НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Романкевич В. О.
“__” червня 2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

зі спеціальності **123 «Комп'ютерна інженерія»**

на тему: Системний модуль аналізу сутностей бази даних продуктів у Salesforce Commerce Cloud

Виконала: студентка IV курсу, групи КВ-92

Корж Анна Андріївна _____

Керівник доц.каф.СПСКС, к.т.н. Петрашенко А. В. _____

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____

Рецензент _____ _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Романкевич В. О.*
(підпис) (ініціали, прізвище)
«___» червня 20___ р.

ЗАВДАННЯ
на дипломний проєкт студента
Корж Анни Андріївни
(прізвище, ім’я, по батькові)

1. Тема проєкту *«Системний модуль аналізу сутностей бази даних продуктів у Salesforce Commerce Cloud»*,

керівник проєкту *доц.каф.СПСКС, к.т.н. Петрашенко А. В.*,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» _____ 20__ р. №_____

2. Термін подання студентом проєкту «*19» червня* 20*23* р.

3. Вихідні дані до проєкту *Технічне завдання*

4. Зміст пояснювальної записки

1) Аналіз існуючих рішень та обґрунтування теми дипломного проєкту

2) Опис платформи SFCC. Створення користувацьких системних модулів

3) Алгоритм роботи системного модуля для аналізу сутностей бази даних продуктів у SFCC

4) Аналіз отриманих результатів.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

1) Схема алгоритму аналізу записів бази даних продуктів SFCC

2) Структурна схема допоміжних класів для обробки та валідації даних

3) ERD-діаграма зв'язків сутності продукту бази даних SFCC

4) Схема взаємодії функціональних модулів створеного картриджу

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 07.11.2022

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Видача завдання на дипломне проектування	07.11.2022	
2	Вивчення літератури за тематикою роботи	21.11.2022	
3	Розроблення та узгодження технічного завдання	28.11.2022	
4	Розроблення структури додатку	20.01.2023	
5	Розроблення дизайну та графічних елементів	03.02.2023	
6	Програмна реалізація додатку	13.03.2023	
7	Тестування додатку	03.04.2023	
8	Підготовка матеріалів текстової частини проєкту	24.04.2023	
9	Підготовка матеріалів графічної частини проєкту	15.05.2023	
10	Оформлення технічної документації проєкту	31.05.2023	

Студент _____ *Корж А. А.*
(підпис) (ініціали, прізвище)

Керівник проєкту _____ *Петрашенко А. В.*
(підпис) (ініціали, прізвище)

* *** Консультантом не може бути зазначено керівника дипломного проєкту.

АНОТАЦІЯ

Бакалаврський дипломний проєкт включає пояснювальну записку (64 стор., 21 рис., список використаної літератури з 11 найменувань, 3 додатків).

Об’єкт розробки: створення системного модулю, що дозволить користувачам переглядати інформацію про невірно сформовані записи продуктів у базі даних.

Системний модуль дозволяє: задавати атрибути необхідні для перевірки запису на коректність; обирати записи, що мають бути відображені у звіті (усі записи продуктів, тільки невірні та тільки вірні записи); задавати продукти для перевірки списком ідентифікаторів або атрибутом сезону, до якого продукти відносяться; передбачені механізми захисту від помилок; переглядати інформацію звіту у вигляді таблиці не покидаючи сторінку додатку; зберігати звіт у вигляді .csv файлу. Передбачена можливість перегляду статистики всіх продуктів сайту у зручному вигляді діаграм.

Програма побудована у вигляді так званого картриджу для Salesforce Commerce Cloud, що являє собою програмний модуль написаний мовою програмування JavaScript, вбудованим у SFCC шаблонізатором ISML та мовою сценаріїв препроцесора SASS.

В ході розробки:

– проведено аналіз існуючих можливостей аналізу записів продуктів у базі даних Salesforce;

– сформульовані вимоги для правильного відображення продукту на сайті;

– розроблена частина програми, що дозволяє обирати атрибути для перевірки;

– розроблена частина програми, що дозволяє вказувати список ключових атрибутів продуктів для перевірки;

– розроблена частина програми, що дозволяє перевіряти продукти, що належать до обраного сезону;

– розроблено метод для відображення звіту у вигляді таблиці;

– розроблено модуль для створення .csv файлу звіту;

– розроблено модуль для відображення статистики по всіх продуктах сайту у вигляді діаграм.

Підключення даного системного модуля в Business Manager SFCC дозволить користувачам переглядати невірно сформовані записи продуктів на етапі налаштування та виправляти їх до публікації на сторінці сайту. Даний функціонал не передбачений у самій платформі і може бути корисним для дата менеджерів при створенні нових записів, а також для валідації існуючих.

Ключові слова: Salesforce Commerce Cloud, база даних, атрибути, сутності, продукти, записи, JavaScript, валідація.

ABSTRACT

A bachelor's thesis project includes an explanatory note (64 pages, 21 figures, a list of references with 11 titles, 3 appendices).

The object of development is the creation of a system module that will allow users to view information about incorrectly generated product records in the database.

The system module allows setting the attributes necessary to check the record for correctness; select the data records to be displayed in the report (all product records, only incorrect and only correct records); to specify products for checking by a list of identifiers or an attribute of the season to which the products belong; to view the report information in the form of a table without leaving the application page; to save the report as a .csv file; to display statistics of all site products in the convenient form of diagrams. Have provided error protection mechanisms to the module.

The program structure is a so-called cartridge for Salesforce Commerce Cloud, a program module written in the JavaScript programming language, the ISML template engine built into SFCC, and the SASS preprocessor scripting language.

During development:

– analyzed the existing capabilities for analyzing product records in the Salesforce database;

– formulated requirements for the correct display of the product on the website;

– developed a part of the program that allows you to select attributes for verification;

– developed a part of the program that allows you to specify a list of key product attributes for verification;

– developed a part of the program that allows checking products belonging to the selected season;

– developed a method for displaying the report in the form of a table;

– developed a module for creating a .csv file of the report;

– developed a module to display statistics on all site products in diagrams.

The connection of this system module to the SFCC Business Manager will allow users to view incorrectly generated product records at the configuration stage and correct them before publishing on the site page. Salesforce Commerce Cloud CRM platform does not provide this functionality. The module can be convenient for data managers while creating new records and validating existing ones.

Keywords: Salesforce Commerce Cloud, database, attributes, entities, products, records, JavaScript, validation.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.002 ТЗ	Системний модуль аналізу	4		
			сутностей бази даних продуктів			
			у Salesforce Commerce Cloud.			
			Технічне завдання.			
	А4	ІАЛЦ.045490.003 ТП	Системний модуль аналізу	2		
			сутностей бази даних продуктів			
			у Salesforce Commerce Cloud.			
			Відомість технічного			
			проєкту.			
	А4	ІАЛЦ.045490.004 ПЗ	Системний модуль аналізу	64		
			сутностей бази даних продуктів			
			у Salesforce Commerce Cloud.			
			Пояснювальна записка.			
	А4	ІАЛЦ.045490.005 Д1	Алгоритм аналізу записів	1		
			бази даних продуктів SFCC.			
			Схема алгоритму.			

Змін.	Арк.	№ докум.	Підпис	Дата	**ІАЛЦ.045490.001 ОА**		
Розробив		Корж А. А.			Системний модуль аналізу сутностей бази даних продуктів у Salesforce Commerce Cloud ***Опис альбому***	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Петрашенко А. В.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-92	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.006 Д2	Схема допоміжних класів	1		
			для обробки та валідації			
			даних.			
			Діаграма класів.			
	А4	ІАЛЦ.045490.007 Д3	ERD-діаграма зв’язків	1		
			сутності продукту бази			
			даних SFCC.			
			Схема структурна.			
	А4	ІАЛЦ.045490.008 Д4	Схема взаємодії	1		
			функціональних модулів			
			створеного картриджу.			
			Схема взаємодії програм.			
		Диск CD-ROM	Текст пояснювальної записки.			
			Архівований код програми			
			у форматі .zip.			
			Інструкція по експлуатації			
			додатку англійською мовою.			
			Презентація дипломного			
			проєкту.			
			Графічний матеріал			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Системний модуль аналізу сутностей бази даних продуктів у Salesforce Commerce Cloud».

Галузь застосування: попередня валідація сутностей продуктів перед їх публікацією на сайтах, що побудовані за допомогою платформи SFCC.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення системного модулю (картриджу SFCC) для перевірки коректності обраних записів у базі даних продуктів.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

1. сумісність з операційними системами ПК (Windows. Linux, IOS);
2. можливість вказання записів для перевірки;
3. можливість управління параметрами для валідації;
4. можливість перегляду статистики для всіх продуктів сайту;
5. можливість перегляду інформації про записи із вказаним атрибутом сезону;
6. наявність системи сповіщень про готовність звіту валідації;
7. можливість перегляду звіту у вигляді .csv файлу та у табличному вигляді;
8. відображення попередньо згенерованих файлів звіту;
9. збереження файлів звіту у сховищі Impex середовища розробки;

5.2. Вимоги до апаратного забезпечення

1. Наявність доступу до мережі Internet;

5.3. Вимоги до програмного та апаратного забезпечення користувача

1. Операційна система Windows, iOS, Linux;
2. Наявність доступу до мережі Internet;
3. Акаунт у системі SFCC із доступом до утиліти Business Manager середовища сайту.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.04.2023
2.	Розроблення та узгодження технічного завдання	30.04.2023
3.	Аналіз існуючих рішень	05.05.2023
4.	Підготовка матеріалів першого розділу дипломного проекту	10.05.2023
5.	Підготовка матеріалів другого розділу дипломного проекту	18.05.2023
6.	Підготовка графічної частини дипломного проекту	20.05.2023
7.	Оформлення документації дипломного проекту	25.05.2023
8.	Попередній огляд матеріалів диплому на кафедрі	30.05.2023

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.004 ПЗ	Системний модуль аналізу	64		
			сутностей бази даних продуктів			
			у Salesforce Commerce Cloud.			
			Пояснювальна записка			
	А4	ІАЛЦ.045490.005 Д1	Алгоритм аналізу записів	1		
			бази даних продуктів SFCC.			
			Схема алгоритму			
	А4	ІАЛЦ.045490.006 Д2	Схема допоміжних класів	1		
			для обробки та валідації			
			даних.			
			Діаграма класів			
	А4	ІАЛЦ.045490.007 Д3	ERD-діаграма зв’язків	1		
			сутності продукту бази			
			даних SFCC.			
			Схема структурна			
	А4	ІАЛЦ.045490.008 Д4	Схема взаємодії	1		
			функціональних модулів			
			створеного картриджу.			
			Схема взаємодії програм			

Змін.	Арк.	№ докум.	Підпис	Дата	**ІАЛЦ.045490.003 ТП**		
Розробив		Корж А. А.			Системний модуль аналізу сутностей бази даних продуктів у Salesforce Commerce Cloud ***Відомість технічного проєкту***	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Петрашенко А. В.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-92	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
		Диск CD-ROM	Текст пояснювальної записки.			
			Архівований код програми			
			у форматі .zip			
			Інструкція по експлуатації			
			додатку англійською мовою.			
			Презентація дипломного			
			проєкту.			
			Графічний матеріал			

ЗМІСТ

ДОДАТКИ

Додаток 1. Копії графічних матеріалів

- ІАЛЦ.045490.005 Д1. Алгоритм аналізу записів бази даних продуктів SFCC. Схема алгоритму.
- ІАЛЦ.045490.006 Д2. Схема допоміжних класів для обробки та валідації даних. Діаграма класів.
- ІАЛЦ.045490.007 Д3. ERD-діаграма зв’язків сутності продукту бази даних SFCC. Схема структурна.
- ІАЛЦ.045490.008 Д4. Схема взаємодії функціональних модулів створеного картриджу. Схема взаємодії програм.

Додаток 2. Архів з програмним кодом розробленого картриджу

Додаток 3. Інструкція по експлуатації системного модуля (Eng.)

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

Customer relationship management (CRM — управління взаєминами з клієнтами) – це технологія для управління всіма відносинами та взаємодією компанії з клієнтами та потенційними клієнтами, інструмент, який допомагає в управлінні контактами, продажами, продуктивністю агентів тощо [1].

Salesforce Commerce Cloud (SFCC) – багатокористувацька хмарна комерційна платформа, раніше відома як Demandware (DW), яка дозволяє брендам створювати інтелектуальний, уніфікований досвід покупок на всіх платформах: мобільній, соціальній, веб та магазинній; є провідним світовим рішенням для торгівлі моделей B2C та B2B [2].

Business Manager (BM) – це керуючий центр для реалізації управління продажами, адміністрування та розробки сайтів у Salesforce B2C Commerce [3].

База даних (бд) — це структуроване сховище інформації, яке дозволяє зберігати, організовувати і керувати даними. Вона використовується для збереження великого обсягу даних, доступу до них, пошуку, оновлення та виконання різних операцій.

JavaScript (JS), Type Script (TS) – мови програмування високого рівня.

Картридж – це механізм для пакування та встановлення програмного коду і даних. Використовуються для розширення функціональності або інтеграції з зовнішніми системами [4].

Open Commerce API (OCAPI) – програмний інтерфейс, який надає розробникам доступ до функціональності Salesforce Commerce Cloud і дозволяє взаємодіяти з eCommerce платформою. OCAPI забезпечує можливість здійснювати операції з продуктами, замовленнями, кошиками, користувачами та іншими елементами eCommerce системи.

Job – це автоматизовані рутинні завдання або довготривалі процеси, такі як імпорт та експорт даних, реплікація даних або коду чи збірка

пошукового індексу. Jobs можуть використовувати готові системні кроки (скрипти, що задають функціонал роботи Job) або розробник може створювати власні кроки завдання [4].

Локалізований атрибут — такий атрибут сутності бази даних, який може мати різні значення для різних локалей або мов.

Custom Object (CO) – об'єкти, які можуть бути створені та налаштовані користувачами для зберігання та управління додатковою інформацією, яка не входить до стандартних об'єктів, що постачаються з платформою.

Base Product (BP), або Master Product – продукт, який представляє всі варіації для певного продукту. Базовий продукт не можна придбати безпосередньо, можна придбати лише варіант продукту з конкретною інформацією про нього [4].

Variation Product (VP) – продукт, який є конкретною варіацією (наприклад, футболка бренду Vegas розміру 10, синього кольору – це продукт-варіація базового продукту “Футболка бренду Vegas”). Продукти-варіації, пов'язані з тим самим базовим продуктом-варіацією, мають більшість атрибутів, визначених для базового продукту, але мають свої власні ідентифікатори та зображення [4].

Variation Group (VG) – це група товарних позицій, які мають спільну ознаку, наприклад, колір або розмір. Вона завжди належить до базового продукту і представляє групу варіацій продуктів [4].

ВСТУП

Salesforce Commerce Cloud вважається лідером серед eCommerce платформ. Вона є зручним інструментом для створення та підтримки сайтів для своїх клієнтів, надаючи їм при цьому широкі можливості персоналізації та менеджменту. Однак, як і будь-яка інша система вона має свої обмеження та недоліки. Цей проєкт створений для того, щоб мінімізувати один з них.

Досить часто у робітників ремесла менеджменту виникають проблеми із заповненням записів у базі даних. З метою покращення взаємодії з нею та зменшення впливу людського фактора у рамках проєкту було розроблено новий системний модуль, що дозволить користувачам створювати безпомилкові сутності продуктів, які гарантовано відображатимуться на сайті коректно.

Тема проєкту була сформована на особистому досвіді роботи із платформою, на основі спілкування із працівниками сфери управління даними. У проєкті поставлено конкретні критерії успішності роботи картриджу та ті задачі, що мають бути ним вирішені.

Задля кращого розуміння інформації, що стосується модуля слід бути у контексті архітектури платформи SFCC та розуміти процес створення розширюючого функціоналу для неї. Для цього в роботі присутній розділ, де наведено опис вищезгаданих понять.

Найбільший розділ пояснювальної записки присвячено опису алгоритму роботи частини програми, відповідальної за валідацію записів. Вона є найважливішою для розуміння функціонування розробленого модуля. Картридж також формує статистику для усіх продуктів сайту, що є додатковим функціоналом створеним для глобального аналізу цілісності сутностей продуктів бази даних.

В останньому розділі наведено результат тестування додатку, а також проведено аналіз його недоліків та запропоновано ідеї для покращення.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

1.1. Обґрунтування теми

Тема дипломного проєкту: «Системний модуль аналізу сутностей бази даних продуктів у Salesforce Commerce Cloud».

На вибір теми дипломного проєкту вплинув власний досвід роботи із базою даних платформи SFCC. А саме, бажання спростити користування нею працівникам сфери менеджменту даних. Одним з недоліків платформи є недосконала валідація даних. Завдяки цьому при заповненні записів не можна гарантувати, що елементи сайту описані сутностями відображатимуться відповідно до уявлень замовника.

Метою є надати користувачам (дата менеджерам, контент менеджерам і т.д.) засіб, що дозволить аналізувати правильність атрибутів продуктів на етапі їх створення, а також перевіряти ті записи, що вже розміщені на сайті. Це дозволить зменшити кількість помилок та забезпечити цілісність бази даних.

Основним завданням даного дипломного проєкту є сформувати критерії згідно за якими можна вважати, що елементи (в даному випадку — продукти на сайті) виглядатимуть коректно, тобто всі необхідні атрибути сутності заповнені та мають очікувані значення, і створити системний модуль, що проводитиме валідацію згідно цих критеріїв та відображатиме її результат у зручному для замовника вигляді.

Також однією із головних вимог проєкту є створення зручного візуального інтерфейсу, що буде зрозумілим для замовника. Цей інтерфейс має дозволяти обирати атрибути за якими сутність має бути провалідована, надавати список ідентифікаторів або ідентифікатор сезону записів, що мають бути перевірені та відображати результат аналізу на сторінці, якщо це

можливо (розмір звіту не перевищує максимального розміру визначеного квотою SFCC).

