**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп’ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

**Курсова робота**

за спеціальністю

121 Інженерія програмного забезпечення: програмна інженерія

на тему:

**РОЗРОБКА ПРИКЛАДНИХ ПРОГРАМ ТА WEB-СЕРВІСІВ**

Виконала студентка 3-го курсу

Новік Марія Володимирівна \_\_\_\_\_\_\_\_\_\_\_

(підпис)

Науковий керівник:

доцент, кандидат фіз.-мат. наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Катеринич Лариса Олександрівна (підпис)

Засвідчую, що в цій роботі

немає запозичень з праць інших авторів

без відповідних посилань.

Студент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Київ – 2019

# **РЕФЕРАТ**

*Обсяг роботи 37 сторiнок, 11 використаних джерел, 21 рисунок. Ключовi слова: CSS, препроцесори, постпроцесори, веб-розробка, PostCSS, автоматизація.*

Об’єктом роботи є процес верстки з використанням спеціальної мови стилізації HTML документів – CSS. Предметом роботи є програмний засіб, а саме, пакет плагінів PostCSS, який надає реалізацію необхідним ідеям і функціям препроцесора для автоматизації руттинних дій з CSS.

Метою роботи курсової роботи було поставлено виокремити особливості препроцесорів та проаналізувати запити розробників у цій галузі, з’ясувати переваги і недоліки кожної технології і на основі цього дослідження здійснити програмну реалізацію, де буде використано і продемонстровано особливості обраної технології.

Інструменти розроблення: інтегроване середовище розробки від компанії Jet Brains - WebStorm, мова програмування JavaScript, PostCSS – програма, що допомагає удосконалити механічні операції над CSS-кодом шляхом розширень написаних на мові Javascript, npm - менеджер пакетів для мови програмування JavaScript.

Результати роботи: у результаті виконання курсової роботи, було досліджено та розглянуто CSS-препроцесори (SASS LESS, Stylus) та PostCSS, виокремлено їхні особливості та виконано порівняння з різноманітними концепціями та підходами до функціоналу препроцесорів. В ході написання курсової роботи було підготовлено практичну частину, що являє собою пакет плагінів PostCSS, який демонструє та пояснює виокремлені в теоретичній частині особливості і надає реалізацію необхідним ідеям і функціям препроцесора.

Програмний продукт може бути використаний при розробці власних веб-орієнтованих систем, в яких передбачена робота з CSS для автоматизації рутинних операцій роботи з каскадними таблицями стилів.

**Зміст**

[**РЕФЕРАТ** 2](#_Toc9797505)

[**СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ** 4](#_Toc9797506)

[**ВСТУП** 5](#_Toc9797507)

[**РОЗДІЛ 1. ОГЛЯД НАЙПОПУЛЯРНІШИХ CSS-ПРЕПРОЦЕСОРІВ** 7](#_Toc9797508)

[1.1 Що таке CSS-препроцесори? Історія та причини створення, основні властивості 7](#_Toc9797509)

[1.2 Огляд основних та найпопулярніших CSS-препроцесорів (LESS, SASS, Stylus) 11](#_Toc9797510)

[1.3 Огляд PostCSS 19](#_Toc9797511)

[1.4 Проблеми найпопулярніших CSS-препроцесорів та можливі методи їх вирішення 22](#_Toc9797512)

[**РОЗДІЛ 2. ТЕХНОЛОГІЯ РОЗРОБКИ ТА СТРУКТУРА ПРОЕКТУ** 25](#_Toc9797513)

[2.1. Основна ідея і мета проекту 25](#_Toc9797514)

[2.2 Підбір технологій, встановлення й налаштування PostCSS 27](#_Toc9797515)

[2.3 Огляд створених програм 31](#_Toc9797516)

[**ВИСНОВКИ** 34](#_Toc9797517)

[**ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ** 35](#_Toc9797518)

[**ДОДАТОК А** Приклад роботи плагінів 37](#_Toc9797519)

# **СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ**

Від англ. – розшифрування аббревіатури на англійській мові;

CSS – (від англ. Cascading Style Sheets), каскадні таблиці стилів;

IDE – (від англ. Integrated Design Environment), інтегроване середовище розробки;

HTML – (від англ. Hypertext Markup Language), стандартна мова розмітки для створення веб-сторінок і веб-додатків;

PostCSS – програма, що автоматизує рутинні операції CSS за допомогою плагінів, написаних на мові Javascript;

# **ВСТУП**

**Оцінка сучасного стану об’єкта розробки.** З плином часу і розвитком IT-індустрії офлайн додатки вже не знаходять місця у повсякденному вжитку користувача. Водночас, швидкими темпами зростає кількість веб-застосувань. Їхня архітектура значно ускладняються з кожним роком і вимагає нових рішень та технологій. Якщо ми говоримо про JavaScript, то не можна не згадати Angular, JQuery, React, про HTML – Bootstrap, Sprout, Kendo. В технології CSS розробникам на допомогу прийшли препроцесори (SASS, LESS, Stylus) та постпроцесори (Pleeease, Bless, CSSNext) та процесори такі як PostCSS. Саме препроцесори та PostCSS буде розглянуто в цій курсовій роботі.

**Актуальність роботи та підстави для її виконання**. Варто зазначити, що використання тієї чи іншої технології залишається не лише питанням смаку, а й питанням недостатньої підтримки певних функцій в одному препроцесорі і навпаки – занадто сильна або строга їх підтримка в інших (наприклад, синтаксис, змінні або домішки). Також невивченим залишається питання розвитку препроцесорів, що має відкритий характер і доповнюється користувачами, що означає також і велику кількість помилок та невідповідностей. Недостатньо обширна документація у більшості випадків породжує багато проблем ще на етапі розробки.

**Мета й завдання роботи.** Враховуючи тенденції та розвиток різноманітних засобів для видозміни CSS-файлів за мету даної курсової роботи було поставлено виокремити особливості препроцесорів та проаналізувати запити розробників у цій галузі, виявити недосліджені досі помилки, що зв’язані з реалізацією, надати власний спосіб їх вирішення, з’ясувати переваги і недоліки кожної технології і на основі цього дослідження здійснити програмну реалізацію, де буде використано і продемонстровано особливості обраної технології.

**Об’єкт, методи й засоби дослідження та розроблення**. В результаті виконання курсової роботи, було досліджено та розглянуто CSS-препроцесори (SASS LESS, Stylus) та PostCSS, виокремлено їхні особливості та виконано порівняння з різноманітними концепціями та підходами до функціоналу препроцесорів. В ході написання курсової роботи було підготовлено практичну частину, що являє собою пакет плагінів PostCSS, який демонструє та пояснює виокремлені в теоретичній частині особливості і надає реалізацію необхідним, на думку автора, ідеям і функціям препроцесора.

**Можливі сфери застосування.** Використання програмної реалізації дозволить розширити базу знань про PostCSS, отримати додатковий ресурс для аналізу, доповнить сховище плагінів для автоматизації рутинних операцій CSS. Сфери застосування можуть бути як практичними, так і науковими, наприклад, як для використання на власних проектах для оптимізації рутинних операцій при роботі з CSS, так і для розробки власних препроцесорів.

