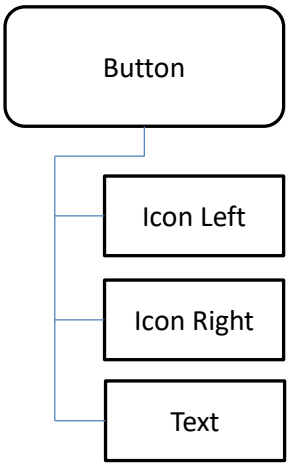
Міксін кнопки



- Тип
- Розміри
- Відступи
- Наявність іконки
- Бордери
- Колір фону
- Колір тексту
- Розмір шрифту
- Жирність шрифту
- Вирівнювання

10

Створення міксіну кнопки



Кнопка:

- Текст
- Додадковий текст
- Іконка зліва
- Іконка справа
  - ім’я
  - розмір
- Класс модифікатор

Класси:

primary | secondary…
cm | sm | md…
is-active | is-disable…
limit-width | full-width…

11

Створення міксіну поля

Знайти товар

Пошук

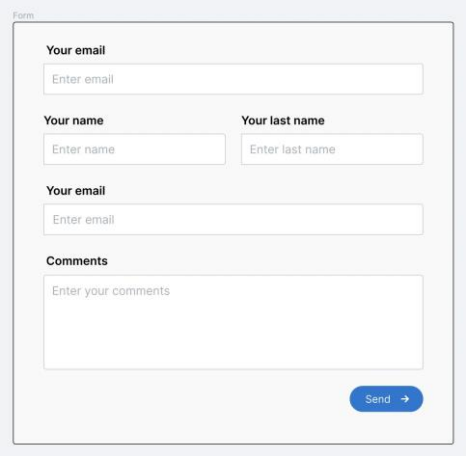
Поле:

- Текст - placeholder
- Лейбл
- Іконка зліва
  - ім’я
  - розмір
- Класс модифікатор

Класси:
cm | sm | md…
is-error | is-disable…
limit-width | full-width…

12

Модулі



Модуль Форми у Pug

```
+form({
   type: “send”
})
   +input ({
      label: “Your email”,
      placeholder: “Enter email”
   })
   +input ({
      label: “Your name”,
      placeholder: “Enter name”
   })
   +input( {
      label: “Your last name”,
      placeholder: “Enter  last name”
   })
    ...
```

13

ВИСНОВКИ

- Проаналізовано різні актуальні на зараз технології.
- Визначено що технології препроцессорів все переважають в розвитку та зручності використання.
- Запропоновано архітектуру побудови проєкту та розроблено систему автоматизації для побудови веб сторінок використовючи готові компоненти.

14

Дякую за увагу!

15