Усі операції з даними мають виконуватись у асинхронному режимі за допомогою job фреймворку. Вивід звіту про результат валідації має бути реалізований у двох видах: табличному (якщо звіт досить невеликий, щоб бути відображеним на сторінці) та у вигляді файлу формату .csv.

Програма має до того ж надавати можливість не тільки задавати критерії для аналізу продуктів за вказанням атрибуту ідентифікатора або сезону, а й можливість перегляду статистики по усіх існуючих продуктах сайту. Статистика має відображатись у вигляді діаграм, що будуються на основі таких даних: частка невірно сформованих записів продуктів відносно усіх записів та частота появи кожного з можливих атрибутів валідації у звіті про неправильні продукти, що відображатиме те який з них і як регулярно стає причиною помилковості записів.

1.2. Запропонований підхід

Робочу програму для дипломного проєкту було вирішено зробити у вигляді системного модуля-розширення (так званого картриджа) для вбудованої у платформу SFCC утиліти Business Manager. Оскільки система має певні обмеження для роботи із базою даних, є лише два оптимальних підходи до розробки додатків, що матимуть до неї доступ: створення SFCC картриджу та використання OCAPI. В даному випадку було віддано перевагу реалізації у вигляді картриджу. При створенні системного модуля основною метою було забезпечити користувача зручним розширенням існуючого функціоналу системи. Для цього добре підходить картридж, оскільки з його допомогою можна додати пункт меню до утиліти Business Manager. Таким чином аналіз бази даних продуктів буде знаходитись у місці, де після

перегляду звіту можна одразу виправити помилки у записах. OCAPI зручно було б використати у випадку створення додатку, що використовує дані SFCC, але може бути незалежним від BM.

Функціональна частина картриджу написана мовою JavaScript. Було вирішено використати саме її, тому що, згідно з вимогами системи, бекенд частина розширень має бути реалізована нею, або мовою TypeScript. Я віддала перевагу мові JS в цьому виборі, оскільки у рамках розробки для платформи SFCC у переважній більшості випадків використовуються типи даних вбудовані у саму систему і проблема динамічної типізації JS нівелюється.

У розробці функціоналу обробки та аналізу бази даних продуктів було зроблено акцент на асинхронність роботи програми, тому що в масштабах реального сайту кількість даних може бути настільки великою, що синхронний аналіз гарантовано призведе до помилок, великого часу затримки та інших неочікуваних наслідків. Асинхронність реалізована за допомогою вбудованої у систему можливості автоматизувати рутинні завдання або довготривалі процеси – Job. Це найбільш оптимальне рішення проблеми, тому що воно максимально нативне та ідеально підходить під задачу обробки даних бази (це одна з основних цілей існування Job).

Інтерфейс картриджу створений мовою генератором шаблонів ISML, що використовується у SFCC платформі, мовою JavaScript та пре-компілятором мови стилізації SCSS. Для даної задачі інтерфейс має не таке велике значення, як функціональна частина, тому мені хотілось уникнути складнощів з інтеграцією JavaScript фреймворків на кшталт React.js. Із цим могли виникнути проблеми, оскільки картридж призначений для утиліти Business Manager, що має обмежену їх підтримку.

1.3. Огляд аналогів

Оскільки проблема, яку має на меті вирішити даний дипломний проєкт є досить індивідуальною та вузькоспрямованою, мені не вдалося знайти прямих аналогів розробки у офіційному сховищі (репозиторії) Salesforce Commerce Cloud, де зберігаються усі вільнодоступні розширення базового функціоналу. Проте, було знайдено декілька картриджів, основним завданням яких є вирішення недоліків бази даних платформи SFCC. Схожість полягає у тому, що вони роблять аналіз заданих атрибутів заданих сутностей і відображають звіт про це користувачу у певному вигляді.

Перший картридж аналог має назву Salesforce Commerce Cloud Business Manager Data Maintenance Extension, згідно з описом розробника, — це фреймворк, що має на меті покращити продуктивність та зручність Business Manager сайту, прибрати невизначеності даних сутностей. Наразі клієнтам SFCC та їхньому персоналу (продавцям та адміністраторам) дуже важко підтримувати чистоту своїх даних, оскільки в Business Manager немає вбудованої функції управління даними. Співробітники SFCC представили новий фреймворк Alert, який закладає основу для цього розширення Business Manager. Однак, поки що він використовується дуже рідко. При підключенні даний картридж висвітлює можливості очищення для клієнта безпосередньо на головній сторінці після входу в систему — підказуючи йому, як очистити свої дані, для того, щоб

- Усунути їх невизначеність.
- Підвищити продуктивність.
- Покращити роботу Business Manager [8].

У порівнянні із модулем створеним у рамках дипломного проєкту, вказаний картридж працює з іншими сутностями — Сайт та Організація. Він проводить валідацію їх даних та виводить результат у вигляді Business Manager повідомлень про знайдені помилки у записах. На відміну від

системного модуля проєкту він не має окремого інтерфейсу користувача і його основу складає асинхронний скрипт, що виконується з заданою періодичністю. У проєкті також використовується job, але функціонал додатку значно ширший і не обмежується лише скриптом, а надає користувачу зручну візуальну репрезентацію налаштувань та виводу результату. Функціонал Data Maintenance Extension більш вузький за розробку проєкту також тому, що не надає можливості вибору атрибутів для валідації та можливості фільтрації помилкових даних за атрибутами сутності. До того ж, він не працює із складними для обробки локалізованими атрибутами, а лише з атрибутами простих типів.

Наступний аналог — Order Guard Pipeline – це кастомний асинхронний скрипт, що дозволяє регулярно перевіряти створення замовлень на Production сервері. Він шукає останні створені замовлення і надсилає повідомлення на електронну пошту, якщо не знайдено жодного замовлення, вік якого не перевищує вказаний максимальний. Тобто, аналізує сутність Замовлення і якщо не знайдено жодного запису за заданий проміжок часу (тобто замовлення надходять із регулярністю меншою за очікувану), повідомляє користувача про це [9].

Order Guard Pipeline аналізує Замовлення, на відміну від розробки проєкту. Цей картридж має ті самі недоліки, що і попередній у порівнянні із картриджем створеним у рамках дипломного проєкту. Даний модуль має значно вужчий функціонал, не надає візуального інтерфейсу для вибору атрибутів перевірки та перегляду результатів, не працює з локалізованими атрибутами та відображає результат лише у вигляді повідомлення на електронну пошту.

Цей розділ висвітлює актуальність поставленої проблеми, особливості підходу до її вирішення, переваги створеної розробки у порівнянні із іншими схожими за концепцією. В наступному наводиться загальний опис архітектури

платформи SFCC та підхід до розробки нових системних модулів для неї для кращого розуміння подальших розділів.

2. ОПИС ПЛАТФОРМИ SFCC. СТВОРЕННЯ КАСТОМНИХ СИСТЕМНИХ МОДУЛІВ

2.1. Архітектура платформи SFCC

Salesforce Commerce Cloud, раніше відома як Demandware (до придбання корпорацією Salesforce), є хмарною платформою для реалізації електронної комерції, яка характеризується відносною простотою використання. Вона застосовується для просування великого бізнесу електронної торгівлі та забезпечує продавців та користувачів корисними можливостями та зручною взаємодією із нею. Акселератор SFCC надає можливості для реалізації міжрегіональних та мультибрендових рішень у сфері електронної комерції. Замовнику надається універсальний інструмент для забезпечення процесу торгівлі, яким можна успішно керувати. У користувачів платформи є можливість ефективної обробки створених клієнтами замовлень, а надані нею інструменти для обслуговування клієнтів сприяють плавному досвіду покупок для кінцевого споживача [10]. Для створення розширень до існуючого функціоналу варто бути обізнаним про архітектуру платформи. На рисунку 1 зображено її схему.

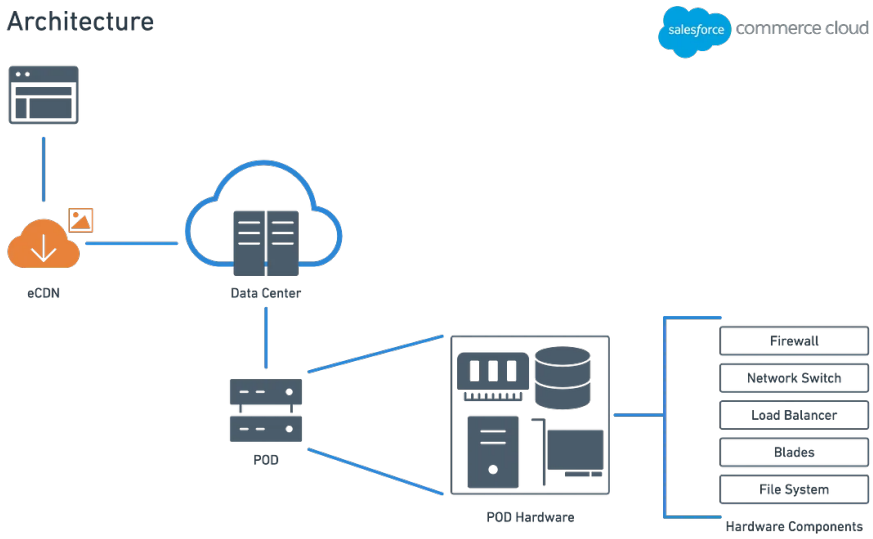


Рисунок 1. Схематичне представлення архітектури платформи SFCC

При доступі до веб-сайту, створеного на SFCC, клієнти отримують вигоду від використання Content Delivery Network (CDN) для доставки всього статичного контенту, включаючи JS, CSS і зображення. Для цього SFCC використовує вбудовану мережу постачання контенту eCDN (Embedded Content Delivery Network), а також дозволяє конфігурувати зовнішні CDN, використовуючи вбудовані можливості платформи. Такий стратегічний підхід гарантує, що платформа залишається необтяженою споживанням ресурсів, пов'язаних зі статичним контентом. Що стосується решти інформації та даних, то вони надійно зберігаються на серверах Salesforce, які розміщені в надійних і географічно розподілених центрах обробки даних, стратегічно розташованих у безпосередній близькості до кінцевих користувачів. У цих центрах обробки даних Salesforce застосовує структурований підхід, використовуючи кілька POD (Points of Delivery). POD - це кластери апаратних ресурсів, які спільно використовуються багатьма клієнтами Salesforce. Їх використання полегшує ефективне розподілення та доступність додаткових ресурсів у відповідь на підвищений попит. POD включають такі важливі компоненти, як брандмауери, комутатори, балансувальники навантаження, обчислювальні потужності та надійну файлову систему [10].

Усі нові проєкти, що реалізуються на платформі SFCC, орієнтовані виключно на вже існуючих великих клієнтів, річний дохід яких становить від 10 до 50 мільйонів доларів і більше. Відповідно, ці клієнти стикаються зі значною проблемою управління очікування величезної кількості відвідувачів веб-сайту, яка може сягати сотень тисяч. У періоди пікових навантажень, такі як “Чорна п'ятниця” або “Кіберпонеділок”, кількість відвідувачів може сягати навіть мільйонів. Майже в кожному проєкті на інтеграцію чекає вичерпний перелік сервісів і систем, що охоплює:

- Провайдери платіжних систем, включно з надійним механізмом боротьби з шахрайством (PSP).

- Провайдери податкових послуг для забезпечення відповідності чинному податковому законодавству.
- Системи управління замовленнями (OMS) для оптимізації процесу виконання замовлень.
- Системи управління взаємовідносинами з клієнтами (CRM) для підвищення рівня залучення та утримання клієнтів.
- Системи перевірки адрес для забезпечення точної та надійної інформації про доставку.
- Рішення для оцінювання та рецензування для збору відгуків і думок клієнтів.
- Аналітичні системи та інструменти для всебічного аналізу даних і звітності.

Багато з цих інтеграцій здійснюються за допомогою готових рішень, відомих як інтеграційні картриджі, які можна налаштувати відповідно до унікальних вимог і бізнес-потреб кожного клієнта. Однак важливо зазначити, що складність бізнес-процесів клієнта в поєднанні з потребою в глобальній підтримці, наприклад, мультибрендових рішень, що охоплюють Європу, Азійсько-Тихоокеанський регіон і Північну та Південну Америку, часто зумовлює необхідність розробки додаткових користувацьких функціональних можливостей. У таких випадках до кодування вдаються для створення специфічних інтерфейсів дизайну та планування, забезпечуючи безперебійну інтеграцію та оптимальну продуктивність, як і у випадку з будь-якою іншою технологією [10].

Впровадження рішень для електронної комерції починається з налаштування демо-версії магазину, яку Salesforce називає еталонною архітектурою магазину (Storefront Reference Architecture, SFRA) або еталонним додатком (Reference Application). Зазвичай клієнти використовують SFCC для створення рішень бізнес-моделі B2C, хоча є також випадки, коли

впроваджуються моделі B2B, B2B2C і навіть D2C. На етапі розробки клієнт отримує підтримку від спеціалізованої команди розробників, до складу якої входять UX та UI дизайнери, бізнес-аналітики та інші кваліфіковані фахівці. Ця команда тісно співпрацює з клієнтом, щоб гарантувати, що кастомізація демо-магазину відповідає його конкретним вимогам і цілям [7].

Salesforce пропонує потужну адміністративну функцію, відому як Business Manager (BM), розроблену спеціально для адміністрування платформи. У Business Manager користувачі мають широкий контроль над різними аспектами веб-сайту. Крім того, цей комплексний інструмент включає в себе модулі, які безпосередньо стосуються продажу продуктів і послуг. Ці модулі дозволяють керувати каталогами товарів, прайс-листами, інвентаризацією, контентом тощо [3]. На рисунку 2 зображено вкладку Merchant Tools, що у Business Manager слугує для зберігання вищезгаданих налаштувань.

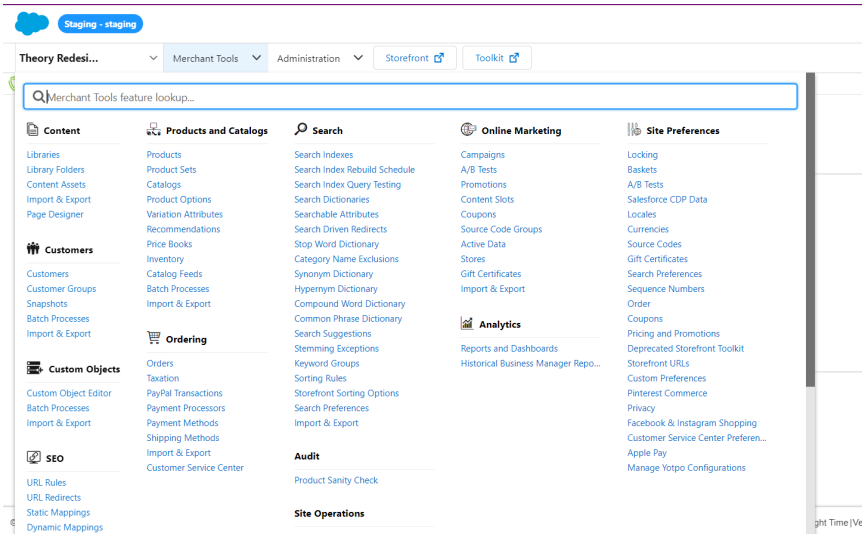


Рисунок 2. Вкладка Merchant Tools адміністративного інструменту Business Manager платформи SFCC.

У Business Manager (BM) є специфічні можливості, які безпосередньо пов'язані з розробкою та IT-операціями. Ці можливості охоплюють такі

функції, як Jobs, подібні до утиліти Cron Job Software Utility, що дозволяє планувати та виконувати автоматизовані завдання. Крім того, BM підтримує інтеграцію веб-сервісів та інших інструментів, які сприяють безперешкодній співпраці між різними системами та додатками. Business Manager надає користувачам можливість керувати створенням коду та реплікацією даних у різних середовищах (sandbox, development, staging, production). Він пропонує спрощений процес контролю та організації передачі коду і даних між різними середовищами. Крім того, Business Manager полегшує планування резервного копіювання, забезпечуючи збереження критично важливих даних. Адміністратор, який управляє Business Manager, має повноваження надавати доступ іншим користувачам і призначати їм певні ролі в системі. Це дозволяє ефективно співпрацювати та делегувати обов'язки. Більше того, адміністратор також може керувати дозволами, надаючи користувачам доступ до певних функцій і можливостей Business Manager відповідно до їхніх ролей і вимог [10].

На стороні сервера Salesforce Commerce Cloud використовує B2C Commerce JavaScript, який є двигуном JavaScript на основі Rhino. Цей двигун підтримує ECMAScript 5, розширення Mozilla, JavaScript 1.8 та E4X. Крім того, код на стороні сервера організовано за допомогою сумісних з CommonJS скриптових модулів B2C Commerce. Поточна версія серверного JavaScript SFRA має схожість з фреймворком Express Node.js, що забезпечує знайомий досвід розробки для тих, хто працював із Express. SFCC надає широкі можливості API для безперешкодної інтеграції з платформою:

- B2C Commerce API: включає в себе B2C Commerce Script API та B2C Commerce Pipelets API (застарілий).
- Open Commerce API.

B2C Commerce Script API пропонує повний набір класів, які безперешкодно взаємодіють з платформою, підтримуючи стандартні робочі процеси електронної комерції. Кожна окрема функціональність, така як

замовлення, кошики, клієнти, каталоги тощо, має свої власні спеціальні класи для ефективної роботи з ними. Важливо зазначити, що пряма взаємодія з базовою базою даних (БД) SFCC неможлива. Доступ до БД надається виключно через API SFCC, включаючи основну логіку платформи.