# **РОЗДІЛ 1. ОГЛЯД НАЙПОПУЛЯРНІШИХ CSS-ПРЕПРОЦЕСОРІВ**

## **1.1 Що таке CSS-препроцесори? Історія та причини створення, основні властивості**

Ми звикли сприймати в загальному Frontend розробку, як поєднання трьох складових: «HTML для створення та структурування документу, CSS для надання цьому документу стилів та Javascript для наділення документа певними функціями та властивостями» [8, 12]. Однак, «вимоги до CSS настільки виросли за останній час, що не існує жодної людини, що здатна тримати всі його функції в голові» [11, 26]. Часто використовуючи саме CSS, можна помітити як одні й ті самі властивості для різних елементів розробник змушений копіювати, а отже й дублювати відносно схожий код декілька разів. На великих проектах, що очевидно, це призводить до великої затрати часу і водночас є процесом, який можна автоматизувати і спростити. Це й стало основною причиною створення CSS-препроцесора. «Препроцесор конвертує дані CSS з одного формату в інший. Препроцесори допомагають нам прибираючи неефективні функції і роблять процес створення і побудови веб-сайтів більш гнучким і простим» [2, 1]. Препроцесор робить керування великою кількістю складних таблиць каскадних стилів (CSS) простішим, завдяки тому, що використовує такі невластиві CSS речі як змінні, функції і домішки (mixins). Саме тому код стає простішим для сприйняття людини, а отже значно знижує ймовірність виникнення механічних (спричинених людиною) помилок у розробці. Наразі існує дуже багато популярних препроцесорів. Основні з них: Sass, Less, Stylus, PostCSS та багато інших. Деякі із них буде детально розглянуто в наступних розділах.

Поширення препроцесорів не можна недооцінювати: за даними 2012 року в «онлайн опитуванні, де взяли участь більше 13000 респондентів-розробників, було встановлено, що у фірмах, які фокусуються на Web-розробці, кількість працівників, які використовують CSS-препроцесори сягає 54%» [OASics-SLATE,2]. Не можна заперечувати й той факт, що це число росте з кожним днем.

Будь-який код написаний за допомогою препроцесора опісля компілюється в звичний CSS-код. І, звісно ж, немало на популярність певного препроцесора впливає і мова, на якій реалізовано компілятор. У тому ж опитуванні 2012-го року провідну позицію зайняв препроцесор LESS (23%), оскільки його компілятор досить легко підключити відразу у HTML файл, бо написаний він за допомогою Javascript. Саме тому наступні за популярністю препроцесори Sass та Stylus отримали лише 5% та 3% відповідно – для компіляції їхнього коду в звичайний CSS-файл необхідно встановити Ruby, завантажити Ruby-gem і компілювати файли через консоль (наразі дані препроцесори значно розвинули процес компіляції, однак від вимоги встановлення мови, не надто популярної для роботи з версткою та й frontend-розробкою загалом, не відмовилися). На рисунку 1.1.1 зображена діаграма, що відображує наведену вище статистику:



Рис. 1.1.1 – Діаграма популярності певних CSS-препроцесорів та кількості користувачів

Розглянемо декілька загальних особливостей і властивостей, які пропонує більшість препроцесорів:

* Змінні

Одна з основних функцій препроцесорів, яка надає розробнику змогу, наприклад, визначити змінну основного кольору, який опісля часто використовується на сторінці. Зручно це і з тієї точки зору, що можна в будь-який момент змінити кольорову гамму без затрат часу на перейменування кожного рядка.

Використовувати функції можна не лише в описі властивостей, а і в селекторах та в самих назвах властивостей

* Розширення

Наслідування необхідне в CSS тоді, коли ми хочемо використати код повторно, однак надати певним елементам незначної відмінності.

* Домішки

Домішки можна порівняти з дуже простим конструктором в ООП. На відміну від розширень, до них можна додати ще аргументи, що розширяє можливості їх використання. Дуже часто домішки використовують для властивостей кросбраузерності.

* Функції

Різні препроцесори мають різноманітні вбудовані функції, багато з яких є спільними для всіх з них. Найчастіше це функції градієнтів, затемнення/освітлення елементів, обробка стрічок та набір певних математичних функцій (наприклад, random).

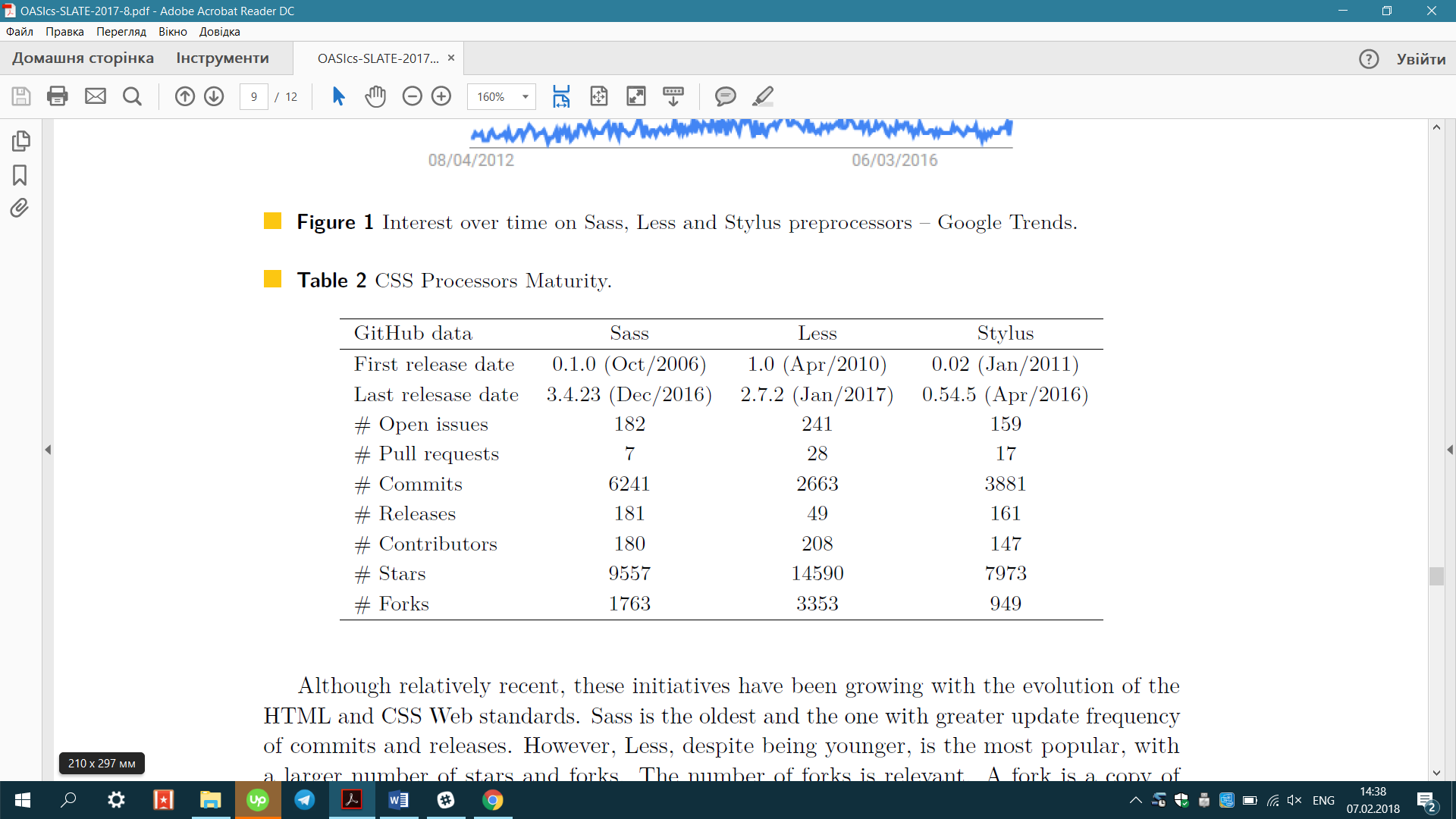
Також можна назначити власну функцію. Для їхнього виклику використову­­­ються так звані «**role directives**» - схожі на домішки, однак повертають не набір властивостей, а значення через @return.

* Цикли і умови

«Деякі з препроцесорів підтримують також цикли та умовні структури (if, else, each, while, for). Однак використовуються вони рідко, оскільки зазвичай створюють елементи, які можна згенерувати за допомогою простіших операцій»[7].

Важко визначити який із наведених вище препроцесорів є найпопулярнішим. Однак декілька наведених статистичних даних демонструють популярність і тенденції розвитку різних препроцесорів.

В таблиці 1.1.1 зображене порівняння 3-ох найвідоміших препроцесорів згідно з їх статистикою на веб-сайті Github.com. Як бачимо, незважаючи на те, що Sass є найстаршим з них і отримав найбільшу кількість комітів (commits) та релізів (releases), Less, який у всіх цих значеннях йому програє, все ж отримав майже вдвічі більшу кількість вподобань від юзерів (stars).



Таблиця 1.1.1 – Порівняння трьох препроцесорів за статистикою на веб-сайті Github.com

Однак, якщо поглянути на статистику по кількості запитів в пошуковій системі google.com, можемо зрозуміти протилежне (детальніше на рисунку 1.1.2). Варто зазначити, що в останні роки популярність Sass різко зросла, а колись популярного Less - відповідно впала. Зумовлюється це тим, що оптимізація компілятора для Sass призвела до того, що він обробляє файли втричі швидше.

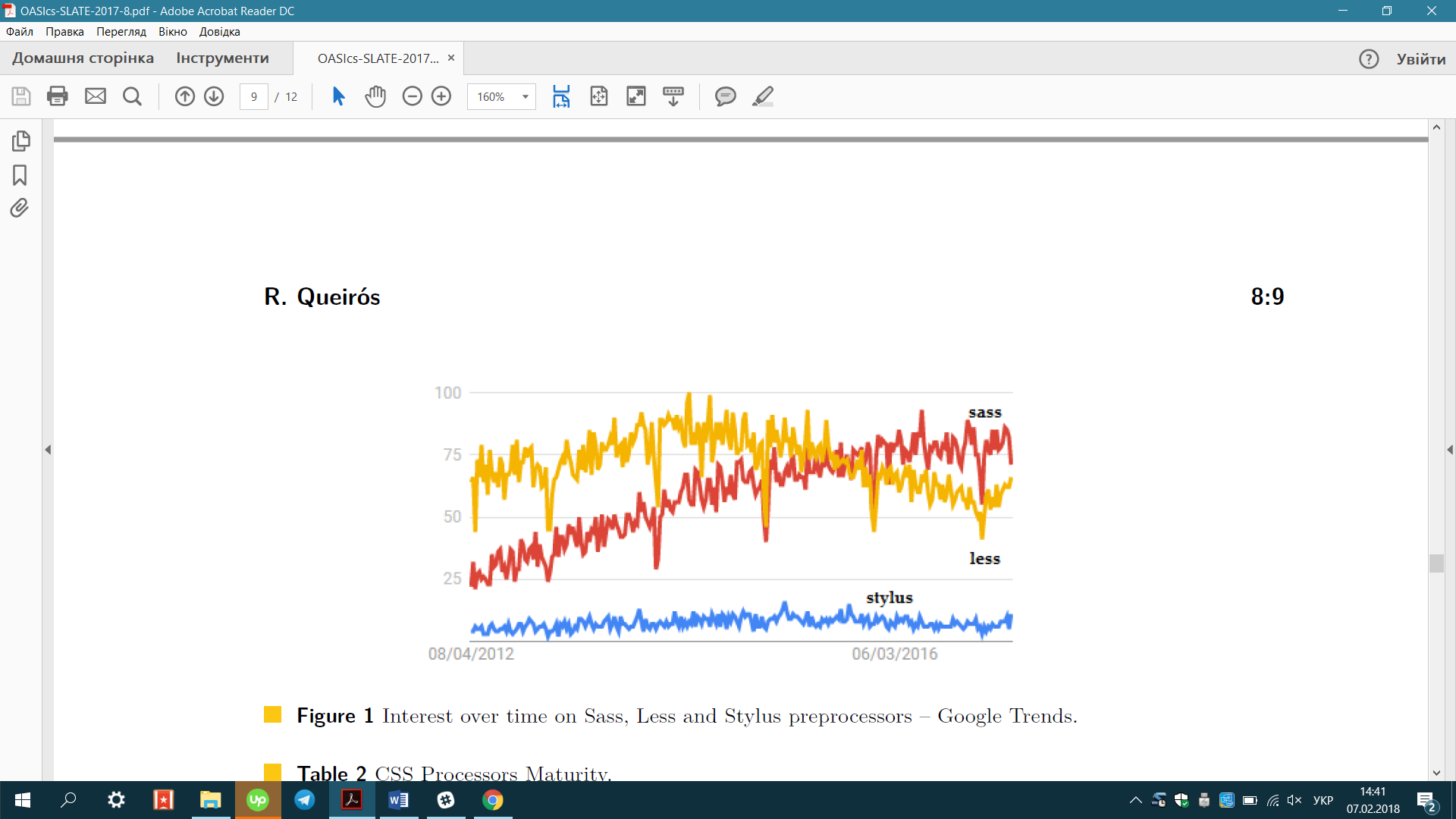


Рисунок 1.1.2 – Порівняння кількості запитів в пошуковій системі google.com

Така статистика спричиняється також і певними відмінностями в функціоналі. Звісно кожен з них має певні переваги та недоліки, які будуть детально розглянуті в пункті третьому цього розділу.

## **1.2 Огляд основних та найпопулярніших CSS-препроцесорів (LESS, SASS, Stylus)**

В даному розділі розглянемо найпопулярніші CSS-препроцесори і спробуємо виокремити основні переваги й, можливо, недоліки деяких з них.

Розглянемо основі особливості цих препроцесорів і реалізації загальних властивостей:

* Синтаксис;
* Розширення;
* Змінні;
* Наявність плагінів та бібліотек;

Та багато іншого

SASS

Sass – скриптова мова, що інтерпретується (або компілюється, в залежності від використання) в CSS-файли. Був створений Хемптоном Кетліном і Наталі Вайзенбаус. Після початкових версій приступили до розробки розширення до Sass - SassScript – скриптова мова, що використовується в нових версіях Sass.

«В Sass виділяють два основних синтаксиси:

* Indentation syntax (керується відступами для відокремлення блоків коду, а новий рядок сприймає за початок нового правила)
* SCSS Syntax (Ближчий до CSS – використовує дужки для відокремлення блоків коду і крапку з комою для нових правил). » [8,20].

Ці два синтаксиси утворюють, відповідно, в результаті файли двох різних

розширень - .sass і .scss

Існує вбудована інтеграція з Bootstrap, Compass, Toolkit.

Компанії, які використовують Sass: Kickstarter, Weebly, Airbnb, Pandora та багато інших.

LESS

Less – динамічна мова стилів, що компілюється в CSS. «Створена під впливом Sass розроблена Алексіс Селліер в 2009-ому році» [9, 41]. Натомість, SCSS синтакс був створений під впливом зростаючої популярності Less. Перші версії були створені за допомогою Ruby, однак пізніше була повністю переписана за допомогою JavaScript. Основна відмінність Less від інших препроцесорів – його код можна компілювати on-the-go через less.js.

Не має жодних офіційних вбудованих інтеграцій.

Компанії, що використовують Less: Weebly, GoSquared, Geocodio та багато інших.

Stylus

Stylus – динамічна мова стилів, що визнана четвертою найбільш популярною в світі. Була створена найвідомішим анонімним розробником світу - TJ Holowaychuk (розробник, що створив Node.js та LUNA). Препроцесор написаний на JADE та Node.js

Не має жодних офіційних вбудованих інтеграцій.

Компанії, що використовують Stylus: Яндекс, Coursera,Ratio, Vtex та інші.

Порівняємо синтаксис усіх трьох препроцесорів:

* Sass і Less підтримують звичайний синтаксис, схожий на CSS

Рис. 1.1.3 – Приклад синтаксису Less та Sass

У Sass також досі підтримується ідентаційний синтаксис, однак велика ймовірність того, що ця функція стане deprecated в наступних версіях. Тим не менш, препроцесор дозволяє відділяти елементи табуляцією.