Такий обмежений доступ до БД пов'язаний з необхідністю збереження контролю над функціонуванням основної платформи, яка не є відкритою для безпосередньої модифікації. Кастомізація повинна здійснюватися у співпраці з існуючою інфраструктурою, як це характерно для будь-якого рішення Software-as-a-Service (SaaS). SFCC керує дизайном бази даних, взаємозв'язками між таблицями та пов'язаними з цим завданнями, звільняючи розробників від цього навантаження. Однак платформа дозволяє розширювати вбудовані системні об'єкти та впроваджувати нові кастомні функції роботи з даними, коли це необхідно. Виконуючи широкий спектр завдань, SFCC дозволяє розробникам зосередитися на створенні ефективних бізнес-рішень, а не на складнощах низькорівневої інфраструктури [10]. Спрощена схема взаємодії сутностей бази даних, важливих для розуміння роботи системного модуля проєкту зображена у додатку ІАЛЦ.045490.007 Д3.

Open Commerce API (OCAPI) – це RESTful (Representational State Transfer) API, який використовує зручну для користувача архітектуру веб-сервісів. Він складається з трьох основних компонентів:

1. Shop API: надає доступ до функціональності Commerce Cloud Digital Storefront. Він відіграє вирішальну роль у спрощенні процесу купівлі товарів клієнтами. Він надає можливості для доступу до продуктів, управління кошиками, створення замовлень та інших функцій, пов'язаних з клієнтами.
2. Data API: надає доступ до конфігурацій Commerce Cloud Digital та функцій інтеграції для пов'язаних проєктів. Він забезпечує безперешкодну інтеграцію з різними джерелами даних і системами.
3. Meta API: надає доступ до ресурсів та описів структур, доступних в Open Commerce API. Він містить інформацію про доступні ресурси та їхні

характеристики, допомагаючи розробникам ефективно використовувати API [10].

2.2. Процес створення та встановлення системного модуля для платформи SFCC

В даному дипломному проєкті під системним модулем мається на увазі так званий картридж. У Salesforce Commerce Cloud картридж — це структурована директорія, яка слугує універсальним механізмом для впровадження кастомізованої функціональності. Він охоплює різні типи файлів, включаючи статичні файли, такі як CSS і JavaScript, а також файли зображень. Крім того, картридж складається зі спеціальних директорій, призначених для файлів, специфічних для SFCC, таких як scripts, templates та визначення форм. Така організація структури дозволяє ефективно керувати та розгортати персоналізовані функції в середовищі Commerce Cloud Digital.

Як вже зазначалось у попередньому підрозділі, SFCC надає користувачам демо-версію сайту — SFRA. Для її встановлення необхідно додати до робочої директорії проєкту декілька дефолтних елементів, які слугуватимуть базою для майбутнього сайту:

1. app_storefront_base: основний картридж для впровадження демо-сайту SFRA, він є надзвичайно важливим, оскільки містить всі базові елементи коду для сайту.
2. bm_app_storefront_base: цей конкретний картридж Business Manager слугує як допоміжний додаток для інтеграції елементів утиліти Page Designer.
3. modules: розташована поряд з папками картриджів, папка модулів слугує основою для глобально доступних модулів у Commerce Cloud, включно з модулями сторонніх розробників.

Картридж app_storefront_base має найважливіше значення серед усіх інших картриджів. Він слугує сховищем для основного базового коду та

функціональності SFRA. Вкрай важливо утримуватися від внесення будь-яких змін до вмісту цього картриджа, як зазначають розробники SFCC посилаючись на правила найліпших практик розробки під платформу. Обґрунтування їх наполягання на збереженні цілісності цього картриджа стає очевидним, якщо розглянути його роль як еталонного для всіх інших картриджів. Уявімо ситуацію, коли є два кастомних картриджа, які покладаються на певний файл у базі app_storefront_base. Якщо буде змінено певний файл напряму у директорії app_storefront_base, ці зміни поширяться на всі картриджі, які його використовують. Тепер уявімо, що один з кастомних картриджів створено з урахуванням цих змін. У цьому випадку, здається, що все працює за призначенням. З іншого боку може статись, що другий користувацький картридж не розрахований на цю зміну. Отже, ця модифікація призведе до непередбачуваних наслідків у коді картриджа, що потенційно може призвести до збоїв у роботі системи. Серйозність ситуації посилюється, якщо врахувати, що така небажана поведінка може негативно вплинути на фінансові показники клієнта [11].

Давайте розглянемо кожну папку в картриджі та зрозуміємо їхнє призначення та вміст. Кожен картридж міститься в одній директорії. Усередині кожного картриджа є спеціальні вкладені директорії, призначені для зберігання певних типів файлів. Наприклад, усі файли Controller мають зберігатися в папці controllers. Усі теки розташовані у теці картриджа, дотримуючись простої структури:

- client. Папка client охоплює весь код, пов'язаний з клієнтською частиною, включаючи фронтенд JavaScript і SCSS. Ця папка, разом з папкою шаблонів, є основним об'єктом уваги фронтенд розробників.
- config. Тека config містить кілька конфігураційних файлів. Однак, зазвичай рекомендується не змінювати вміст цієї теки.

- controllers. У папці контролерів знаходяться всі endpoints з відповідною логікою, що використовуються сайтом магазину. Розробники бекенду переважно присвячують свій час цій папці.
- experience. У папці experience зберігаються файли визначень, що використовуються інструментом Page Designer.
- forms. У папці forms знаходяться всі файли визначень для форм. Ці файли необхідні для перевірки даних і визначення полів. Щоразу, коли створюється форма, її опис звичайно зберігається у файлі в цій папці.
- models. Тека models містить визначення об'єктів, які використовуються для передачі даних на сторінках сайту магазину та отримання даних про об'єкти з sandbox. Моделі широко використовуються в контролерах, що робить розуміння цих об'єктів надзвичайно важливим для бекенд-розробників, щоб ефективно налаштовувати та використовувати їх відповідно до вимог проєкту.
- pipelines. У папці pipelines міститься логіка бекенда, яка вважається застарілою і використовується на сайті вітрини магазина. Конвеєри - це графічні представлення (функціональні лише з плагіном Eclipse IDE) XML-файлів, які дотримуються певної структури. Під час роботи з SFRA ці файли рідко доводиться змінювати, оскільки їх було замінено на контролери.
- scripts. Salesforce B2C Commerce пропонує власну мову програмування сценаріїв, відому як DWScript. Ця мова програмування на стороні сервера інтерпретується сервером та інтегрується на сторінки магазину як JavaScript. Папка scripts містить всі допоміжні скрипти, які можуть бути використані контролерами, моделями, іншими скриптами і навіть темплейтами.
- static. Папка static містить усі статичні файли, створені в процесі компонування Webpack, наприклад, зображення для брендингу, скомпільовані файли JS та CSS. Ці файли навряд чи зміняться, їх не рекомендується модифікувати на пряму.

- templates. Папка templates містить не лише файли шаблонів, але й файли пакетів ресурсів (розширення .properties). Ці файли пакетів ресурсів дозволяють відображати написи та інший текст на сайті магазину кількома мовами. Як і у випадку з папкою client, саме тут розробники фронтенду в першу чергу зосереджують свою увагу [11].

Коли виникає потреба додати нову функціональність до вашого сайту-магазину, рекомендується створити новий картридж в рамках вашого проєкту. Картридж app_storefront_base слугує основою для магазину SFRA в Commerce Cloud Digital. Він включає в себе:

1. Бізнес-рівень, що складається з усіх серверних компонентів, таких як цифрові скрипти.
2. Базовий презентаційний рівень, що включає шаблони ISML, загальні файли CSS, форми та файли ресурсів.
3. Специфічні SCSS, JavaScript та розширені елементи інтерфейсу користувача, необхідні для створення бажаного вигляду та інтерфейсу магазину SFRA.

Існує два способи створення нового картриджа: створення вручну або використання node.js пакета sgmf-scripts. Розглянемо перший. Створення нового картриджа передбачає включення двох основних файлів:

1. .project: Цей файл визначає конфігурацію проєкту.
2. <назва_картриджа>.properties: Цей файл містить властивості, характерні для новоствореного картриджа.

Щоб створити новий картридж з назвою “app_storefront_custom” у папці картриджів, виконайте такі дії:

1. Створіть нову директорію в директорії cartridge і назвіть її “app_storefront_custom”.
2. Відкрийте файл “.project”, що знаходиться в картриджі “app_storefront_base”.
3. Скопіюйте вміст файлу “.project”.

4. Вставте скопійований вміст у файл “.project” всередині новоствореної папки “app_storefront_custom”.

5. У новоствореному файлі “.project” змініть вміст тегу “name”, щоб він відповідав назві вашого нового картриджа. У цьому випадку замініть “app_storefront_base” на “app_storefront_custom”.

6. Далі знайдіть файл “.properties” всередині картриджа “app_storefront_base”.

7. Скопіюйте файл “.properties”.

8. Вставте скопійований файл “.properties” у потрібне місце в папці “app_storefront_custom”.

9. Відкрийте скопійований файл “.properties” в папці “app_storefront_custom”.

10. Замініть усі входження “app_storefront_base” на назву вашого нового картриджа “app_storefront_custom”.

Виконавши ці кроки, ви створите новий картридж “app_storefront_custom” з необхідними файлами та зміненим вмістом. При створенні нового картриджа для додавання функціональності дуже важливо дотримуватися тієї ж структури папок, що і в картриджі “app_storefront_base”. Це гарантує стабільну функціональність і сумісність.

Коли йдеться про “шлях” в контексті поля форми, це означає список картриджів, які будуть використані в проєкті, розділений двокрапкою. Кожен сайт має свій власний унікальний шлях. Завантаження вашого коду до Sandbox є важливим кроком для забезпечення його функціональності на сайті вашого магазину. Однак це не єдина вимога. Щоб сайт міг використовувати певний картридж, ви повинні додати цей картридж до шляху сайту в Business Manager [11]. Щоб зробити це, слід виконати наступні дії:

1. Відкрийте Business Manager.
2. Перейдіть до Administration>Sites>Manage Sites>{назва сайту}>Settings.
3. Знайдіть поле “Cartridges”. Це поле відображає вищезгаданий шлях.

4. Додайте новий картридж до шляху картриджів, переконавшись, що він знаходиться перед картриджем “app_storefront_base”. Важливо зазначити, що якщо в різних картриджах є файли, що повторюються, то пріоритет матиме перший зустрічний в порядку, вказаному в полі “Cartridges”.

5. Переконайтеся, що для відповідного сайту вимкнено кешування сайту, щоб уникнути проблем, пов'язаних з ним.

Дотримуючись цих інструкцій, ви успішно налаштуєте шлях до картриджа для сайту в Business Manager, що дозволить сайту використовувати функціональність доданого картриджа.

Слід зазначити, що кастомний картридж не обов’язково має складатись лише з тих директорій, що є в базовому картриджі та перекривати існуючий функціонал платформи. Він може мати інакшу структуру та використовувати файли визначені у картриджах, що стоять після нього у шляху. Тобто, програміст у створеному картриджі “app_storefront_custom” може посилатись до тих файлів, що існують у “app_storefront_base” (але перед створенням посилань треба переконатись, що вони точно існують, інакше станеться помилка). Це можна зробити використовуючи спеціальні маски шляху. На стороні бекенду платформи SFCC можна використовувати такі маски, як * — посилання до усіх картриджів проєкту (до тих що знаходяться у шляху картриджів перед тим в якому застосовується посилання та включно із ним самим) та ~ — посилання до файлів виключно поточного картриджу. Наприклад, якщо програмісту треба скористатись функціоналом допоміжного файлу renderTemplateHelper, то він може використати такий шлях до нього: “*/cartridge/scripts/renderTemplateHelper”, що вказуватиме на відповідний файл, що лежить у поточному картриджі або в одному з картриджів, що йому передують у шляху визначеному в BM. Якщо ж він використає такий запис: “~/cartridge/scripts/renderTemplateHelper”, це означатиме, що файл має бути знайдений саме в поточному картриджі.

Отже, в даному розділі було розглянуто теоретичну інформацію необхідну для ознайомлення із платформою. В наступному розділі йдеться про основний алгоритм розроблений з метою вирішення задачі поставленої в дипломному проєкті.

3. АЛГОРИТМ РОБОТИ СИСТЕМНОГО МОДУЛЯ ДЛЯ АНАЛІЗУ СУТНОСТЕЙ БАЗИ ДАНИХ ПРОДУКТІВ У SFCC

3.1. Опис алгоритму аналізу сутностей продуктів бази даних SFCC

В даному розділі наведено детальний опис алгоритму, що працює при запуску аналізу сутностей БД продуктів SFCC платформи. Спрощена схема алгоритму наведена у додатку ІАЛЦ.045490.005 Д1.

Алгоритм бере початок у ендпоінті BMProduct-ExportReport (він спрацьовує, коли користувач натискає кнопку “Run the report”). Основна задача цього кроку — зберегти дані, що ввів користувач, щоб вони були доступні у Job та запустити виконання асинхронного скрипту. Збереження даних здійснюється за допомогою Custom Object, оскільки при запуску Job з програмного коду відсутня можливість передачі параметрів у скрипт. На цьому етапі відбувається наступне:

1. BMProduct-ExportReport викликає метод controller.createCOforJob, передаючи у нього дані вводу користувача.
2. Метод controller.createCOforJob робить транзакцію у БД у якій створює CO типу productSanityConfig з унікальним ідентифікатором, заповнює поля створеного CO інформацією від користувача, заповнює поле status значенням Processing (маркує об’єкт як той, що зараз обробляється) та повертає ідентифікатор створеного об’єкту.
3. BMProduct-ExportReport здійснює запуск Job 'Product Sanity Check' за допомогою використання системного класу Pipelet. Алгоритм роботи Job буде детально описано нижче. Далі ендпоінт переходить на наступний крок не чекаючи закінчення виконання Job, оскільки Job – це асинхронний скрипт.
4. Ендпоінт повертає ідентифікатор кастомного об’єкта, створеного після закінчення виконання методу controller.createCOforJob. Ці дані потрібні для здійснення можливості виводу результату у табличному вигляді. До цього повернемось після розгляду алгоритму роботи 'Product Sanity Check' Job.

Задача асинхронного скрипту — створити та експортувати звіт у файл та надіслати користувачу на електронну пошту повідомлення про готовність файлу (про те, що файл звіту сформовано і він готовий до встановлення на сторінці додатку). Після того, як BMProduct-ExportReport запустив виконання Product Sanity Check відбувається наступне:

1. Job перевіряє чи існує у БД сайту CO типу productSanityConfig зі значенням Processing атрибуту status. Якщо немає, не робить нічого, якщо є, переходить до наступного кроку.

2. Використовуючи дані знайденого CO формує об’єкт конфігурації, що буде переданий методу обробки у якості параметру. Цей об’єкт складається з полів: props — містить кастомні (такі, що створені користувачем БД) атрибути продукту, що мають бути перевірені під час аналізу (у звіті відображаються у колонці copy); columns — колонки, що мають відображатись у звіті (переважно це системні атрибути продукту або інших сутностей пов’язаних з продуктом таких як каталог, інвентар); productsToInclude — прапорець, що позначає які саме записи відображати у звіті (всі записи, тільки записи з невірною інформацією, тільки записи з вірною інформацією). Також в залежності від того як користувач передав інформацію про записи для перевірки цей об’єкт може містити: filePath — шлях до файлу формату .csv зі списком ідентифікаторів продуктів; pids — список ідентифікаторів продуктів, що користувач ввів з клавіатури; seasons — список значень атрибуту сезону до якого належать записи, що мають бути перевірені.

3. В залежності від того які дані записів БД продуктів були передані користувачем, визначає метод, яким буде виконуватись аналіз:

   а) Якщо користувач обрав файл зі списком ідентифікаторів продуктів (типу Master Product) для аналізу, то буде використано метод controller.parseFileAndGenerateReport;

b) Якщо користувач ввів вручну список ідентифікаторів продуктів (типу Master Product) для аналізу, то буде використано метод controller.generateReportForMultipleProducts;

c) Якщо користувач ввів значення атрибуту сезону тих продуктів (будь-якого типу), що потрібно проаналізувати, то буде використано метод controller.generateReportFilteredBySeasons;

4. Job викликає метод controller.writeAndExportReport(method, config) передаючи йому сформований об’єкт конфігурації та метод яким має виконуватись аналіз даних (нижче буде розписано алгоритм роботи кожного зі згаданих методів). Цей метод робить основну роботу з аналізу даних та формування звіту.

5. Після закінчення минулого кроку Job надсилає повідомлення користувачу про те, що його звіт готовий до перегляду на сторінці додатку.

6. Job робить транзакцію у бд, змінюючи значення атрибуту status СО типу productSanityConfig на Completed (маркуючи його як той, що обробився і може бути видалений).

Розглянемо призначення методу controller.writeAndExportReport, що викликається на 4 кроці роботи Job. Цей метод служить обгорткою для того, щоб передати методу генерації звіту екземпляр вбудованого класу CSVStreamWriter, що слугуватиме для того, щоб експортувати дані звіту у файл формату .csv. Для цього у ньому використано допоміжний метод fileHelper.getCSVStreamWriter. Він приймає шлях до файлу першим параметром, а другим — callback

функцію, що визначає дії, що мають відбутись з файлом після його успішного відкриття. Callback функція в свою чергу має три параметри: перший — успішно відкритий файловий потік типу CSVStreamWriter, другий — файловий потік типу FileWriter, третій — файловий об’єкт, що є екземпляром класу File. Цей метод є обгорткою для зручної роботи з файлами.