Рис. 1.1.4 – Приклад використання ідентаційного синтаксису Sass

* Stylus дозволяє використовувати усі з наявних синтаксисів, при чому необов’язково використовувати лише якийсь один, у одному файлі можна змішувати синтаксиси, препроцесор правильно їх обробить.

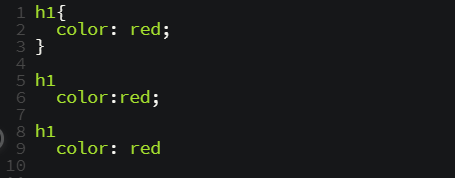


Рис. 1.1.5 – Приклад використання усіх можливих видів синтаксису в Stylus

Якщо порівнювати способи оголошення змінних у даних препроцесорів, то отримаємо такі варіанти:

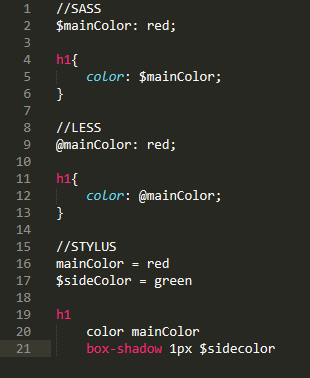


Рис. 1.1.6 – Порівняння методу оголошення змінних в трьох препроцесорах

SASS пропонує нам нотацію через символ ‘$’, LESS через ‘@’, Stylus ж натомість не вимагає від нас жодних ідентифікаторів змінних, для розпізнавання змінної йому важлива наявність її значення через знак ‘=’.

Домішки також широко розповсюджені у розробників у використанні. Ці функції дозволяють легко та просто уніфікувати і видозмінювати код, що використовується в багатьох елементах і водночас дублікується. Розглянемо реалізацію цієї корисної функції у препроцесорах:

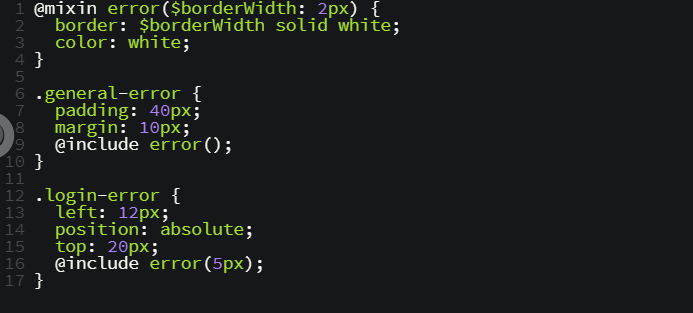


Рис. 1.1.7 – Приклад використання домішки в Sass

Як бачимо, реалізовано домішку для повідомлення про помилку. В функцію передається параметром $borderWidth – ширина контура. Значення за замовчуванням - 2 px, однак ми завжди можемо змінити її, зазначивши інше значення в параметрах під час виклику функції. В Sass використовуємо для виклику функції “@include”. Сама ж домішка вимагає нотації “@mixin”.

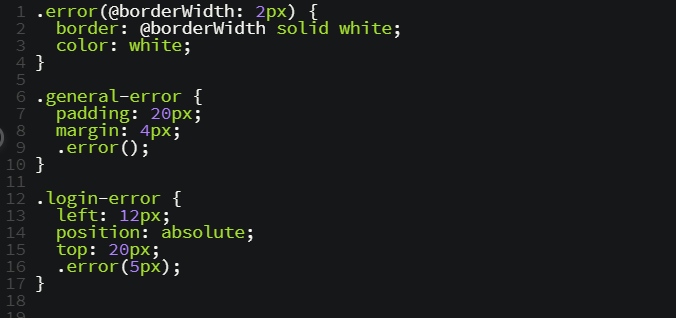


Рис. 1.1.8 – Приклад використання домішки в Less

Домішкою в Less може бути будь-який елемент (class, id, tag) і відповідно будь-який елемент може передавати в собі певні параметри. Не вимагається використання жодних додаткових службових слів, тощо.

Реалізація цього функціоналу в Stylus є ідентичною до Less.

Наслідування блоків коду - це те, для чого більшість розробників використовує препроцесор. Це зручно, дозволяє зменшити об’єм написаного коду і відповідно час на редагування необхідних змін.

Sass, Less, Stylus дуже схоже використовують їх за синтаксисом, однак по-різному компілюють їх в pure CSS.

Розглянемо реалізацію унаслідування певного блоку коду в Sass і Stylus на рисунку 1.1.9.

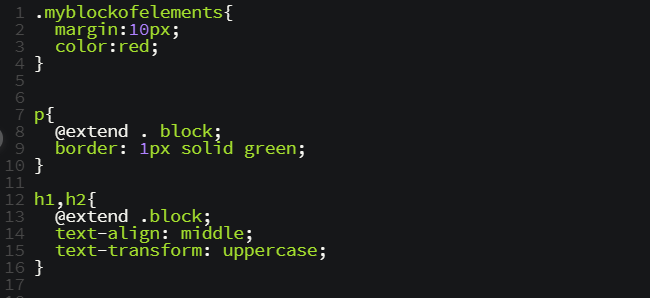


Рис. 1.1.9 – Приклад використання наслідування в Sass та Stylus

Відкомпільований в CSS-файл код цього фрагменту виглядатиме наступним чином (див. рис.1.1.10):

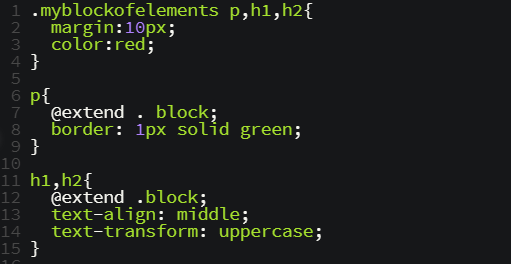


Рис. 1.1.10 – Код Sass і Stylus відкомпільований в CSS

З Less ситуація складається дещо інакше, він справді не підтримує наслідування так, як це роблять Sass і Stylus.

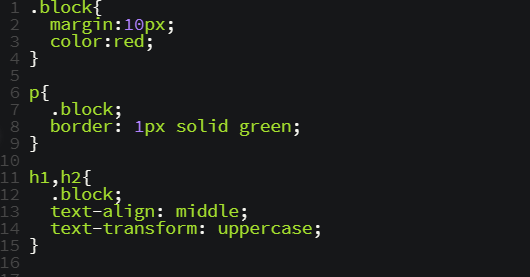


Рис. 1.1.11 – Реалізації наслідування в Less

Замість додавання декількох селекторів до одного набору функцій, він реалізовує домішку без аргументів і імпортує стилі до всіх селекторів окремо. Можна віднести до мінусів те, що це значно збільшує кількість рядків коду. Результат відкомпільованого коду дивіться на рис. 1.1.12

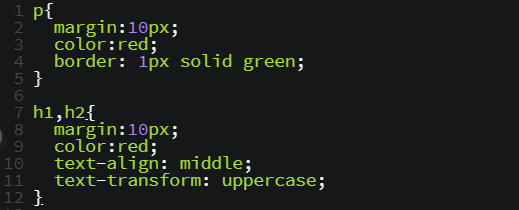


Рис. 1.1.12 – Код Less відкомпільований в CSS

Підсумовуючи, варто сказати, що важко визначити який з даних препроссорів є явним лідером за усіма параметрами: кожен з них реалізовує функціонал дуже різноманітно. Stylus надає розробнику найбільшу свободу дій, однак це зовсім не означає корисність. SASS надає найбільш строгі рамки щодо використання синтаксису, LESS залишається посеред цих двох препроцесорів. Основну перевагу звісно надають SASS та LESS переважна кількість розробників, що з ними працюють, однією з причин є постійне оновлення і доповнення цих препроцесорів, що не можна сказати про Stylus. В першому пункті ми розглядали кількість оновлень та доповнень на ресурсі Github по кожному з цих аспектів.

## **1.3 Огляд PostCSS**

Пройшовши довгий шлях від початку заснування такого поняття як препроцесори до набуття ними широкого вжитку, методи реалізації даної технології змінювалися разом із набуттям популярності певних мов програмування. В той час як препроцесор SASS був створений в 2006-ому році в момент, коли піку своєї популярності набувала мова Ruby, то LESS, створений в 2016-ому, керувався тенденцією на розвиток мови Javascript. Логічнішим і справді видається реалізація на мові, що найпростіше і майже завжди застосовується в Frontend розробці. Адже зникає необхідність встановлювати Ruby, додавати gem, тощо. Однак і препроцесор LESS не відразу надав користувачам дійсно доступний метод керування і компілював файли лише через Node.js, зараз це можливо зробити за допомогою під’єднання в HTML-файл стороннього сервісу ( посилання на віддалений сервер, що містить необхідні файли), який виконує і зчитування, і компіляцію.

Окрім реалізації препроцесора напряму на мовах програмування є й інші варіанти: окремі способи пропонують такі технології як PostCSS та Rework.

PostCSS – програма, що допомагає удосконалити механічні операції над CSS-кодом шляхом розширень написаних на мові Javascript. Була створена групою російських розробників з компанії «Злі марсіанини». Сам функціонал PostCSS не обмежується лише написанням шаблонів, які опісля компілюються в CSS, поле його застосування значно ширше, але включає в себе і можливість створити препроцесор по типу SASS/LESS.

PostCSS складається з чотирьох рівнів обробки розширення:

* Парсер (генерує дерево об’єктів із CSS-стрічки)
* Набір класів з яких формується дане дерево
* Генератор ( що створює CSS-код за деревом)
* Генератор карт-коду (який знаходить зміни в початковому і кінцевому CSS-коді)

PostCSS набув широкої популярності за останні роки: він є найбільш скачуваним розширенням в npm, ним користуються такі компанії-гіганти як Google, Facebook та Github. Зокрема самі розробники SASS радять використовувати PostCSS разом із їхнім продуктом (тим не менш, не визнаючи те, що PostCSS ледь не повністю покриває їхній функціонал власними розширеннями). «Його модульна архітектура означає, що ми можемо обрати і використати все, що ми хочемо. Це дозволяє зберігати фіксований малий розмір бібліотеки і додає функцію масштабування»[1, 3].

PostCSS доповнюється й різноманітними розширеннями для спрощення роботи:

* Автопрефіксер (дозволяє генерувати для вже готового коду CSS браузерні префіси)
* «PreCSS (набір функцій і розширень, які майже повністю покривають функціонал LESS/SASS)»[4].
* RTLCSS (дозволяє перетворювати код для арабського читання (справа наліво))
* Stylelint (перевіряє синтаксис коду, видає і дозволяє виправити наявні помилки)

Застосовувати можна не лише синтаксис PostCSS, а й під’єднавши кореневі файли LESS/SASS/Stylus. PostCSS не компілюватиме CSS-файл згідно з їхніми правилами, однак дозволить застосувати синтаксис у власному коді.

В PostCSS існує декілька видів плагінів:

* Пакети - інколи користувач не хоче встановлювати довгий список необхідних плагінів для одного проекту - саме в таких випадках використовуються пакети. Пакети містять в собі декілька модульних плагінів з однієї тематики і дозволяють встановити та задеплоїти їх одночасно. «Прикладом такого плагіну є PreCSS, який містить в собі більше 20 плагінів для CSS оптимізації» [3].
* Майбутній синтаксис CSS – цей тип плагінів «дозволяє вам використовувати той тип синтаксису, який ще не затверджений форматом W3C однак дуже скоро увійде в дію або просто не підтримується деякими браузерами» [3]. Прикладом таких плагінів є cssnext.
* Fallback – створені для того аби код працював в будь-яких браузерах. Зазвичай ці плагіни працюють без жодних кодових слів і знаходять необхідні для виправлення помилки в звичайному CSS-коді. Прикладом такого плагіну є Аutoprefixer.
* Розширення мови - наприклад можна «встановити прийнятний синтаксис по типу синтаксису деяких препроцесорів таких як LESS, SASS, Stylus, а також для деяких стандартів та методологій – таких як наприклад BEM та SUIT»[5].
* Grids – ця категорія плагінів працює з сіткою розміщення і не вимагає жодних передвизначених стилів або препроцесорних домішок. Прикладами таких плагінів є LOST, postcss-grid, postcss-neat
* Оптимізація – «діляться на дві основні категорії – мініфікація і кодова мініфікація»[3]. Вони можуть обрізати зайві коментарі, дужки, пробіли, з’єднювати весь файл в одну стрічку, тощо

Основним конкурентом PostCSS є Rework від засновника мови Node.js TJ Holowaychuk. Він з’явився першим і завдяки відомому розробнику здобув велику популярність. Не дивлячись на те, що можливості менші за PostCSS, однак і реалізація є простішою та меншою. Як визначає сам засновник PostCSS вона «швидша в два рази, краще підтримує карт-код, зберігає форматування і надає зручніший доступ до API».

## **1.4 Проблеми найпопулярніших CSS-препроцесорів та можливі методи їх вирішення**

Поява препроцесорів, а в особливості LESS та SASS, докорінно змінила шлях роботи з CSS як з технологією загалом. Однак препроцесори існують вже більше 10 років і мають багато недоліків, які ми розглянемо в цьому розділі.

Відкидаючи основні заперечення використання препроцесорів на користь ускладнення процесу розробки, розглянемо такі аргументи:

* Користувач не знає, що отримає як кінцевий код

Даний аргумент залежить не лише від знань розробника про той чи інший препроцесор, зокрема його методи обробки синтаксису і його інтерпретації, а й від загального сприйняття: код, що написаний за допомогою препроцесора в результаті (і навіть в більшості з усіх випадків) не несе жодної економії у кількості коду, створює дублювання там, де це не є необхідним і таким чином займає набагато більший об’єм пам’яті, ніж якби той же код писала людина на Pure CSS. З розділу 3 можна побачити, як деякі із препроцесорів (зокрема LESS I SASS) намагаються боротися з даною проблемою, Stylus в міру дозволу великої кількості синтаксичних вільностей втрачає значну частину можливостей для відслідковування таких критичних моментів. Основне збільшення об’єму відбувається за рахунок дублікації ієрархій, компіляції циклів (кожен item циклу буде відкомпільований окремо) і множення префіксів для коректного відображення елементів у різних браузерах.

Рішення: Насправді, збільшення об’єму пам’яті не є настільки значним і навіть у дуже великих проектах достатньо прибрати одну невеличку картинку із веб-сайту аби різниці між чистим CSS і зкомпільованим з препроцесора не було.

* Debug стає складнішим

Обов’язковим кроком роботи, звісно, є компіляція, відповідно виконаний CSS файл завжди відрізнятиметься від початкового, в більшості випадків – відмінності значні. Це все ускладнює відлагодження кінцевого коду, адже розробник не орієнтується ( і за основним принципом роботи препроцесорів, не повинен) в отриманому файлі, як результат – тим складніше йому зрозуміти проблему.