Його основним призначенням є відкриття файлу, запис до файлу даних оброблених певним чином (визначеним функцією параметром) та закриття файлу. Якщо на якомусь з етапів станеться помилка, метод відловить її, поверне статус з кодом помилки та запише у лог файл повідомлення про неї. Тепер розглянемо алгоритм роботи методу:

1. writeAndExportReport викликає метод fileHelper.getCSVStreamWriter. Усі подальші кроки виконуються всередині callback функції параметру.
2. Додає до об’єкту конфігурації поле csvStreamWriter, ініціалізуючи його значенням файлового потоку для запису, тобто значенням першого параметру callback функції.
3. Викликає функцію writeHeader, що згідно з об’єктом конфігурації записує у відкритий файловий потік заголовок звіту. Приймає першим параметром потік запису, а другим — об’єкт конфігурації звіту.
4. Відкриває ще один потік запису методом fileHelper.getCSVStreamWriter. Цей потік потрібен, щоб записати дані у іншому вигляді задля подальшого використання їх програмою з метою відображення звіту у табличному вигляді. Усі подальші кроки виконуються всередині callback функції параметру.
5. Додає до об’єкту конфігурації поле streamWriter, ініціалізуючи його значенням другого файлового потоку для запису, тобто значенням першого параметру callback функції.
6. Викликає метод для формування та запису звіту (що був визначений на 3 кроці роботи Job) з оновленим параметром конфігурації.

Далі детально розглянемо алгоритми роботи кожного з методів призначених для аналізу БД продуктів та запису результату аналізу у файл. З першого погляду може здатись, що логічніше було б розділити обробку даних та запис їх у файл. Проте, слід наголосити, що отримання даних та запис відбувається порядково (точніше, записуються дані про усі продукти типу Product Variant, що належать до запису Master Product, що було оброблено) і

такий підхід вимагає того, щоб узявши якусь частину даних одразу записати їх у файл (у тій самій функції). Він необхідний для того, щоб не утримувати у пам’яті об’єкт, що містить невідому кількість записів (на початку роботи програми неможливо проконтролювати розмір вихідного результату), оскільки це може призвести до перевищення допустимої квоти на розмір масиву і програма завершиться помилкою. Тобто не можна взяти масив одразу з усією інформацією (про усі Product Variations усіх Master Products, що обробилися під час роботи програми) і записати його у файл, це необхідно робити поступово, при отриманні зліченної кількості рядків з інформацією про продукти.

Метод controller.parseFileAndGenerateReport має на меті зробити аналіз файлу формату .csv з вхідними даними (має містити список ідентифікаторів тих записів продуктів, які необхідно проаналізувати), згенерувати об’єкт звіту з інформацією про задані продукти та записати її у два файли (перший потрібен для користувача, другий — для того, щоб відобразити звіт на сторінці у табличному вигляді). Як параметр приймає об’єкт конфігурації, сформований на основі даних, що ввів користувач (включно з іменем файлу, що має обробити метод) та містить вказівники на потоки запису обох файлів. Алгоритм роботи методу такий:

1. parseFileAndGenerateReport викликає допоміжний метод fileHelper.getCSVStreamReader, що має такі самі характеристики, як вищезгаданий getCSVStreamWriter, але створює файловий потік типу CSVStreamReader, у callback функції виконує зчитування файлу та необхідні дії з даними, що містяться у файлі (визначені цією функцією). Його callback функція також має три параметри, але перший і другий відрізняються типом файлового потоку (CSVStreamReader, FileReader). Всі основні подальші кроки метод parseFileAndGenerateReport виконує всередині callback функції.

2. Метод виконує цикл для зчитування рядків файлу. Всередині циклу відбувається наступне:

a) Значення наступного ідентифікатора Master Product, що знаходиться у файлі переписується у змінну.

b) Виклик функції generateReport, призначенням якої є формування об’єкту звіту з інформацією про усі продукти, що належать до Master Product заданого ідентифікатором. Функція приймає 4 параметри: ідентифікатор BP; масив з ідентифікаторами колонок, що мають бути відображені у звіті; масив з ідентифікаторами користувацьких атрибутів, що мають бути перевірені; прапорець зі значенням продуктів, що мають з’явитись у звіті. Вона повертає масив з об’єктами, що містять дані звіту по кожному Product Variation, що належить заданому Master Product, або масив з одним об’єктом, що містить поле invalidID, якщо не існує BP з заданим ідентифікатором. Нижче детальніше розглянемо роботу цього методу.

c) Два виклики функції exportReport призначеної для запису у файл даних, що повернула попередня функція. Ця функція приймає 3 параметри: масив зі звітом (що повернула функція generateReport), callback функцію (що буде використана для запису даних) та булеве значення (що вказує чи це файл згенерований для клієнта). У обох викликах перший параметр однаковий, бо для табличного і файлового вигляду дані однакові, а два інших відрізняються. У першому випадку передається callback, що записує дані у чистому вигляді і прапор, що вказує, що дані мають бути перетворені у пасуючий для .csv запису формат (завдяки цьому прапору всередині exportReport викликається функція getCsvData, що робить це форматування). У другому виклику передається тільки callback функція, у якій дані записуються у форматі JSON, для подальшого зручного їх використання при створенні таблиці.

3. Останній крок відбувається поза межами callback функції, викликаної у пункті 1. Тут з внутрішньої файлової системи видаляється файл, що користувач надав у якості вхідних даних.

Метод controller.generateReportForMultipleProducts створює та записує у файл дані звіту на основі масиву ідентифікаторів ВР, сформованого з даних вводу користувача. Параметром приймає те саме, що і попередній метод, тільки цього разу об’єкт з даними необхідними для функції містить дані введених ідентифікаторів (у вигляді рядка зі значеннями ідентифікаторів розділених символом «|») замість імені вхідного файлу. Його алгоритм наступний:

1. Обробка вхідних даних: перетворення рядка розділеного вказаним символом на масив методом split.
2. Створення циклу по усіх елементах одержаного масиву. У циклі:

a) Виклик функції generateReport (опис призначення якої знаходиться вище) та запис повернутого значення у змінну.

b) Два виклики функції exportReport (особливості кожного з викликів описані вище) для запису файлу для користувача та системного файлу для відображення таблиці.

Метод controller.generateReportFilteredBySeasons створює звіт про продукти відфільтровані засновуючись на значенні вводу атрибуту season користувача, і записує цей звіт у два файли (як і попередні два розглянуті методи). Ввід атрибуту season в даному випадку — це рядок значень атрибуту сезону продукту (product.custom.season), які користувач обрав для перевірки, розділених символом «|». У роботі методу використовується вбудована у SFCC модель пошуку ProductSearchModel. Було обрано саме цей підхід для фільтрації продуктів, оскільки він працює з використанням пошукових індексів, що робить його дуже швидким та відповідним даній задачі. Алгоритм цього методу такий:

1. Налаштування і виконання пошуку. При налаштуванні пошуку задається атрибут сезону за яким треба знайти продукти, а також атрибут сортування за ідентифікатором ВР. Сортування потрібне для кроку 3.

2. Створення циклу по знайдених записах продуктів. Всередині циклу:

a) Перевірка чи поточний продукт є BP. Оскільки усі методи для створення звіту починають роботу саме з BP, а вже з них беруть дочірні VP, треба впевнитись, що у функції обробки як параметр потрапить саме BP. Тут алгоритм перевіряє чи поточний продукт належить до типу BP (використовуючи атрибут булевого типу master):

- Якщо ні, то йде наступна перевірка, чи збігається ідентифікатор BP поточного продукту (у кожному дочірньому продукті є атрибут вказівник masterProduct на його батьківський продукт типу BP) з ідентифікатором останнього обробленого BP. Якщо збігається (тобто цей BP вже був оброблений до цього), то відбувається перехід на наступну ітерацію циклу. Якщо не співпадає, то ми записуємо ідентифікатор цього BP у змінну, що утримує значення останнього обробленого BP і переходимо далі по алгоритму записавши потрібний BP у змінну product.
- Якщо так, то перевіряється чи збігається ідентифікатор цього продукту з останнім обробленим BP. Якщо так, то відбувається перехід на наступну ітерацію циклу.

b) Після попередніх маніпуляцій ще не оброблений BP гарантовано зберігається у змінній product. І тепер викликаються дві функції: getDataForReport та formatReportData. Вони удвох працюють практично так само як вищезгадана функція generateReport, але тут було використано їх поєднання замість неї, тому що вона приймає ідентифікатор продукту в якості параметра, створює продукт і перевіряє чи існує продукт з таким ідентифікатором, а потім викликає getDataForReport та formatReportData з цим продуктом. В даному випадку продукт вже гарантовано існує, тому виклик функції-обгортки не потрібен. Отримуємо той самий результат, що і після виклику generateReport.

с) Далі, як і у попередніх двох методах, робиться два виклики методу exportReport, записуючи отримані дані аналізу у файл для користувача і файл для подальшої обробки.

Далі детально розглянемо алгоритм роботи вищезгаданого методу generateReport, метою якого є створення масиву з об’єктами, що містять інформацію про результат аналізу продуктів згідно із заданими параметрами. Функція приймає 4 параметри: ідентифікатор BP, що треба перевірити; масив з ідентифікаторами колонок (атрибутами згідно з якими треба валідувати продукт), що треба відобразити у звіті; масив з ідентифікаторами кастомних атрибутів, наявність яких треба перевірити у записах продуктів; рядок ідентифікатор виду продуктів, які мають бути відображені у звіті, може приймати три значення: all – всі записи, над якими було здійснено перевірку, invalid – ті записи, які мають помилкове значення щонайменше в одному з атрибутів валідації, valid – ті записи, що не мають помилок у жодному з атрибутів перевірки.

1. Першим кроком алгоритму є перевірка введеного користувачем ідентифікатора. Функція робить запит у базу даних на продукт із заданим ідентифікатором за допомогою Script API dw.catalog.ProductMgr.
2. Далі слідує умовний вираз для перевірки того чи було знайдено продукт із заданим ідентифікатором. Якщо ні, то функція повертає об’єкт із полем invalidID, що містить переданий ідентифікатор (при формуванні звіту це використовується для того щоб відобразити користувачеві інформацію про помилковий ввід даних) і цим припиняє своє виконання. Якщо продукт було знайдено, то алгоритм переходить до наступного кроку.
3. generateReport викликає функцію getDataForReport, передаючи їй знайдений продукт та параметри налаштувань звіту (масив з ідентифікаторами колонок, масив з ідентифікаторами кастомних атрибутів, рядок ідентифікатор

виду продуктів). Згідно аргументів ця функція поверне або пустий масив або масив з об’єктами, що містять дані валідації VP, що належать заданому BP.

4. Функція викликає метод formatReportData, передаючи йому результат валідації отриманий на попередньому кроці та ідентифікатор продукту, що обробився. Цей метод слугує для того, щоб відформатувати дані валідації у зручному вигляді для відображення звіту.

Слід також детально розглянути функцію getDataForReport. Ця функція приймає 4 параметри: екземпляр класу Product, масив з ідентифікаторами колонок, масив з ідентифікаторами кастомних атрибутів, рядок ідентифікатор виду продуктів. Покроковий алгоритм її виконання такий:

1. Створення пустого масиву, що буде слугувати для збереження результату валідації.

2. Перевірка чи належить переданий параметром продукт до типу BP та чи містить він у собі записи VP. Якщо хоча б одна з вищезгаданих умов не виконується, то алгоритм закінчується на цьому кроці, відбувається перехід на крок 8 і функція повертає пустий масив. Якщо продукт задовольняє обидві умови, алгоритм переходить до наступного кроку.

3. Створюється цикл, що проходить по усіх VP заданого BP. Наступні кроки виконуються всередині цього циклу.

4. Викликається допоміжний метод productHelper.getVarGroupProductFromVariant(productMaster, VariantID), метою якого є отримати екземпляр VG із заданого екземпляру BP та ідентифікатору поточного VP, що належить до шуканої групи.

5. Створюється екземпляр кастомного класу Validator, а за ним екземпляр кастомного класу Product. Клас Validator призначений для виконання валідації полів, що містить клас Product. Клас Product в свою чергу потрібен для того, щоб взяти із записів продуктів бд необхідну для валідації інформацію та зберегти її у своїх полях в зручному вигляді для подальшої обробки класом

Validator та для виводу інформації у звіті. Конструктор класу Product приймає такі параметри: об’єкт, що містить екземпляри BP, VG та VP, що мають обробитись на поточній ітерації циклу; масив з ідентифікаторами колонок; масив з ідентифікаторами кастомних атрибутів. Детальний опис структури та призначення згаданих класів буде наведено нижче.

6. Викликається метод validate створеного екземпляру класу Product, який параметром приймає екземпляр класу Validator. Це потрібно для реалізації патерну проєктування visitor, далі буде описано детальніше. Після цього виклику екземпляр класу Validator у своєму полі invalidFields міститиме масив з інформацією про усі поля, де в записі містяться помилки. Якщо поле міститиме пустий масив, значить у даному записі продукту немає помилок у заданих атрибутах валідації.

7. У кінці циклу створеного на кроці 3 відбувається перевірка параметру рядка ідентифікатора виду продуктів. Якщо цей параметр дорівнює значенню all (тобто всі записи мають бути у звіті), або значенню invalid (тільки записи з помилками) і поле invalidFields екземпляру класу Validator не пусте (масив містить хоча б один елемент з інформацією про помилку), або значенню valid (тільки безпомилкові записи) і поле invalidFields пусте (не знайдено жодної помилки у записах), то у масив результату додається об’єкт з полями product та result. Поле product ініціалізується екземпляром кастомного класу Product, а поле result – значенням поля invalidFields екземпляру класу Validator. Таким чином один об’єкт, що є елементом масиву, містить дані валідації одного VP, що є дочірнім продуктом переданого параметром BP. Після цього починається наступна ітерація циклу.

8. Після виходу з циклу функція повертає масив з об’єктами, утвореними під час своєї роботи, або, якщо продукт переданий параметром не задовільнив умові на 2 кроці алгоритму, то повертає пустий масив.

Коротко про метод formatReportData. Як вже було вище згадано, він слугує для форматування масиву об’єктів, що повернула функція getDataForReport. Немає сенсу описувати алгоритм його роботи, проте слід зазначити що саме цей метод повертає в залежності від переданих йому параметрів. Він приймає масив об’єктів з інформацією про валідацію записів та рядок, що містить ідентифікатор продукту, що обробився. В цьому методі виконуються такі основні логічні кроки: перевірка даних на існування та форматування даних. Якщо перший параметр (дані звіту) містить пустий масив, то даний метод повертає другий параметр (ідентифікатор продукту), що вказує на те, що оброблюваний продукт або має невірний тип (не BP) або не містить VP. Якщо перший параметр не пустий, то відбувається перетворення даних у вигляд зручний для їх виводу.

Останнє, що варто розглянути в контексті опису алгоритму аналізу сутностей продуктів бази даних SFCC це допоміжні класи створені для полегшення обробки даних (що використовуються під час роботи функції getDataForReport). Всього з цією метою було створено 7 класів, 6 з яких для збереження та підготовки даних до валідації та виводу і 1 власне для валідації. Діаграма створених класів зображена у додатку ІАЛЦ.045490.006 Д2. Класи для роботи з даними — це по суті і є ті самі «колонки» з атрибутами для перевірки, які обирає користувач. Якщо користувач не задасть жодного атрибуту для валідації, то буде створено лише один з цих класів – Product, тому що він є базовим і містить у собі ідентифікатори оброблених записів.

1. Product – клас, що містить базову інформацію про продукт бази даних типу VP. Його призначенням, окрім утримування базової інформації для валідації, є згідно із переданими переметрами (об’єкт, що містить екземпляри BP, VG та VP; масив з ідентифікаторами колонок; масив з ідентифікаторами кастомних атрибутів) створити екземпляри решти класів та записати їх у один з своїх атрибутів. Одними з його атрибутів є ідентифікатори VP, VG до якої

він відноситься та батьківського ВР. Ці атрибути є об’єктами, що містять поля value – рядок із значенням ідентифікатора продукту, invalidReason – масив заповнюваний під час валідації (пустий, якщо атрибут пройшов перевірку, або заповнений значенням, якщо не пройшов), UUID – рядок із унікальним ідентифікаційним номером запису у базі, потрібний для того щоб створити посилання на продукт у ВМ при виводі звіту в табличному вигляді. Клас Product також має атрибут includedColumns, що утримує в собі об’єкт з екземплярами інших класів цього типу. Він має метод fillIncludedColumns, що приймає ті самі параметри, що і конструктор, призначений для того, щоб згідно із даними запиту користувача сформувати екземпляри класів з тими атрибутами, що необхідно проаналізувати.

2. Category – клас, що містить інформацію про ідентифікатор категорії Demandware класу Category, до якої присвоєно продукт. Його атрибутами при поточній реалізації можуть бути primaryID, або classificationID в залежності від того чи обрав користувач колонку для перевірки, що відповідає цим атрибутам. Вони є об’єктами, що містять поля value – рядок із значенням ідентифікатора категорії (primary або classification а залежності від того який саме екземпляр створюється) та invalidReason – масив заповнюваний під час валідації (пустий, якщо атрибут пройшов перевірку, або заповнений значенням, якщо не пройшов). Атрибути primaryID та classificationID можуть вважатись невірними в тому випадку, якщо їх не існує, тобто продукт не призначений категорії цього типу.

3. Image – клас, що містить інформацію про види картинок, що містить запис VG продукту. Можна сказати, що цей клас робить перевірку на прихований атрибут — наявність заданих варіацій картинок у продукті. Цей атрибут напряму не можна «дістати» з об’єкту Demandware класу Product. Щоб перевірити наявність певних варіацій картинок даний клас має такі поля: standardVersions – масив з рядками, що позначають ті варіації картинок, що за умовою замовника мають бути присутніми у кожного продукту на сайті;

invalidReason – масив, де зберігатимуться помилки в цьому атрибуті після валідації; versions — масив з рядками, що позначають варіації картинок, що присутні у даному записі. Для ініціалізації поля versions використовується функція getProductImageVersions, що приймає VG як параметр. Якщо VG не існує, то у поле versions записується значення null. getProductImageVersions, використовуючи ще декілька функцій, проходить по усіх екземплярах Demandware класу MediaFile, що призначені заданій VG, аналізує їх атрибути URL та записує усі знайдені в них ідентифікатори типу картинки у масив. Атрибут, що описує цей клас вважається невірно сформованим, якщо не всі варіації картинок визначені умовою (містяться у полі standardVersions) присутні у записі продукту (у полі versions).

4. Inventory – клас, що містить числове значення доступної для придбання кількості заданого VP, що визначена у класі ProductInventoryRecord. Цей клас має два атрибути: invalidReason – міститиме результат валідації даного класу; quantity — кількість заданого продукту доступна до придбання.

5. ProductCustomAttributes – клас, що утримує дефолтні та локалізовані значення кастомних атрибутів продукту. Цей клас обробляє ті атрибути, що користувач вказує у полі вводу, перевіряючи чи ініціалізовані вони та чи мають локалізовані значення (якщо користувачем обрано відображати перевірку локалізації та локаль існує на сайті). Обробка локалізованих атрибутів Demandware продукту, що виконується при створенні екземпляру даного класу є нетривіальною задачею, оскільки визначення наявності локалізованого значення певного атрибуту з програмного коду вимагає виконання багатьох маніпуляцій з даними: перевірка наявності локалі на сайті, створення «глибокої» копії значень атрибутів для дефолтної локалі, зміна локалі запиту, перевірка на еквівалентність значення атрибуту заданого для дефолтної локалі із значенням для поточної локалі (якщо вони співпадають, можна зробити висновок, що значення атрибуту для поточної локалі не задано). Атрибутами даного класу є ті кастомні атрибути продукту, що

користувач задав у відповідному полі (якщо їх було задано, у іншому випадку даний клас не буде створено), а також у окремих атрибутах даного класу зберігаються їх локалізовані значення. До того ж, даний клас має допоміжний атрибут invalidReason необхідний для валідації.

6. ProductSystemAttributes – клас, що містить системні атрибути продукту. При поточній реалізації, описує лише один атрибут — онлайн статус, але для кожного типу продукту бд: VP, VG, BP. Тобто даний кастомний клас має такі поля: variantOnline, groupOnline, masterOnline. Кожне поле ініціалізовано об’єктом, що містить у собі: value – прапор булевого типу, що вказує на статус продукту та invalidReason. Записи продуктів вважаються невірними при валідації, якщо в даних полях значення value дорівнює false.

Така побудова класів була створена тому, що по суті усі класи цього типу утримують у собі певні атрибути одного й того ж оброблюваного продукту і тому мають зберігатись у одному місці. А розділення їх було виконано за логічними функціями з метою полегшення процесу валідації, а також, щоб спростити створення нових валідаційних атрибутів (колонок звіту) при необхідності. Кожен з цих класів має метод validate, в якому викликає відповідний метод у екземпляра класу Validator. Таким чином реалізовано патерн проєктування visitor [5]. Наприклад, клас Product у своєму методі validate викликає метод validateProduct передаючи йому свій екземпляр. І в цьому validateProduct відбувається валідація тих полів, що містить Product, а саме ідентифікаторів. Ідентифікатор продукту може бути не валідним лише у тому випадку, якщо його не існує (а отже не існує і продукту). Цей конкретний сценарій можливий тільки для VG, оскільки якщо б не існувало BP та VP, то клас Product не було б створено. А от VP може бути не призначений жодній VG, а отже її може не існувати. Також у своєму методі validate клас Product викликає аналогічні методи validate для усіх створених ним екземплярів класів, що містяться у його атрибуті includedColumns (проходячи по них циклом). Таким чином одним викликом виконується валідація усіх атрибутів.

Розглянемо клас Validator, призначенням якого є виконання валідації полів, описаних у вищезазначених класах. Цей клас описує методи для аналізу кожного класу з даними: validateProductCustomAttr, validateProductSysAttr, validateProduct, validateInventory, validateCategory, validateImage. Короткий опис усіх методів у тому порядку в якому до цього було описано інші класи:

1. validateProduct — метод призначений для аналізу атрибутів класу Product. Він аналізує поле value кожного з атрибутів (variantID, groupID, masterID) і якщо воно виявляється пустим, додає у атрибут invalidFields екземпляру класу Validator та у поле invalidReason атрибуту {productType}ID (що мають тип Array) рядок із значенням “missing {productType}”, де productType – це той тип продукту, в якого немає ідентифікатора, тобто якого не існує (як було описано вище, це може бути тільки Variation Group).
2. validateCategory — метод призначений для аналізу атрибутів класу Category. Він працює так само, як і попередній, але проводить валідацію поля екземпляру класу Category (тобто classificationCategory або primaryCategory), додаючи у атрибут invalidFields екземпляру класу Validator та у поле invalidReason атрибуту {categoryType}Category рядок із значенням “{categoryType} category is not assigned to master product”, де categoryType – це той тип категорії, яка не визначена для даного запису BP.
3. validateImage — метод призначений для аналізу атрибутів класу Image. Спочатку у цьому методі робиться перевірка на те, чи пустий масив, що міститься у атрибуті versions даного екземпляру класу Image. Якщо пустий, то у поле invalidReason екземпляру Image та у поле invalidFields екземпляру Validator додається рядок “images are not set for variation group” та метод завершує своє виконання. Якщо ж не пустий, то метод циклом проходить по усіх значеннях ідентифікаторів картинок, що мають бути в кожному продукті (по масиву розміщеному в полі standardVersions екземпляру класу Image) та перевіряє поле versions — масив зі значеннями ідентифікаторів тих картинок, що присвоєно даному запису. Якщо поточного елементу standardVersions не

існує у versions, у поле invalidReason екземпляру Image та у поле invalidFields екземпляру Validator додається рядок “missing {version} image version”, де version – поточний елемент циклу.

4. validateInventory — метод призначений для аналізу атрибутів класу Inventory. Якщо поле quantity екземпляру класу Inventory має значення нуль, у поле invalidReason екземпляру Inventory та у поле invalidFields екземпляру Validator додається рядок “out of stock”.

5. validateProductCustomAttr — метод призначений для аналізу атрибутів класу ProductCustomAttributes. У ньому в циклі перевіряється кожен атрибут екземпляру класу ProductCustomAttributes і якщо для якогось з них значення не задане, то у поле invalidReason екземпляру ProductCustomAttributes та у поле invalidFields екземпляру Validator додається рядок “missing {attributeName}”, де attributeName – це назва атрибуту, що не пройшов перевірку.

6. validateProductSysAttr — метод призначений для аналізу атрибутів класу ProductSystemAttributes. Він аналізує поле value кожного з атрибутів (variantOnline, groupOnline, masterOnline) і якщо воно має значення false, додає у атрибут invalidFields екземпляру класу Validator та у поле invalidReason атрибуту {productType}Online рядок із значенням “{productType} offline”, де productType – це той тип продукту, що має статус офлайн.

3.2. Опис алгоритму для створення статистики по усіх продуктах сайту

Частина програми, що відповідає за відображення статистики по усіх продуктах сайту використовує ті самі інструменти для отримання даних, що були описані вище. На рисунку 3 зображено результуючі діаграми, що відображають результат роботи цієї частини програми. Для неї було створено окрему Job і запрограмовано регулярний її запуск раз на день, щоб щодня отримувати свіжі дані статистики. Також при необхідності її завжди можна

вручну запустити за допомогою ВМ. Однак, треба мати на увазі, що оскільки скрипт має збирати статистику по усіх продуктах сайту, виконання її може зайняти досить довгий час (в залежності від об'єму даних, на обробку ~ 15 тис. продуктів витрачається 12-16 хвилин часу). При старті Job запускається скрипт generateStatistics. Ця job приймає параметром ідентифікатор категорії, що містить усі ВР даного сайту. Це необхідно для того, щоб не робити додаткових перевірок на тип продукту перед запуском аналізу і зекономити час виконання одразу маючи тільки необхідні записи. Алгоритм роботи її скрипту такий:

1. Створення об'єкту для збору статистики із вхідними даними. Він містить такі поля: productsNumber — числовий тип, загальна кількість продуктів сайту; invalidProductsNumber — числовий тип, кількість продуктів сайту, що мають помилки у даних принаймні одного атрибуту; failingColumnsCount — об'єкт, що містить ідентифікатори атрибутів, що мають пройти перевірку, всі його поля числового типу. Усі значення числового типу спочатку ініціалізуються нулем.

2. Виконання запиту до dw.catalog.CatalogMgr API з метою отримати категорію із ідентифікатором, що задано параметром.

3. Створення циклу по всіх елементах колекції, шо міститься в атрибуті onlineProducts отриманої на 2 кроці категорії.

4. Всередині циклу відбувається виклик методу controller.updateStatistics, що приймає два параметри: об'єкт для конфігурації статистики та об'єкт статистики створений на кроці 1. Для розуміння того як саме обробляється об'єкт статистики далі буде наведено алгоритм роботи цього методу.

5. controller.updateStatistics, як вже було вказано приймає параметром об'єкт конфігурації. Він має такі поля: product — містить вказівник на об'єкт продукту, що має обробитись на поточній ітерації циклу; productsToInclude — ідентифікатор типу продуктів, що мають обробитись на поточній ітерації, оскільки для відображення статистики треба відобразити загальну інформацію

по усіх продуктах і немає потреби у заданні налаштувань звіту користувачем, значення ідентифікатора встановлено в all; columns – об’єкт з ідентифікаторами усіх доступних для перевірки атрибутів; props — об’єкт з ідентифікаторами тих кастомних атрибутів продукту, які за умовою вважаються необхідними для перевірки за замовчуванням (цей параметр визначається вимогою замовника). Першим своїм кроком метод викликає два методи getDataForReport та formatReportData з параметрами, що містяться у об’єкті конфігурації. Тут було викликано ці два методи без обгортки у вигляді функції generateReport, тому що продукт немає потреби перевіряти на існування, він гарантовано існує і має тип BP, завдяки попереднім маніпуляціям. Таким чином на цьому кроці генеруються дані для подальшої обробки.

6. До поля productsNumber об’єкту статистики додається значення довжини масиву з даними отриманого на попередньому кроці, тому що кожен елемент цього масиву являє собою об’єкт з даними для одного запису продукту типу PV.

7. Створення циклу по об’єктах, що містяться в масиві з даними отриманому на 5 кроці. Подальші дії відбуваються всередині циклу.

8. Створюється прапор булевого типу, що позначає чи запис продукту не валідний, спочатку ініціалізується значенням false. Після цього слідує цикл по усіх ключах поточного об’єкту з інформацією аналізу продукту. Ключами цього об’єкту є ідентифікатори колонок, а значеннями — об’єкти з полями value та invalidReason.

9. Перевірка того чи поле invalidReason запису з поточним ідентифікатором колонки не пусте. Якщо це поле не пусте, значить запис не пройшов перевірку для атрибуту з відповідним ідентифікатором. Отже, у об’єкт, що міститься у полі failingColumnsCount об’єкта статистики, у поле з ключем, що дорівнює ідентифікатору не валідної колонки додається значення один. Таким чином прослідковується кількість разів коли той або інший

атрибут валідації став причиною помилковості усього запису. Якщо хоча б один раз вищезгадана умова виконалась, тобто хоча б один атрибут виявився неправильним під час аналізу, то прапор ідентифікуючий помилковість продукту створений на кроці 8 ініціалізується значенням true. На цьому цикл по ключах об’єкту запису переходить на наступну ітерацію.

10. Після завершення циклу по ключах об’єкту запису слідує перевірка на те чи прапор помилковості даних поточного продукту встановлений у true. Якщо так, то до поля invalidProductsNumber об’єкта статистики додається 1, що означає збільшення кількості помилкових записів. На цьому цикл по об’єктах масиву із даними аналізу створений на кроці 7 переходить на наступну ітерацію.

11. Коли вищезгаданий цикл по масиву з даними закінчився, закінчується і виконання функції controller.updateStatistics. Далі продовжуватиметься опис алгоритму Job GenerateStatistics.

12. Після того, як функція controller.updateStatistics була виконана для кожного продукту категорії, відбувається обробка даних, що містяться в об’єкті статистики. У об’єкті створюється поле invalidProductsPercentage, що ініціалізується значенням відношення числа, що міститься у полі invalidProductsNumber до загального числа продуктів сайту, що міститься у полі productsNumber.

13. Далі у об’єкті статистики створюється поле invalidReasonsCount ініціалізуючись значенням числового типу, що дорівнює сумі усіх чисел, що містяться в полях об’єкту поля failingColumnsCount. Тобто загальному числу помилок в усіх атрибутах усіх продуктів сайту.

14. Створення поля failedColumnsPercentage у об’єкті статистики. Це поле ініціалізується об’єктом, ключами якого є ненульові ключі об’єкта поля failingColumnsCount. Значення асоційовані з ключами дорівнюють співвідношенню числа, що за таким самим ключем міститься у

failingColumnsCount до загального числа усіх помилок в усіх записах — invalidReasonsCount.

15. Останнім кроком алгоритм GenerateStatistics Job створює транзакцію у бд записуючи у поле data CO типу productSanityStatistics сформований об'єкт статистики перетворений у рядок формату JSON. У подальшому цей рядок перетворюється назад на об'єкт та використовується у системному модулі з метою відмальовки діаграми.

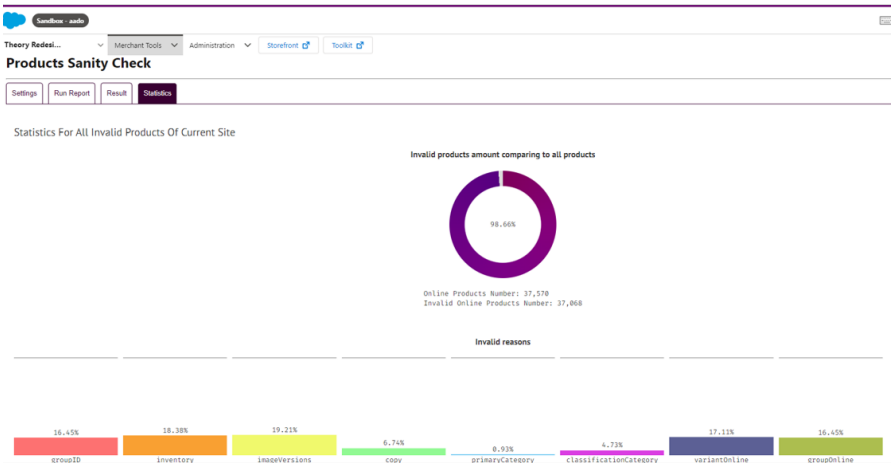


Рисунок 3. Вкладка статистики додатку, що відображає діаграми за результатом аналізу записів усіх продуктів сайту.

3.3. Опис взаємодії клієнтської та серверною частин модуля

Схема взаємодії усіх функціональних розділів створених у рамках проєкту зображена у додатку ІАЛЦ.045490.008 Д4, в даному підрозділі розглядається взаємодія клієнта із сервером. У роботі модуля використовується підхід server side rendering, це означає, що сервер відповідає за відображення клієнтської частини, формуючи її засновуючись на вхідних даних необхідних для відображення. При відкритті сторінки системного модуля у BM відбувається запит на ендпоінт BMProduct-Audit. Метою цього ендпоінта є рендеринг search.isml файлу (так званого темплейту). Він містить

розмітку сторінки додатку виконану мовою шаблонізаціїї ISML, тобто HTML елементи та деякі дозволені логічні конструкції, що спрощують створення розмітки веб сторінки. BMProduct-Audit готує дані для відображення згаданого темплейту, такі як: Demandware об’єкт форми, що слугуватиме для передачі вхідних даних для створення звіту від клієнта до сервера при виконанні запитів; посилання на ендпоінт призначений для генерації файлу звіту (BMProduct-ExportReport, алгоритм його роботи описаний вище) та на ендпоінт призначений для того, щоб відмалювати звіт у табличному вигляді; масив з екземплярами SFCC класу ObjectAttributeDefinition, що містить усі кастомні атрибути сутності продукту і необхідний для того, щоб надати користувачу можливість обирати їх для формування звіту; масив із значеннями атрибуту сезону представленими екземплярами DW класу EnumValue, що використовуватимуться в темплейті для надання користувачу можливості обрати записи для перевірки за допомогою потрібних значень атрибуту сезону; масив з рядковими ідентифікаторами кастомних атрибутів, що мають бути перевірені за замовчуванням; масив із посиланнями на скачування доступних файлів звітів (після створення файли звіту зберігаються деякий час у бд, щоб користувачі могли інсталювати їх у потрібний момент); масив з ідентифікаторами колонок (атрибутів) доступних для перевірки під час створення звіту; об’єкт статистики для відображення діаграм звіту по усіх продуктах сайту.

В попередніх розділах вже було опосередковано описано, що користувач має можливість обирати ті атрибути, що відображатимуться в звіті, але тепер зупинимось на цьому детальніше. На рисунку 4 зображено інтерфейс першої вкладки додатку.

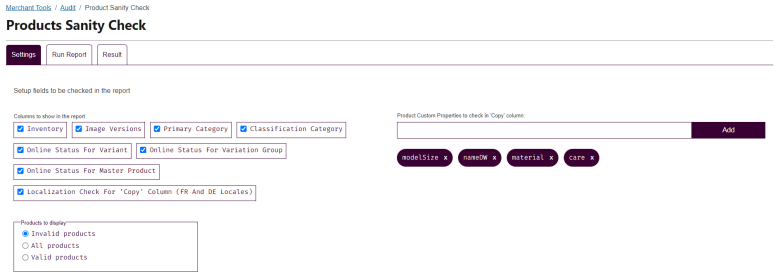


Рисунок 4. Інтерфейс вкладки налаштувань звіту системного модуля для аналізу сутностей продуктів бд SFCC.