* Поява постпроцесорів

На заміну препроцесорам згодом прийшли постпроцесори. За їхньою назвою складно зрозуміти, як вони впливають на вже робочий CSS-код після його зчитування. Трактувати назву слід так: якщо препроцесори читають і збирають розширену мову в CSS, то постпроцесори читають і збирають готовий CSS. Найпопулярнішими технологіями в цьому напрямку наразі є Autoprefixer, PostCSS та Rework. До їхніх переваг можна приписати: автоматичне доповнення вендорних префіксів для підтримки кросбраузерності (нам не потрібно писати для цього спеціальні домішки), додавання так званих «хаків» для певних браузерів ( у випадку препроцесора нам необхідно створювати домішку на кожен випадок такого «хаку», в постпроцесорі ми описуємо його відразу для усіх схожих властивостей у функції). До того ж існує бібліотека Pleeease!, яка зібрала в собі усі найбільш поширені випадки «хаків» (в особливості для браузера Internet Explorer)

Рішення: Пре- і постпроцесори мають різні цілі і інколи є зміст використовувати в проекті і те, і інше. Препроцесори для логіки\структури\композиції, а постпроцесори для того аби змінити\прибрати певні стилі.

* Змінні не завжди зменшують час роботи над кодом

Змінні в препроцесорах явище зручне і впровадження його зрозуміле – розробник не хоче змінювати колір сайту у кожному конкретному атрибуті стилю і завдяки препроцесору йому достатньо змінити лише значення змінної кольору. Однак чи так часто ми змінюємо значення певного компоненту, а не сам компонент? Доволі часто зустрічається одна і інша ситуація. І якщо першу проблему це вирішує, то зміна властивостей відносно певного компонента вимагає технічних перетворень.

* Нерозвинена спільнота

Більшість препроцесорів є open-source програмами, тобто відкритими для редагування і доповнення будь-якому розробнику. Водночас з поєднанням малої популярності деяких з них (до прикладу, Stylus) це зовсім не покращує роботу самого процесора. Зокрема в github-репозиторії препроцесора Stylus досі можна знайти bug reports (звіти, щодо помилок і виправлень) дворічної давності на яку досі немає відповіді від розробників. В галузі, що настільки швидко розвивається, так неприпустимо. Тому багато розробників дописують власні функції і доповнюють код локально аби отримати бажаний результат.

* Велику кількість функціоналу препроцесорів покриває хороше середовище розробки

Зараз популярності набувають такі середовища розробки як Espresso, Caffeine, тощо, які і справді дозволяють нам користуватися можливостями препроцесорів завдяки функціоналу або значно спрощують проблеми, які ті самі препроцесори намагаються виправити. Таким чином, інколи досягти правильної ієрархії і використати змінні стає легше без встановлення препроцесора і довгої (при великій кількості коду) компіляції. До прикладу, середовище розробки Espresso дозволяє нам розглянути на розробку CSS як на дерево, де властивості певного елемента є листками цього дерева.

# **РОЗДІЛ 2. ТЕХНОЛОГІЯ РОЗРОБКИ ТА СТРУКТУРА ПРОЕКТУ**

## **2.1. Основна ідея і мета проекту**

Препроцесори та постпроцесори створені більше для рішень суб’єктивних, аніж об’єктивних, адже стилістику важко розглядати як щось однозначне для усіх розробників. Метою даної курсової роботи стало розглянути і проаналізувати деякі власні суб’єктивні аспекти препроцесорів на базі плагінів для PostCSS. Оскільки окрім багатьох переваг, варто відзначити і велику перевагу в модульності системи PostCSS – це значно зменшує об’єм збережених даних в бібліотеці і керується це лише плагінами, що використовуються на даний момент і завантажені самим розробником в проект.

Найкращою ідеєю в даному випадку стало поєднати основні функції препроцесорів (тобто сформувати міні-препроцесор з мінімальним набором функціоналу) і додати до нього ряд плагінів, що суттєво доповнять стандартний набір функцій. Серед функцій я намагалася обрати такі, що не отримали реалізації або мають досить застарілі версії.

Серед стандартних функцій у вигляді плагінів було створено наступні:

* Підтримка змінних за допомогою нотації через $
* Можливість коригувати кольори для браузера Інтернет Експлорер
* Можливість використання умовних операторів (if та else)
* Можливість скорочено задавати кольори
* Можливість скорочено задавати шрифтові набори
* Та інші.

Основними ідеями, що я керувалася при створенні набору препроцесорних плагінів були:

* «Створення вендорних префіксів для елементів, що дозволено лише в постпроцесорах, тобто фактично ця функція в препроцесорах відсутня»[4]. Однак можливості PostCSS дають це зробити.
* Відловлювання винятків для старих версій браузера Internet Explorer, що автоматично змінює/додає необхідні значення певним параметрам ( наприклад, перетворення rgba формату, що не підтримується в IE (Internet Explorer) на hex формат і фільтр opacity відповідно).
* Автоматична генерація @font-face атрибутів для зовнішніх і вбудованих шрифтів.

Окрім безпосереднього встановлення PostCSS на локальній машині, надається можливість користуватися сервісами з інтегрованою технологією: CodePen та Prepros.

У випадку ж встановлення PostCSS на комп’ютері (той спосіб, що був використаний для виконання практичної частини ) необхідний такий набір технологій:

* Node.js
* Grunt (для Windows/Linux/Ubuntu) та Gulp (для MacOS)
* NPM

Для початку необхідно встановити Grunt або Grunt-CLI через консоль NPM, так аби структура проекту виглядала наступним чином (див. рис. 2.2.1) :

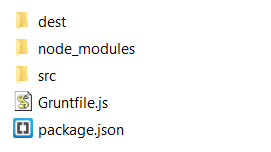


Рис 2.2.1 – Приклад структури проекту

Файл Gruntfile.js міститиме набір необхідних опцій для запуску PostCSS. Встановимо модуль grunt-postcss за допомогою наступної команди (див. рис. 2.2.2):



Рис. 2.2.2 – Приклад команди для встановлення PostCSS

Додаток “—save-dev”” використовуємо для того аби всі завантажені модулі зберігалися в нашому проекті локально і користувачу не було потрібно встановлювати їх, якщо він хоче користуватися цим набором плагінів.

В наступному розділі розглянемо налаштування проекту та короткий туторіал до написання одного з плагінів практичної частини.

## **2.2 Підбір технологій, встановлення й налаштування PostCSS**

Для того аби розпочати роботу із сконфігурованим за допомогою команд в розділі 2.1 проекті - відредагуємо файл Gruntfile.js. Ініціалізуємо Grunt та поставимо «завдання» підвантажити PostCSS

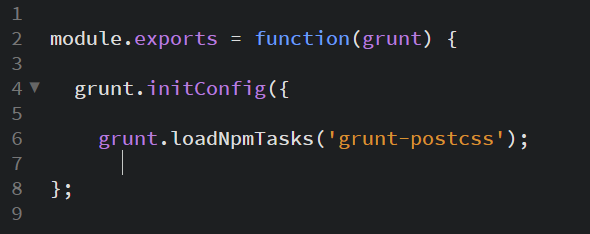


Рис. 2.2.3 – Налаштування Gruntfile.js

До блоку initConfig додаємо PostCSS, що містить в собі два елементи:

* Processors зберігає конфігурації проекту
* Dist зберігає вхідний та вихідний файл для перетворення за допомогою застосованих в processors плагінів.

В processors згадуються усі плагіни, що встановлені через консоль і знаходяться в папці node-modules. Туди ж додаються і папки власних плагінів (як в прикладі на рисунку 2.2.4 – власні плагіни це postcss-myplugin, postcss-size, postcss-color-short). Для того щоб скомпілювати написаний нами код у вихідний код чистого CSS достатньо просто виконати команду grunt postcss.



Рис. 2.2.4 – Приклад конфігурації файлів

Розглянемо алгоритм дій для створення власного плагіну на прикладі вже готового плагіну практичної частини - postcss-fontface.

До папки node-modules, що знаходиться в кореневому каталозі проекту додамо теку postcss-fontface. В неї додаються js файли, демо та документація плагіну. Детальніше розглянути офіційний маніфест щодо структуризації плагінів можна в офіційній документації PostCSS.