Тут знаходяться усі налаштування атрибутів, що мають бути проаналізовані під час валідації записів продуктів. Щоб побачити всю інформацію для звіту за замовчуванням, етап налаштувань можна пропустити. Самі в цій секції користувач може задати ті параметри згідно з якими алгоритм описаний у пункті 2.1 буде формувати дані для звіту.

Секція “Columns to show in the report” потрібна для того, що обрати колонки (атрибути), що будуть у ньому присутні. Виходячи з тих даних, що користувач обере будуть створені класи для валідації, діаграма яких зображена у додатку ІАЛЦ.045490.006 Д2. За замовчуванням колонки всі будуть відображені.

Секція “Products to display” відповідає за значення ідентифікатора типу записів продуктів, які користувач хоче побачити у звіті. Можна обрати усі продукти, тільки записи з помилками або безпомилкові записи. Від цього залежатиме чи будуть додані ті чи інші дані до файлу.

Секція “Product custom properties to check in ‘Copy’ column” дозволяє обрати кастомні атрибути сутності продукту зі списку для того, щоб перевірити наявність значення кожного з них та відобразити результат у колонці під назвою “Copy”. Від цих параметрів залежатимуть поля валідаційного класу ProductCustomAttributes.

Далі розглянемо вкладку для налаштування записів до перевірки. На рисунку 5 зображено її інтерфейс.

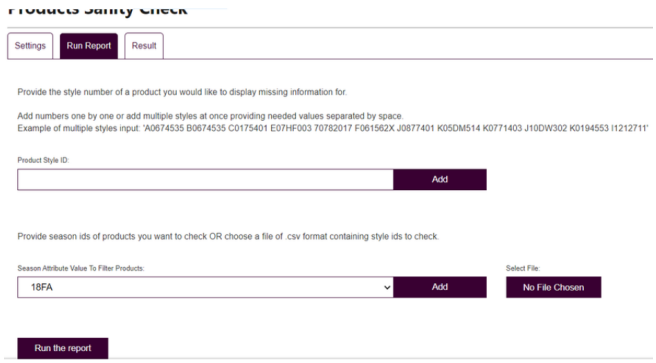


Рисунок 5. Інтерфейс вкладки задання записів для звіту системного модуля для аналізу сутностей продуктів бд SFCC.

На цій вкладці розміщено поля, що надають можливість введення параметрів записів BP, які користувач має на меті перевірити. У поле “Product Style ID” користувач може ввести довільну кількість ідентифікаторів BP, що мають стати основою для звіту.

Поле “Season Attribute Value To Filter Products” потрібне для того, щоб з випадаючого списку обрати ідентифікатори атрибуту сезону за яким буде відфільтровано ті записи, що мають бути у звіті. Кількість ідентифікаторів не обмежена. Цей атрибут задається для запису при його створенні з метою класифікувати продукт як приналежний до до певного сезону або події. Наприклад, значення параметру сезону створеного під час події “День Матері 2023” може бути закодовано ідентифікатором 23MD (2023 mother’s day). Це стане у нагоді дата менеджерам при додаванні нових записів у базу для того щоб одразу перевірити валідність створених продуктів в рамках заданого сезону.

У вкладці для визначення записів також є поле для вибору файлу формату .csv. Воно було створене для зручності, щоб користувач міг сформувати список ідентифікаторів не тільки у полі вводу, а у вигляді таблиці в іншому додатку та імпортувати його в системний модуль для валідації. Вірно сформований файл має вигляд зображений на рисунку 6. Тобто, кожен запис знаходиться у першій колонці файлу, один запис на один рядок.

	A
1	A0674535
2	B0674535
3	C0175401
4	E07HF003
5	70782017
6	F061562X
7	J0877401
8	K05DM514
9	K0771403
10	J10DW302
11	K0194553
12	I1212711

Рисунок 6. Вигляд вірно сформованого файлу формату .csv у додатку LibreOffice Calc для імпорту в системний модуль для валідації.

На разі, системний модуль не надає можливості перевіряти записи згідно усіх трьох полів. З них має бути заповнене одне, якщо заповнено декілька, програма проігнорує деякі з них. У полів задано такі пріоритети:

1. Файл, що користувач імпортує в додаток має найвищий пріоритет. Якщо задано це поле, всі інші будуть проігноровані.
2. Поле вводу ідентифікаторів має другий пріоритет. Якщо задано це поле, то поле сезону буде проігноровано.
3. Поле вводу сезону має найнижчий пріоритет. Його буде взято за основу для створення звіту тільки в тому випадку, якщо всі інші поля пусті.

Унизу вкладки налаштування записів знаходиться кнопка “Run the report”. При натисненні на неї відбувається запуск алгоритму валідації згідно із створеними параметрами (описаного в пункті 3.1). Тобто, робиться запит на ендпоінт BMProduct-ExportReport. Як згадувалось під час опису роботи цього ендпоінта, він повертає ідентифікатор CO, який зберігає інформацію, що ввів користувач та статус її обробки. Розглянемо вкладку з результатом валідації та те як саме відбувається створення звіту в двох виглядах: табличному та у вигляді файлу. Якщо користувач натиснув кнопку для створення звіту на цій вкладці він бачитиме позначку завантаження, цей сценарій зображено на рисунку 7.

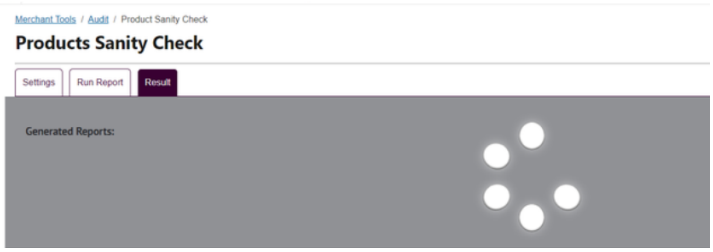


Рисунок 7. Вигляд вкладки результату після натиснення кнопки “Run the report”.

Позначка завантаження говорить про те, що звіт ще не готовий до того, щоб бути відображеним на поточній сторінці у табличному вигляді. Вона з’являється тоді, коли скрипт клієнтської частини робить запит на ендпоінт BMProduct-CheckIfReadyAndRender, мета якого як раз і є відтворення даних звіту на сторінці результату у вигляді таблиці. Йому скрипт передає параметр — ідентифікатор того кастомного об’єкта, що був створений під час запиту на BMProduct-ExportReport. Розглянемо алгоритм роботи цього ендпоінта:

1. Взяття кастомного об’єкту з бази за ключем, що передав параметром клієнтський скрипт.
2. Створення об’єкту результату. Початково він ініціалізується з полями ready: false, page: ‘’.
3. Перевірка статусу CO. Якщо він дорівнює “Completed”, тобто звіт готовий для парсингу і відображення, алгоритм переходить до наступного кроку. Якщо ні, то алгоритм переходить до завершення повертаючи JSON рядок об’єкту результату.
4. Встановлення поля ready об’єкту результату в значення true.
5. Подальші кроки обернуті в блок відловлення помилки. На цьому етапі може виникнути помилка перевищення допустимої квоти на розмір розмітки сторінки, що рендериться у ендпоінті (3 MB), а також квоти на розмір колекції з даними звіту (1200 рядків).
6. Виклик методу controller.getReportFromFile для взяття результату звіту. У цьому методі відбувається парсинг допоміжного файлу, що був

сформований на етапі запису даних звіту (під час виконання методу controller.writeAndExportReport). Усі розпаршені рядки записуються у масив з даними. Після обробки допоміжний файл видаляється. Якщо під час запису у масив виникло перевищення квоти на розмір колекції, повертається повідомлення про це.

7. Далі йде спроба створення розмітки сторінки за допомогою даних згенерованого звіту. Якщо на попередньому кроці виникла помилка, то дані звіту пусті і повертається розмітка, що містить текст помилки. Якщо помилки перевищення квоти на розмір масиву не виникло, але під час створення сторінки виникає помилка перевищення квоти на розмір файлу розмітки, то алгоритм переходить у блок catch і генерує розмітку з текстом помилки (зображена на рисунку 8). Якщо помилок не виникло на жодному з кроків, то повертається розмітка сторінки, що містить таблицю з даними звіту (зображена на рисунку 9). Розмітка записується у поле page об’єкту результату.

8. Останнім кроком ендпоінт повертає клієнтові JSON рядок з об’єктом результату.

Рисунок 8. Вигляд розмітки з помилкою відображення звіту в табличному вигляді.

Рисунок 9. Вигляд успішно згенерованої таблиці результату.

Цілком можливий сценарій при якому запит від клієнта може зависнути на 3 кроці алгоритму, оскільки звіт може генеруватись досить довго (залежно від кількості записів заданих на перевірку). Він опрацьовується наступним чином: від клієнтського скрипту запити надходять на ендпоінт BMProduct-CheckIfReadyAndRender з заданим інтервалом у 5 секунд. Встановлено таймер на 30 секунд і якщо за цей час бек енд не поверне JSON розмітку об’єкта результату поле ready якого дорівнює значенню true, то запити припиняються і користувачу висвітлюється повідомлення про те, що звіт надто великий щоб бути відображеним у табличному вигляді (зображене на рисунку 8).

Даний розділ є основним для розуміння структури написаної програми, підходів, що були застосовані при розробці та взаємодії її частин одна з одною. В наступному розділі наведено інформацію про тестування системного модуля, можливості, що він надає, а також аналіз його переваг та пунктів для покращення.

4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Тестування системного модуля

Для якісного тестування модуля створеного в рамках дипломного проєкту треба було розглянути усі можливі сценарії роботи модуля. Спочатку розглянемо правильність роботи функціоналу задання записів. Модуль надає можливість задання записів для перевірки трьома способами: списком ідентифікаторів BP через поле вводу, списком ідентифікаторів BP заданих у файлі відповідного вигляду і формату (рисунок 6), списком значень кастомного атрибуту season сутності продукту через обирання значень з селектору. Інтерфейс цієї частини додатку зображено на рисунку 5.

Розглянемо аналіз двох записів продуктів, що задані списком ідентифікаторів BP за допомогою поля вводу. На рисунку 10 зображено зразок правильного вигляду такої конфігурації звіту.

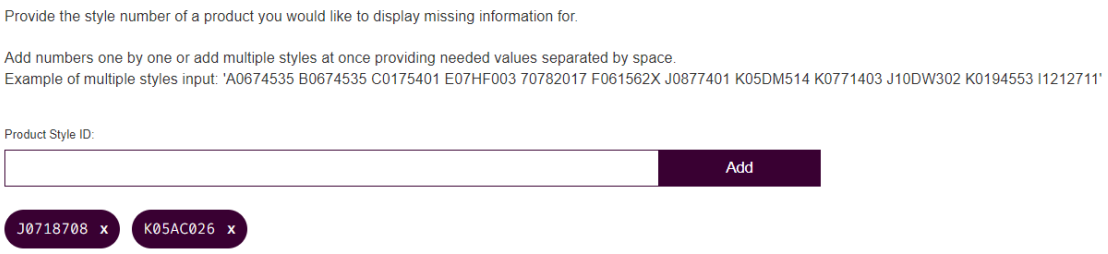
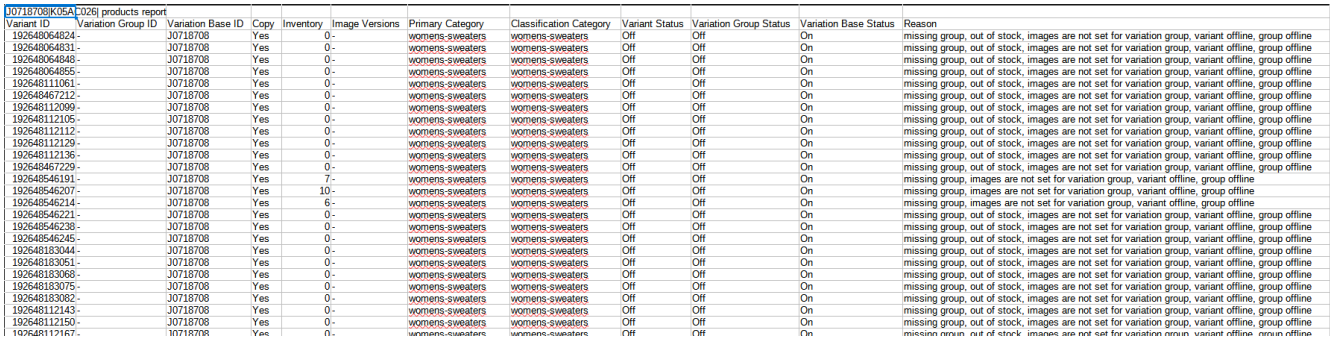


Рисунок 10. Вигляд вірної конфігурації поля вводу списку ідентифікаторів BP для створення звіту

Тут задані два валідних ідентифікатори, пізніше при розгляді обробки помилок розглянемо сценарій, коли користувач ввів неправильний ідентифікатор продукту. Запускаємо генерацію звіту кнопкою “run the report” і переходимо до вкладки результату. Оскільки задано було всього два записи (звіт досить малий, щоб бути відображеним на сторінці), користувачу відображається таблиця із результатом аналізу згідно з усіма параметрами

(колонками для валідації) за замовчуванням для заданих двох записів (схожа на ту, що зображена на рисунку 9). Також водночас із таблицею створюється результуючий файл формату .csv, його вигляд зображено на рисунку 11.



J0718708|K05A|026| products report

Variant ID	Variation Group ID	Variation Base ID	Copy	Inventory	Image Versions	Primary Category	Classification Category	Variant Status	Variation Group Status	Variation Base Status	Reason
192648064824	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648064831	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648064848	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648064855	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648111061	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648467212	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648112099	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648112105	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648112112	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648112129	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648112136	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648467229	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648546191	-	J0718708	Yes	7	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, images are not set for variation group, variant offline, group offline
192648546207	-	J0718708	Yes	10	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, images are not set for variation group, variant offline, group offline
192648546214	-	J0718708	Yes	6	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, images are not set for variation group, variant offline, group offline
192648546221	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648546238	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648546245	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648183044	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648183051	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648183068	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648183075	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648183082	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648112143	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648112150	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline
192648112167	-	J0718708	Yes	0	-	womens-sweaters	womens-sweaters	Off	Off	On	missing group, out of stock, images are not set for variation group, variant offline, group offline

Рисунок 11. Вигляд декількох рядків файлу звіту формату .csv для двох продуктів заданих ідентифікаторами BP у полі вводу.

В табличному вигляді кожна причина помилковості колонки відображається окремо при наведенні на неї курсором миші, а у вигляді файлу всі причини групуються та відображаються в останній колонці звіту під назвою “Result”.

Розглянемо аналіз записів продуктів заданих кастомним атрибутом season. На рисунку 12 зображено зразок правильно заданого значення двох сезонів за якими буде відображено звіт.

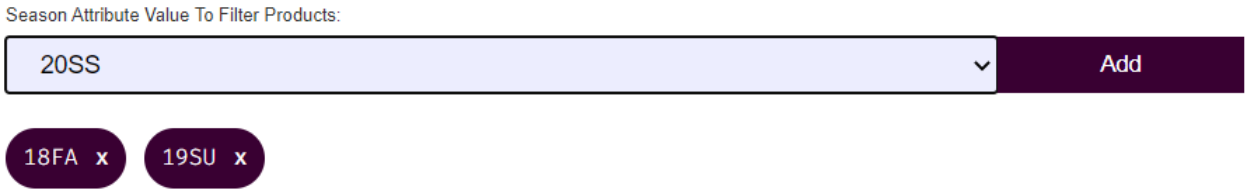


Рисунок 12. Вигляд вірної конфігурації поля вводу двох значень кастомного атрибуту season згідно з якими буде згенеровано звіт.

Якщо записів, що задовольняють умові буде досить мало для того, щоб відобразити звіт на одні сторінці, то користувач побачить на вкладці результату таблицю із звітом по аналізу на кшталт тієї, що зображена на рисунку 9 (тільки з відповідними записами, що мають значення атрибуту сезону 18FA або 19SU), а також зможе завантажити згенерований файл .csv із тим самим звітом. У іншому випадку буде створено тільки файл на кшталт схожий на той, що зображено на рисунку 11. У першому рядку файлу будуть вказані задані користувачем значення атрибуту сезону, замість значень ідентифікатора продукту.

У випадку задання записів для перевірки за допомогою файлу користувач у результаті отримає те саме, що і у випадку задання через поле вводу (файл зі звітом та таблицю, якщо не виникне помилок перевищення квот у ході її генерації). Особливості цього підходу полягають в тому, щоб надати на вхід правильно сформований .csv файл (рисунок 6). Отриманий аналіз базуватиметься на основі тих ідентифікаторів продуктів типу BP, що зазначені у вхідному файлі.

У разі, якщо користувач не задав жодного з параметрів вказання записів для перевірки, він не зможе натиснути “run the report”. Висвітлиться повідомлення про те, що треба вказати принаймні один з них. Для цього сценарію передбачена валідація зі сторони клієнтського скрипту.

Далі розглянемо гнучкість задання налаштувань звіту (вкладка налаштувань зображена на рисунку 4). Як вже зазначалось раніше, звіт може генеруватись у відповідності до 8 параметрів, так званих колонок валідації, таких як: кількість товару на складі, наявність усіх необхідних варіацій картинок продукту, приналежність продукту до первинної категорії, приналежність продукту до класифікаційної категорії, онлайн статуси для кожного з типів записів (PV, BP, VG), наявність значень за замовчуванням та локалізованих варіацій у заданих кастомних атрибутах продукту. Тут слід зазначити, що оскільки за замовчуванням в кожному звіті мають бути

присутні ідентифікатори усіх типів записів для заданого продукту, то перевірка може бути не пройдена тими продуктами, що не мають запису типу VG. Кожну з цих колонок можна відобразити, або приховати у звіті. Якщо всі колонки приховано, то у звіті фігуруватимуть лише обов'язкові колонки, що містять ідентифікатори записів продуктів (в такому разі продукт вважається не валідним, якщо не містить VG). Для прикладу оберемо лише одну колонку валідації — приналежність продукту до первинної категорії (таке налаштування зображене на рисунку 13). При цьому у звіті будуть представлені записи проаналізовані лише за цим параметром.