Наш плагін буде допомагати з вирішенням проблеми додаткових шрифтів. Коли ми вказуємо якийсь особливий шрифт, досить часто деякі браузери нездатні його зчитати, а отже необхідно перераховувати ряд шрифтів схожого типу як додаткові. Ми допоможемо користувачу замість довгої стрічки додаткових шрифтів, як на рисунку 2.2.5, вказувати лише тип шрифта (Arial, Times New Roman, тощо), а наш плагін уже генеруватиме необхідні значення з набору.

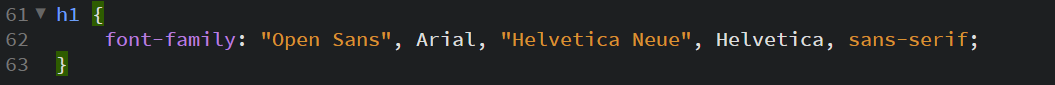


Рисунок 2.2.5 – Незручний спосіб вказання шрифтів

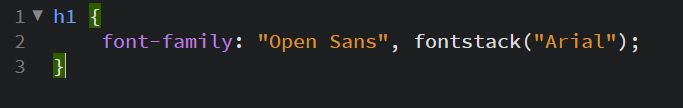


Рис. 2.2.6 – Приклад необхідного синтаксису для плагіну

Для нашого плагіну додаємо в кореневий каталог файл index.js.

Повний код плагіну виглядає ось так:



Рисунок 2.2.7 – Код плагіну

Розберемо покроково, що відбувається в плагіні.

В першому рядку викликаємо postcss для того, аби виконався наступний код, що починається з 9-го рядка.

Модуль підвантажує наш плагін (саме завдяки цьому його можна пізніше викликати в Gruntfile.js), в семантику функції можна також додати опції як певні параметри, однак в даному плагіні така реалізація не знадобилася (тим не менш, як опції можна було б вказувати додаткові шрифти, що не є стандартними для CSS). В самій функції міститься два цикли: «css.walkRules проходить через кожне правило в CSS-файлі» [6] (під правилом мається на увазі набір елемент-селектор-властивість. Тобто саме елемент і все, що знаходиться між фігурними дужками), а «rule.walkDecls проходить через кожну декларацію, тобто – кожен стиль елементу (набір властивість-значення)»[6].

Отже, кожній декларації шукатимемо такий рядок, де присутнє значення fontstack – кодове слово, де в дужках міститиметься коротке значення необхідного набору шрифтів.

Оскільки значення нам прийде в вигляді стрічки з лапками, використаємо регулярний вираз щоб прибрати з дужок лапки і й непотрібні символи.

Також нотування шрифтів в CSS та JS case-sensitive, тому застосовуємо функцію toTitleCase і перетворюємо першу літеру значення на велику.

Опісля шукаємо в нашому масиві шрифтів таке значення ключа, яке відповідає значенню fontstack\_requested. Перед тим, як додавати відповідне значення до ключа в css-файл перевіримо чи основний шрифт, що вказується перед fontstack не присутній в нашому наборі – для цього запишемо значення substr в first\_font (в разі, якщо такого значення немає залишиться просто пуста стрічка) і об’єднуємо його в змінній new\_value. Передаємо її в decl\_value, де вже PostCSS замінить всі згадування fontstack в файлі на необхідні значення.

Ось і все, що необхідно для створення плагіну для PostCSS. В наступному розділі ми детальніше розглянемо всі плагіни, що були створені для практичної частини курсової роботи з більш загальної точки зору.

## **2.3 Огляд створених програм**

В цьому розділі детальніше розглянемо створені для практичної частини плагіни.

В ході виконання практичної частини було реалізовані наступні корисні для користувача плагіни.

Перший плагін реалізовує виправлення проблеми з сприйняттям деякими браузерами властивості opacity. До прикладі, браузер Internet Explorer та деякі старі версії Mozilla Firefox та Safari не підтримують цю властивість. Internet Explorer натомість застосовує фільтр типу:

**progid:DXImageTransform.Microsoft.Alpha(Opacity=?);**

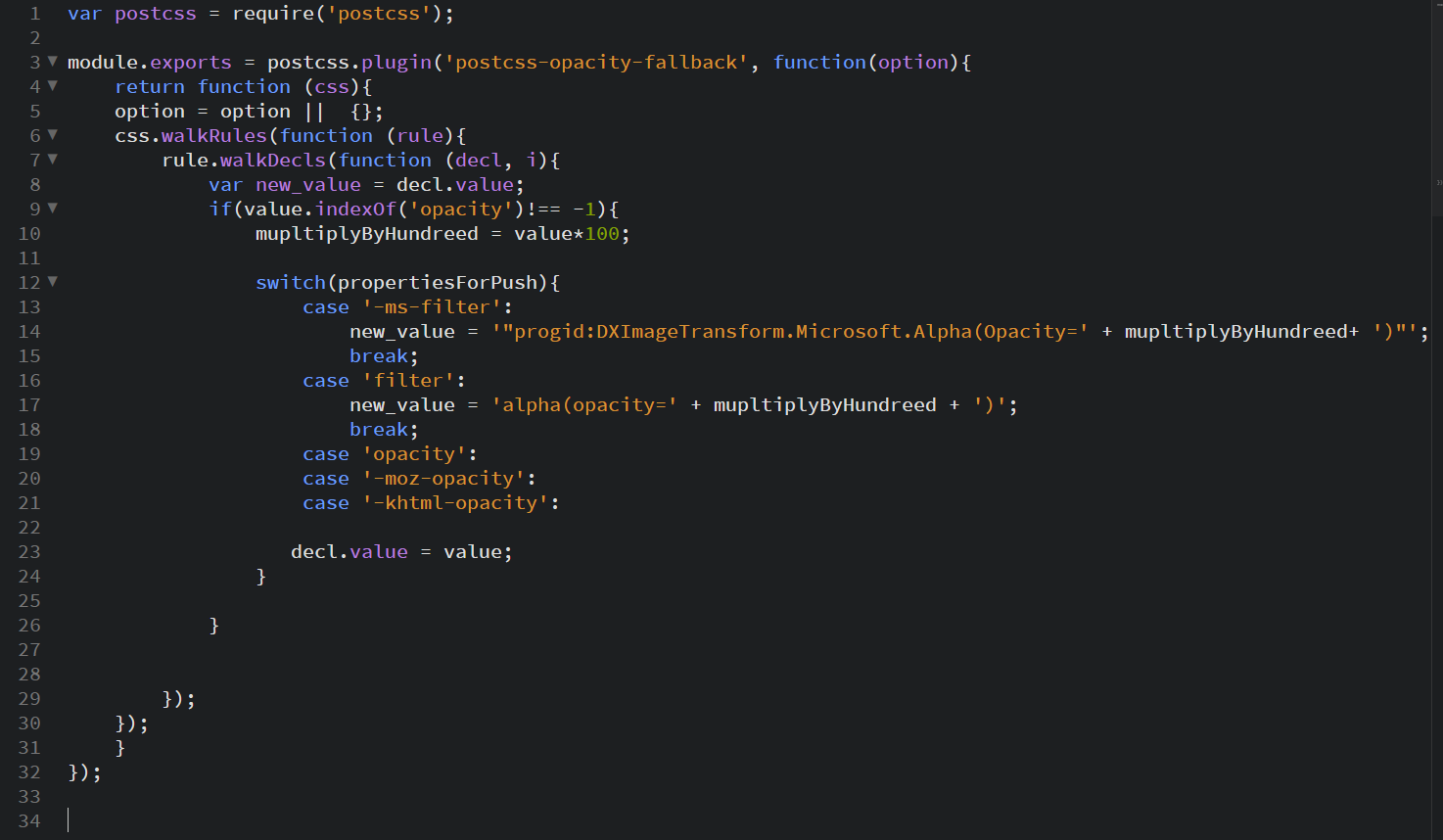


Рисунок 2.3.1 – Приклад коду плагіну postcss-opacity-ie

Цей досить простий плагін побудовано за схемою того, що ми перевіряємо кожне правило на наявність властивості opacity – якщо вона є – перевіряємо чи не задані вже додаткові властивості і тоді саме їх пропускаємо. Якщо немає – йдемо далі і додаємо нові значення в декларацію властивості в строгому порядку (інакше деякі з них можуть не прочитатися), якщо якась з властивостей уже існує, то додаємо інші в порядку черги.

Для прикладу, також було реалізовано postcss-opacity-ie-плагін для браузера Internet Explorer. «На відміну від більшості браузерів, де у використанні вже давно є формат кольору RGBA (Red, Green, Blue, Alpha), де канал Alpha відповідає за прозорість кольору»[10, 149]. Умовно цей формат кольору можна розбити на два інші елементи – rgb та opacity, однак браузер Internet Explorer не підтримує атрибут opacity. Саме з цією проблемою повинен допомогти наступний плагін:

- для його роботи непотрібно вживати в CSS-файлі жодних ключових слів, він сам знайде необхідні елементи з переліку і перевірить чи є в них елемент rgba (серед таких елементів background, text-color, background-color, color, тощо)

- працює за ідеєю постпроцесингу, тобто обробляє вже фінальний варіант CSS-коду, отже плагін необхідно підвантажити пізніше в списку

- застосовуємо попередній плагін з opacity додавши до нього необхідне перетворення трьох каналів з RGBA в RGB

Останнім з плагінів для розгляду пропоную розглянути responsive-image. Для роботи цього плагіну достатньо додати атрибут image-size та вказати йому значення responsive. Це автоматично додасть атрибути необхідні для коректного відображення картинки на усіх екранах (див. рис. 2.3.2).

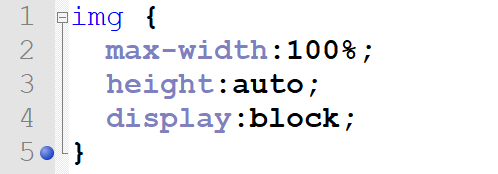


Рисунок 2.3.2 – Результат роботи плагіну responsive-image

На рисунку 2.3.3 зображений повний код роботи плагіну.

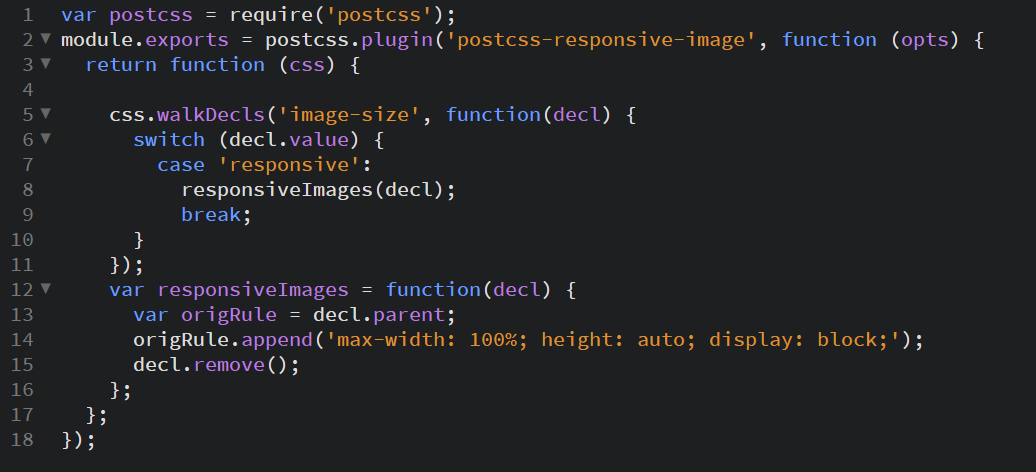


Рисунок 2.3.3 – Код роботи плагіну

Спочатку проходимося по усім значенням властивостей і шукаємо там атрибут image-size, якщо його знайдено – до значення декларації з “responsive” застосовуємо функцію responsiveImages, де надаємо значення декларації стрічці з необхідними параметрами.

Після цього згенерований файл виглядатиме так, як на рисунку 2.3.2 ( див. рис. 2.3.2.) .

# **ВИСНОВКИ**

В результаті написання даної курсової роботи було виокремлено та досліджено особливості CSS-препроцесорів та процесорів по типу технології PostCSS, виконано детальний огляд найпопулярніших препроцесорів (SASS, LESS, Stylus), основних плагінів та додатків. Виконано глибоке порівняння наявних реалізацій, основних підходів до здійснення перетворення CSS-файлів. Завдяки даному аналізу та виконанню практичної частини вдалося створити пакет плагінів, що дозволить розширити базу знань для розробників, що хочуть вивчити або використати у своєму застосуванні PostCSS, а також використати ці плагіни для власного користування в проекті. Завдяки реалізації на PostCSS, розробник зможе знайти для себе ще один зручний спосіб керування синтаксисом CSS.

Підсумовуючи, варто виділити те, що CSS-препроцесори досі залишаються на хвилі своєї популярності, однак потрохи витісняються більш сучасними і зручними технологіями, такими як PostCSS. Тим не менш, самі препроцесори заслуговують окремої уваги Front-End розробників, оскільки їхні ідеї та концепти увійшли до, зокрема, PostCSS. Препроцесори як ідея покликані спростити та звести до мінімуму роботу розробника з незручними функціями та дублюванням коду.

# **ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ**

1. Alex L. Mastering PostCSS for Web Design / Libby Alex., 2016. – 414 с. – (Packt Publishing).
2. Anirudh P. Beginning CSS Preprocessors / Prabhu Anirudh., 2015. – 142 с. – (Apress).
3. Bracey K. PostCSS Quickstart Guide: Exploring Plugins [Електронний ресурс] / Kezz Bracey // Webdesign TetsPlus. – 2015. – Режим доступу до ресурсу: <https://webdesign.tutsplus.com/tutorials/postcss-quickstart-guide-exploring-plugins--cms-24566>.
4. Bracey K. Using PostCSS Together with Sass, Stylus, or LESS [Електронний ресурс] / Kezz Bracey // Webdesign TutsPlus. – 2015. – Режим доступу до ресурсу: <https://webdesign.tutsplus.com/tutorials/using-postcss-together-with-sass-stylus-or-less--cms-24591.>
5. Bracey K. Using PostCSS with BEM and SUIT Methodologies [Електронний ресурс] / Kezz Bracey // Webdesign TutsPlus. – 2015. – Режим доступу до ресурсу: <https://webdesign.tutsplus.com/tutorials/using-postcss-with-bem-and-suit-methodologies--cms-24592.>
6. Bracey K. PostCSS Deep Dive: Create Your Own Plugin [Електронний ресурс] / Kezz Bracey // Webdesign TutsPlus. – 2015. – Режим доступу до ресурсу: <https://webdesign.tutsplus.com/tutorials/postcss-deep-dive-create-your-own-plugin--cms-24605.>
7. Bracey K. PostCSS Deep Dive: Roll Your Own Preprocessor [Електронний ресурс] / Kezz Bracey // Webdesign TutsPlus. – 2015. – Режим доступу до ресурсу: <https://webdesign.tutsplus.com/tutorials/postcss-deep-dive-roll-your-own-preprocessor--cms-24584>.
8. Giraudel H. Jump Start Sass: Get Up to Speed With Sass in a Weekend / H. Giraudel, M. Suzanne., 2016. – 180 с. – (1).
9. Libby A. Learning Less.js / Alex Libby., 2014. – 342 с. – (Packt Publishing).
10. McFarland D. CSS: The missing manual / David McFarland., 2015. – 718 с. – (4).
11. Verou M. CSS Secrets / Michailia Verou., 2015. – 370 с. – (O'Reilly).

# **ДОДАТОК А** Приклад роботи плагінів