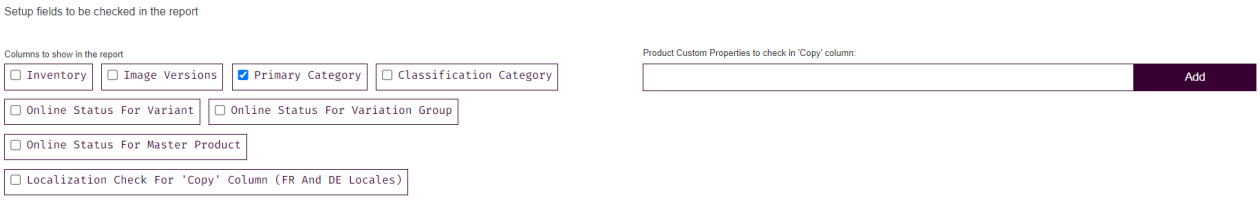


Рисунок 13. Зображення конфігурації згідно з якою продукти будуть провалідовані тільки за параметром приналежності до первинної категорії.

З таблиці звіту зображеної на рисунку 14 можна зробити висновок, що не дивлячись на те, що продукти пройшли перевірку на приналежність до первинної категорії, вони не пройшли валідацію за замовчуванням — наявність VG записів до яких приналежні знайдені PV записи, тобто відсутність класифікації деяких записів продуктів за кольором (що унеможливлює їх правильне відображення на сайті).

Variant ID	Variation Group ID	Variation Base ID	Primary Category
192648064824	-	J0718708	womens-sweaters
192648064831	-	J0718708	womens-sweaters
192648064848	-	J0718708	womens-sweaters
192648064855	-	J0718708	womens-sweaters
192648111061	-	J0718708	womens-sweaters
192648467212	-	J0718708	womens-sweaters
192648112099	-	J0718708	womens-sweaters

Рисунок 14. Табличний вигляд результату валідації запису за параметром приналежності до первинної категорії.

Таким чином можна здійснювати перевірку записів за атрибутом за замовчуванням та за будь-яким з обраних валідаційних атрибутів.

Також у користувача є можливість обирати тип записів, що будуть відображені у звіті. Ними можуть бути лише помилкові записи, усі записи та тільки валідні записи. Для прикладу згенеруємо звіт з усіх записів продукту (до того на усіх представлених зображеннях результатів звіту фігурували типи записів за замовчуванням — тільки помилкові записи). На рисунку 15 зображено звіт за усіма параметрами валідації для усіх записів продукту і очевидно, що вірно сформовані записи зображені поряд з невірно сформованими, тобто у звіті відображаються усі існуючі записи у бд для заданого продукту.

Variant ID	Variation Group ID	Variation Base ID	Copy	Inventory	Image Versions	Primary Category	Classification Category	Variant Status	Variation Group Status	Variation Base Status
883360391847	A0674535_001	A0674535	Yes	316	L0, B0, D0, F0, 0, SWATCH	mens-shirts	mens-shirts	On	On	On
883360391854	A0674535_001	A0674535	Yes	195	L0, B0, D0, F0, 0, SWATCH	mens-shirts	mens-shirts	On	On	On
888718068549	A0674535_001	A0674535	Yes	0	L0, B0, D0, F0, 0, SWATCH	mens-shirts	mens-shirts	Off	On	On
192648905042	-	A0674535	Yes	0	-	mens-shirts	mens-shirts	Off	Off	On

Рисунок 15. Табличний вигляд звіту згенерованого із налаштуванням відображення усіх записів заданого продукту.

Колонка валідації кастомних атрибутів продукту відрізняється від усіх інших, оскільки реалізовано можливість задавати ті атрибути, що будуть

перевірені в рамках неї. Проте, при поточній реалізації є змога лише перевірити наявність існування значення в заданих атрибутах (тобто що вони не є пустими). Існує перевірка на наявність значення за замовчуванням та значення за передвизначеними локалізаціями: німецька та французька (для тих сайтів, де ці локалізації визначені). Для прикладу додамо до поля вводу декілька кастомних атрибутів та зробимо їх додаткову перевірку на локалізацію (відповідне налаштування зображене на рисунку 16).

Рисунок 16. Налаштування звіту для відображення колонки “Copy” із додатковою перевіркою на наявність локалізованого значення.

На рисунку 17 зображено результат валідації із вищезгаданими параметрами. Звернемо увагу на те, що в результуючій таблиці валідація для записів не пройшла тому, що додані кастомні атрибути не мають для записів ані заданого значення за замовчуванням, ані локалізованого значення.

Variant ID	Variation Group ID	Variation Base ID	Copy
192648157304	J0794505_001	J0794505	No
192648157311	J0794505_001	J0794505	No
192648157328	J0794505_001	J0794505	No
192648157335	J0794505_001	J0794505	No
192648157342	J0794505_001	J0794505	missing BootType, missing BootType in fr_FR locale, missing FitShape in fr_FR locale, missing fabric in fr_FR locale, missing BootType in de_DE locale, missing FitShape in de_DE locale, missing fabric in de_DE locale
192648157359	J0794505_001	J0794505	
192648931171	-	J0794505	
192648931188	-	J0794505	
192648931195	-	J0794505	
192648931201	-	J0794505	

Рисунок 17. Табличний вигляд результату валідації продуктів з розширенням параметру “Copy” та перевіркою заданих атрибутів на наявність локалізованого значення.

Розглянемо обробку помилок під час створення звіту. Якщо користувач введе неправильний ідентифікатор BP у поле вводу, або за допомогою файлу, він побачить повідомлення зображене на рисунку 18 в табличному вигляді, а у файлі звіту він побачить вміст зображений на рисунку 19. Таке повідомлення не потрібне для атрибуту сезону, оскільки в цьому випадку користувач не може ввести значення вручну, а лише обрати запропоновані з випадного списку.

Variant ID	Variation Group ID	Variation Base ID	Copy	Inventory	Image Versions	Primary Category	Classification Category	Variant Status	Variation Group Status	Variation Base Status
There is no results found for requested Style Number Style Number J079450t is invalid										

Рисунок 18. Табличний вигляд повідомлення про невірний ідентифікатор BP.

J079450t	products report											
Variant ID	Variation Group ID	Variation Base ID	Copy	Inventory	Image Versions	Primary Category	Classification Category	Variant Status	Variation Group Status	Variation Base Status	Reason
Style Number J079450t is invalid											

Рисунок 19. Вміст файлу звіту за умови невірного задання ідентифікатора BP у вхідних даних.

У випадку, якщо звіт занадто великий, щоб бути відображеним на сторінці додатку в табличному вигляді, тобто перевищено одну з квот (на розмір масиву результату або на розмір розмітки таблиці), або якщо звіт не було сформовано протягом 30 секунд, користувач побачить повідомлення про це, зображене на рисунку 8. Файл зі звітом буде згенеровано в очікуваному вигляді.

При поточній реалізації не передбачено обробки інших помилок, тому при їх виникненні користувач бачитиме повідомлення зображене на рисунку 5 навіть при виникненні іншої помилки. Для уточнення даних неочікуваної помилки розробник має перевірити вміст log файлу середовища, де генерувався звіт.

Також, в роботі реалізовано функціонал для висвітлення репрезентативної інформації по усіх продуктах сайту у вигляді діаграм.

Сторінка статистики зображена на рисунку 3. Тут у користувача немає можливості напряму кастомізувати дані. Налаштування для статистики є налаштуваннями за замовчуванням для створення звіту: відображаються усі валідаційні колонки та задані заздалегідь кастомні атрибути продукту за замовчуванням (для колонки “сору”). Для статистики проводиться аналіз усіх записів продуктів (не тільки не валідних, як за замовченням).

4.2. Аналіз отриманих даних

Користувацький інтерфейс картриджу надає широкі можливості кастомізації звіту. До того ж, завжди за замовчуванням на виході користувач отримує файл з результатом валідації та при можливості таблицю з результатом на сторінці додатку. Під час тестування програми, було вручну перевірено правильність валідації за допомогою процесу налагодження програми, а також за допомогою перегляду відповідних записів продуктів у Business Manager. Функціонал картриджу передбачає відловлення поширених помилок, щоб не шокувати користувача неочікуваною поведінкою програми. Додаток створено таким чином, щоб не змушувати довго чекати на результат. Максимальний період очікування відгуку після початку роботи — 30 секунд. За 30 секунд звіт або буде відображено на сторінці у вигляді таблиці, або буде висвітлено повідомлення про те, що чекати не варто і звіт може бути створений лише у вигляді файлу. Після завершення генерації файлу користувач отримує повідомлення про це на електронну пошту з якої він має доступ до BM (що прив’язана до облікового запису SFCC).

З точки зору покращень для картриджу можна виділити декілька пунктів. По-перше, обробка помилок могла б бути більш конкретною. На разі відловлюються лише помилки пов’язані із відображенням таблиці та невірністю введених ідентифікаторів BP. Користувач може бути введений в

оману при виникненні помилки іншого роду і чекатиме на звіт, що ніколи не буде згенерований.

По-друге, при створенні картриджу було зроблено акцент на певні конкретні запити від умовного клієнта. Саме тому він не є корисним, але не досить широким для вжитку і в будь-якому разі при підключенні до платформи SFCC його доведеться «підігнати» під конкретні потреби сайту. Наприклад, для валідації картинок було взято за умову, що кожен валідний продукт має принаймні шість картинок із заданими ідентифікаторами певного формату. Ця умова може виконуватись не для кожного сайту. Але, варто зазначити, що структура розробки навмисне створена таким чином, що дозволяє впроваджувати нові валідаційні правила без особливих зусиль зі сторони програміста.

По-третє, в коді програми використовується застарілий виклик асинхронного скрипта new Pipelet('RunJobNow'). Відомо два способи запуску Job з програмного коду: вищезгаданий — застарілий, та за допомогою OCAPI. В даному випадку було використано саме застарілий спосіб для спрощення інтеграції даного модуля у платформу. Оскільки для використання OCAPI належне виконувати додаткові дії для створення клієнта з відповідними правами, виконати додатковий запит авторизації. Це може здатись досить складною задачею для проєктів, що вже не працюють з OCAPI. До того ж використання застарілого рішення для виконання однієї цієї дії здається допустимим з урахуванням того, що загалом модуль не використовує OCAPI (чому було обрано підхід створення картриджу, а не застосування RESTful API описано в розділі 1.2).

І останнє, що варто зазначити з точки зору покращень — це можливість відійти від тієї концепції, що у звіті є обов’язкові параметри для валідації. Справа у тому, що на разі при потребі перевірити продукти за одним параметром у користувача немає можливості відключити аналіз самих записів продуктів. Наприклад, як зображено на рисунку 14, при потребі валідувати

приналежність до певної категорії, користувач навіть при її наявності бачить певні продукти записи VG яких відсутні. Тут ми стикаємось із проблемою реалізації. Так було зроблено тому, що основна ідея усього аналізу початково була у знаходження продуктів, що неправильно відображаються або зовсім не відображаються на сайті. Тобто, ідея кастомізації звіту була інтегрована пізніше і є другорядною в даному випадку. Проте, дійсно в якості покращень модуля це важливий пункт, оскільки не всі опціональні колонки валідації залежать від наявності запису VG у базі. Для певних з них перевірку цього атрибуту варто було б опустити.

Проаналізуємо також дані статистики, зображені на рисунках 20 та 21.

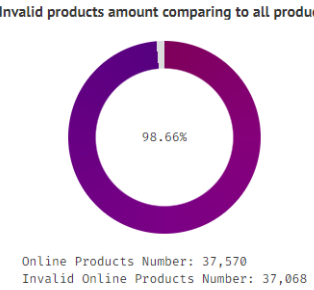


Рисунок 20. Діаграма, що відображає співвідношення числа невірно сформованих продуктів сайту до усіх продуктів.

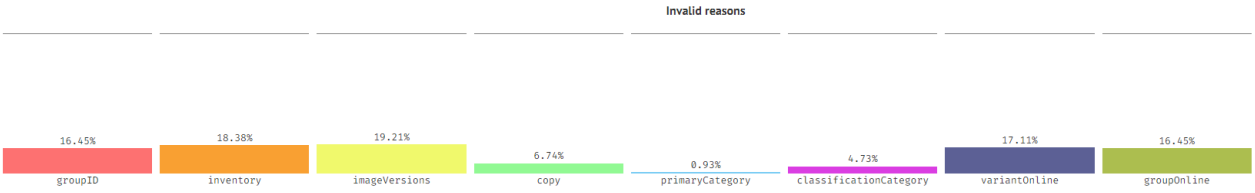


Рисунок 21. Діаграма, що відображає відсоток частоти появи того або іншого атрибуту в звіті про хибність продукту.

З діаграми зображеної на рисунку 20 робимо висновок, що понад 98 відсотків продуктів сайту не відповідають вимогам валідації сформованої під час збору статистики. Таке неочікуване співвідношення пояснюється тим, що статистика була забрана в середовищі Sandbox, що необхідне лише для

розробки сайту і не співпадає із тим, що бачить кінцевий користувач. Більш того, для цього середовища достатньо невеликої кількості валідних продуктів, щоб розробник міг працювати без проблем із сайтом. Тут відключене кешування і чим більше їх буде, тим довше можуть завантажуватись певні сторінки, до того ж довше проходитиме реіндексація бд при потребі, що сповільнить процес розробки.

З діаграми на рисунку 21 бачимо, що найчастішою проблемою в продуктах є відсутність певної варіації картинки. На другому місці те, що продукти недоступні до придбання через відсутність на складі. Параметри валідації groupID та groupOnline виявились однаково часто зустрічними у звіті. Судячи з усього так склалось тому, що на сайті параметр статусу продукту виставлявся на рівні PV (17.11% частоти появи в звіті валідації), а не на рівні VG, завдяки чому VG тут можуть бути оффлайн тільки в тому випадку коли їх не існує (немає groupID). Найрідше зустрічаються продукти, що не призначені жодній первинній категорії. Тому що цей параметр впливає на дуже багато аспектів відображення елементів сторінки продукту на сайті і якщо продукт існує, то згідно із рекомендаціями до розробки під платформу SFCC, йому має бути призначено цей тип категорії.

ВИСНОВОК

Системний модуль розроблений в рамках дипломного проєкту створений таким чином, щоб задовольняти усім вимогам поставленим у пунктах 1.1 та 1.2. Формування звіту відбувається повністю асинхронно, додаток має зручний візуальний інтерфейс, надає користувачам широкі можливості кастомізації процесу аналізу продуктів та додатково робить статистику по результатах валідації усіх сутностей продуктів бази даних, що існують на сайті.

У ході аналізу результатів тестування було виділено деякі недоліки створеного модуля та наведено ідеї для його покращення. Він має недосконалу обробку помилок, є досить вузькоспрямованим з точки зору параметрів валідації, використовує застарілий модуль Pipelet та його основний алгоритм має деякі обов'язкові атрибути для перевірки, які не можуть бути опущені.

Проте, не зважаючи на зазначені недоліки, він вірно виконує основні свої функції і може стати у нагоді для підприємств, що займаються електронною торгівлею з використанням платформи SFCC. Можна сказати, що використовуючи створений у проєкті картридж, цільова аудиторія буде допускатись меншої кількості помилок при створені нових продуктів у базі даних та зможе позбутись старих. Його гнучкість дозволяє генерувати інформацію із заданням потрібних атрибутів та задавати записи для перевірки декількома способами, що може стати у нагоді в різних ситуаціях.

Він є досить простим для розуміння досвідченими програмістами платформи SFCC. Завдяки особливостям структури він може бути легко розширений новим функціоналом.

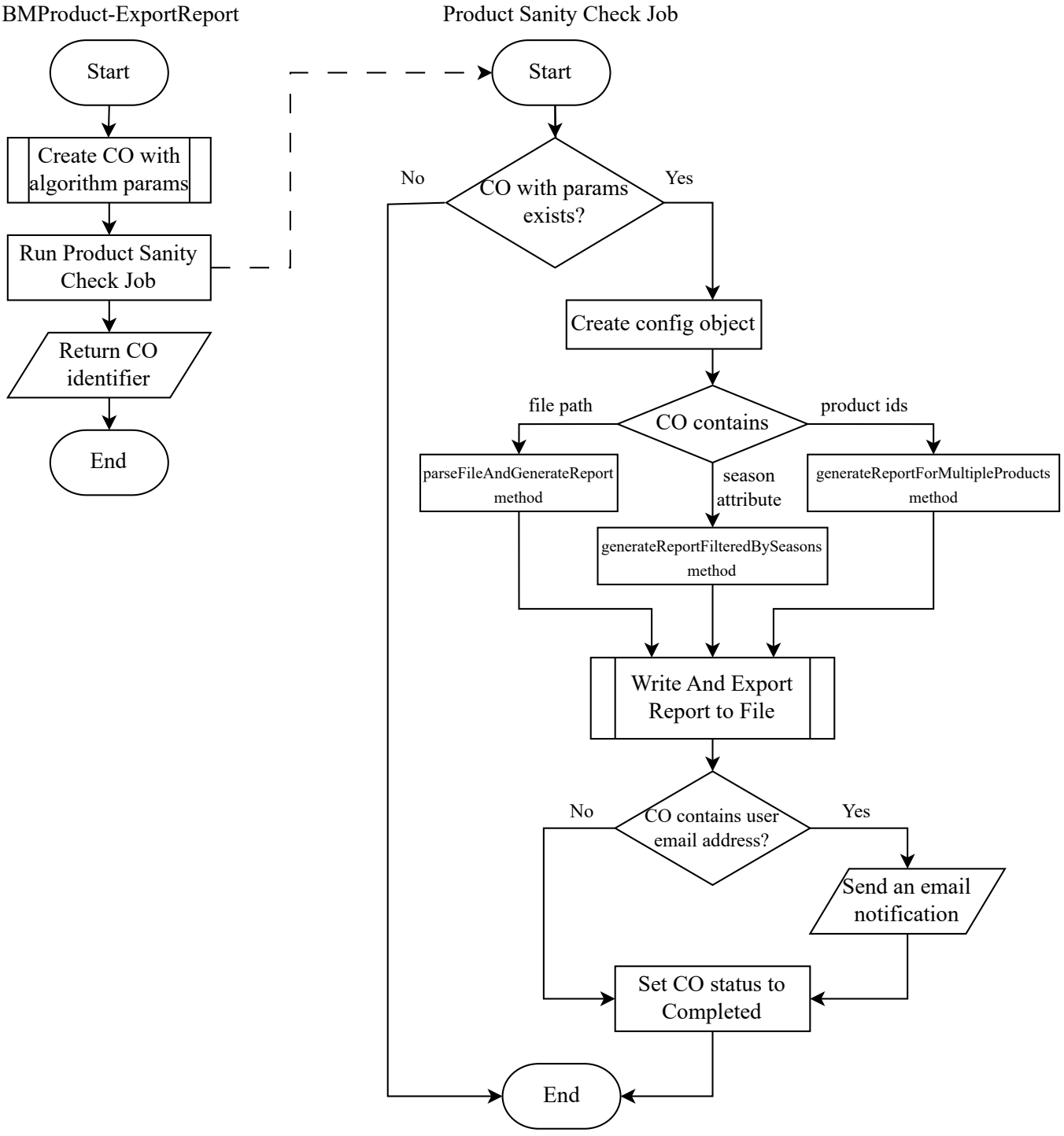
Той факт, що він створений у вигляді розширення до адміністивної утиліти Business Manager надає додаткові можливості, що полягають у легкості виправлення невірних записів. В табличному вигляді звіт містить посилання на записи продуктів у BM, де можна одразу їх змінити.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

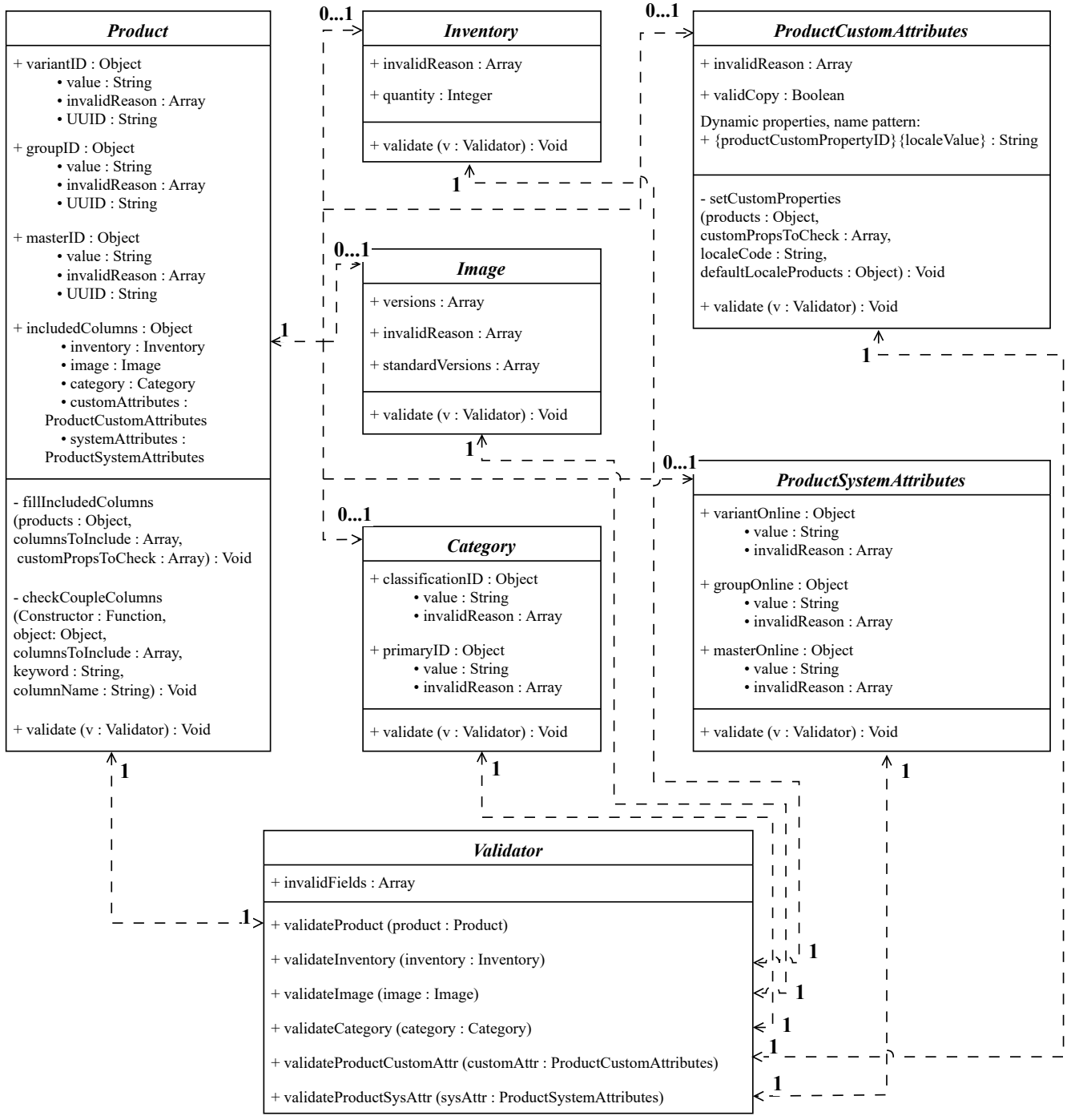
1. Salesforce official site. CRM 101: What is CRM? URL: https://www.salesforce.com/crm/what-is-crm/ (дата звернення 22.05.2023)
2. Salesforce official site. Frequently Asked Questions. URL: https://www.salesforce.com/products/commerce-cloud/faq/ (дата звернення 25.05.2023)
3. Trailhead Learning Module: Get Started with Commerce Cloud Business Manager. URL: https://trailhead.salesforce.com/content/learn/modules/cc-digital-for-developers/cc-business-manager (дата звернення 25.05.2023)
4. Документація Salesforce B2C Commerce 23.5. URL: https://documentation.b2c.commercecloud.salesforce.com/DOC1/index.jsp (дата звернення: 15.05.2023).
5. Refactoring guru. Design Patterns/Behavioral Patterns: Visitor. URL: https://refactoring.guru/design-patterns/visitor (дата звернення 28.05.2023).
6. Trailhead Learning Module: Salesforce B2C Commerce for Developers. URL: https://trailhead.salesforce.com/content/learn/modules/cc-digital-for-developers (дата звернення: 01.04.2023).
7. Trailhead Learning Module: Architecture of Salesforce B2C Commerce. URL: https://trailhead.salesforce.com/content/learn/modules/architecture-of-commerce-cloud-digital (дата звернення: 17.04.2023).
8. Salesforce Commerce Cloud Business Manager Data Maintenance Extension. URL: https://github.com/SalesforceCommerceCloud/sfcc-data-maintenance#readme (дата звернення 21.05.2023).
9. Order Guard Pipeline. URL: https://github.com/SalesforceCommerceCloud/bc_orderwatcher/blob/master/documents/Description.md (дата звернення 21.05.2023).

10. Medium. Article: How to Understand Salesforce Commerce Cloud Platform. Author: Oleg Sapishchuk. URL: https://osapishchuk.medium.com/how-to-understand-salesforce-commerce-cloud-78d71f1016de (дата звернення 06.01.2023).

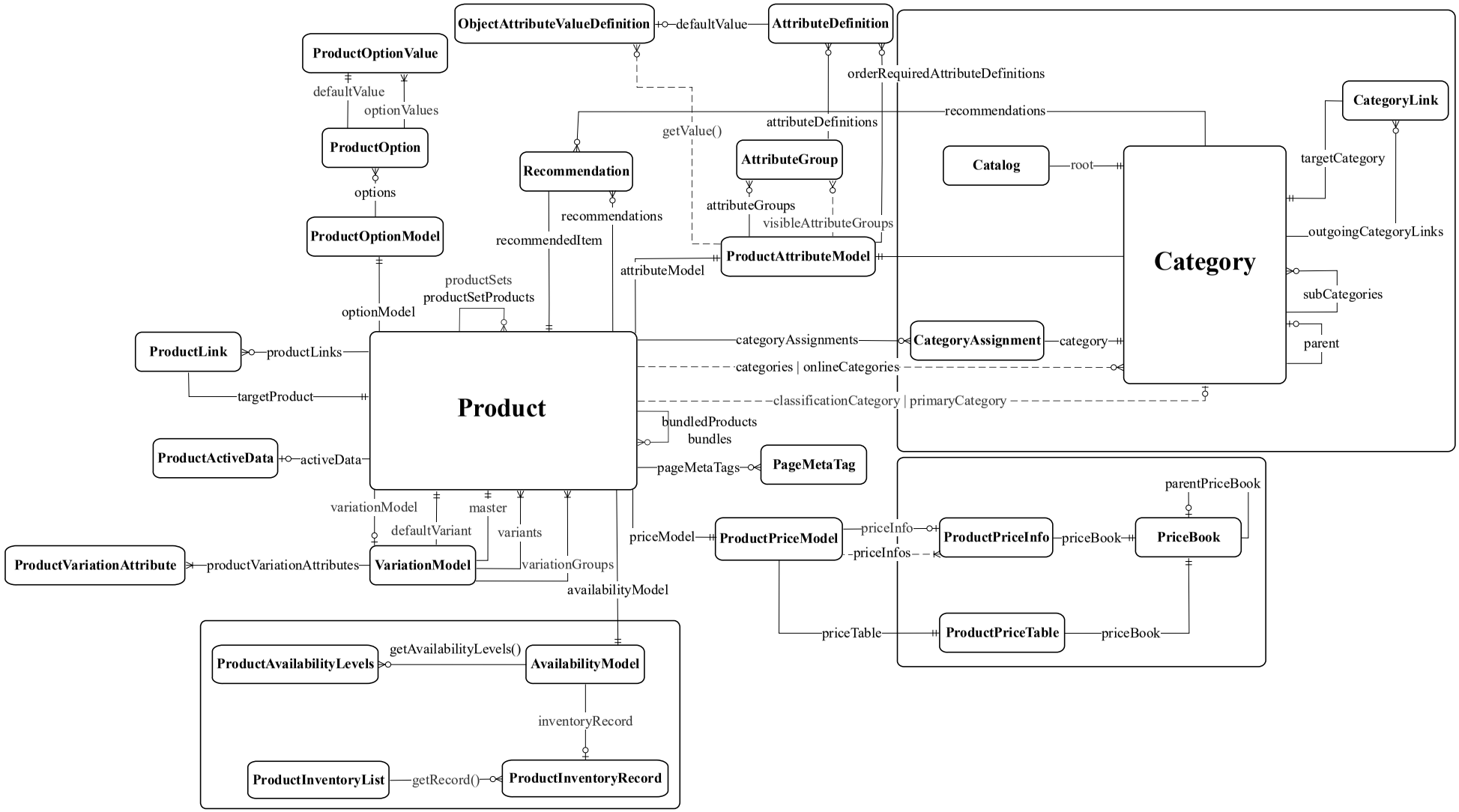
11. Salesforce B2C Commerce Certified Developer Questions. Exploring the Cartridge folder. URL: https://salesforceb2c-certificationquestions.com/sfra-course/sfcc-cartridge-path/ (дата звернення 06.01.2023)



					ІАЛЦ.045490.005 Д1	
Змін.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		Корж А. А.			Системний модуль аналізу сутностей бази даних продуктів у Salesforce Commerce Cloud ***Алгоритм аналізу записів бази даних продуктів SFCC***	*Літ.* / *Аркуш* / *Аркушів*: 1 / 1
Перевірив		Петрашенко А.В.				КПІ ім. Ігоря Сікорського, ФПМ гр. КВ-92
Н. контроль		Клятченко Я.М.				
Зав. каф.		Романкевич В.О.				



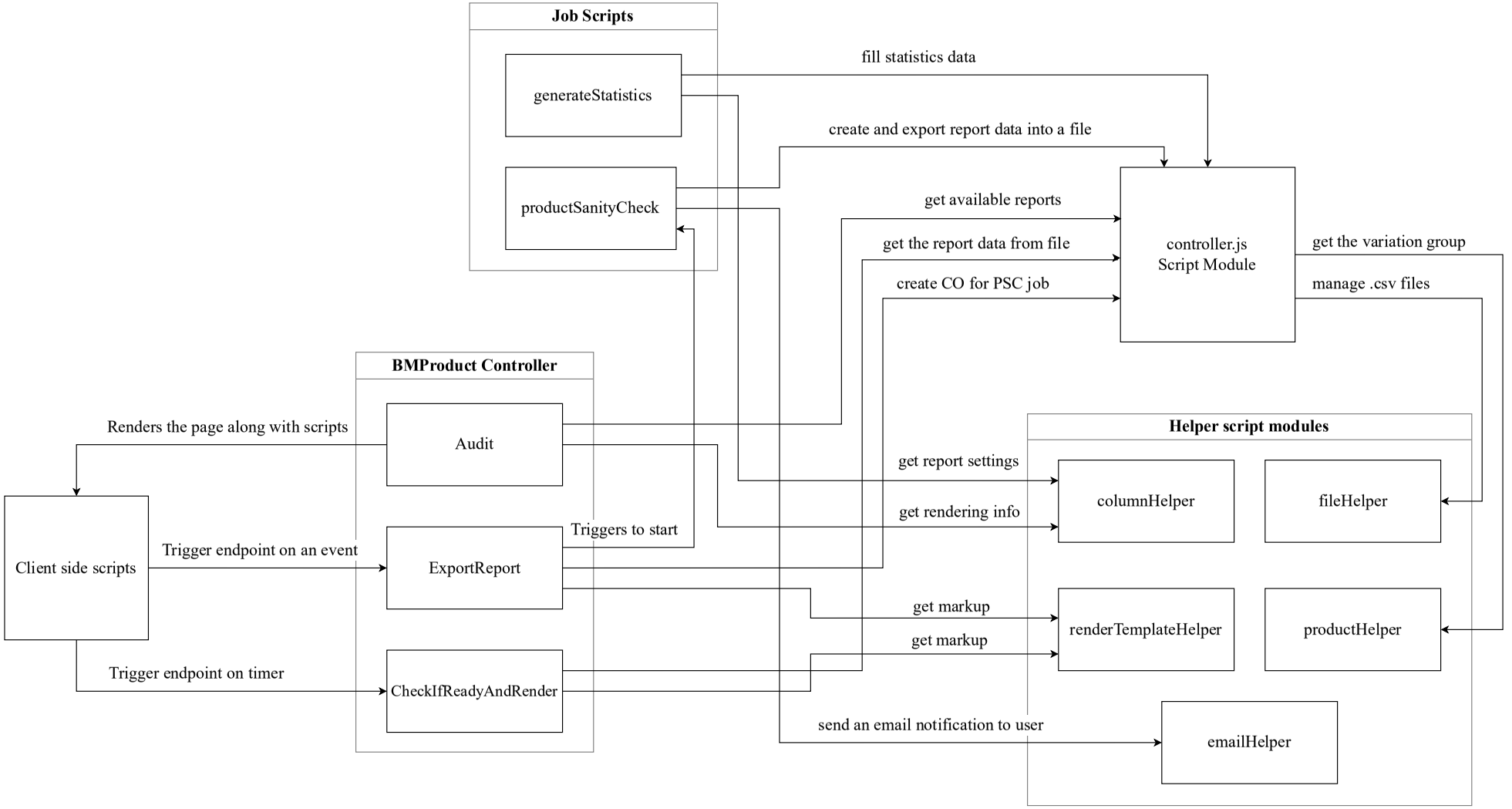
					ІАЛЦ.045490.006 Д2	
Змін.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		Корж А. А.			Системний модуль аналізу сутностей бази даних продуктів у Salesforce Commerce Cloud ***Схема допоміжних класів для обробки та валідації даних***	*Літ.* | *Аркуш* | *Аркушів* 1 | 1
Перевірив		Петрашенко А.В.				
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ гр. КВ-92
Зав. каф.		Романкевич В.О.				



					ІАЛЦ.045490.007 ДЗ
Змін.	*Арк.*	*№ докум.*	*Підпис*	*Дата*	
Розробив		Корж А. А.			Системний модуль аналізу сутностей бази даних продуктів у Salesforce Commerce Cloud ***ERD-діаграма зв'язків сутності продукту бази даних SFCC***
Перевірив		Петрашенко А.В.			
Н. контроль		Клятченко Я.М.			
Зав. каф.		Романкевич В.О.			

Літ.	*Аркуш*	*Аркушів*
	1	1

КПІ
ім. Ігоря Сікорського,
ФПМ гр. КВ-92



					ІАЛЦ.045490.008 Д4
Змін.	*Арк.*	*№ докум.*	*Підпис*	*Дата*	
Розробив		Корж А. А.			Системний модуль аналізу сутностей бази даних продуктів у Salesforce Commerce Cloud ***Схема взаємодії функціональних модулів створеного картриджу***
Перевірив		Петрашенко А.В.			
Н. контроль		Клятченко Я.М.			
Зав. каф.		Романкевич В.О.			

Літ.	*Аркуш*	*Аркушів*
	1	1

КПІ
ім. Ігоря Сікорського,
ФПМ гр. КВ-92