Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка програмного модуля для підбору колірних поєднань на основі заданого відтінку

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ-21-2

_____/Саватєєв В. О. /

Керівник кваліфікаційної роботи _____ / Шаповалова Н.Н./

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2025

Криворізький національний університет
Факультет: Інформаційних технологій
Кафедра: Моделювання та програмного забезпечення Ступінь вищої освіти: бакалавр
Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____ А. М. Стрюк
«__» _____2025р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-21-2 Саватєєву Володимиру Олександровичу

1. Тема: Розробка програмного модуля для підбору колірних поєднань на основі заданого відтінку затверджено наказом по КНУ № 200с від «10» квітня 2025 р.
2. Термін подання студентом закінченої роботи: «15» травня 2025р.
3. Вихідні дані по роботі: розроблюваний плагін повинен генерувати колірні палітри на основі введеного кольору.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): виконати аналіз теорії кольору та сучасних інструментів для підбору колірних палітр, вибрати та обґрунтувати алгоритми генерації гармонійних палітр, спроектувати програмний модуль для Figma з інтуїтивним інтерфейсом та інтеграцією, здійснити програмну реалізацію плагіна, провести тестування плагіна на стабільність роботи та зручність використання.
5. Перелік ілюстративного матеріалу: схема архітектури плагіна (клієнт-серверна модель), блок-схеми алгоритмів генерації колірних палітр, знімки екранних форм інтерфейсу плагіна, знімки результатів тестування генерації палітр.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури з теорії кольору та сучасних інструментів підбору палітр	27.02.2025 – 29.02.2025
2	Аналіз і порівняння методів генерації колірних палітр та стандартів доступності	30.02.2025 – 09.03.2025
3	Підготовка матеріалів першого розділу роботи	10.03.2025– 18.03.2025
4	Розроблення алгоритмів для автоматичної генерації гармонійних колірних палітр	19.03.2025 – 23.03.2025
5	Проектування архітектури плагіна для інтеграції з Figma	24.03.2025– 30.03.2025
6	Підготовка матеріалів другого розділу роботи	1.04.2025– 04.04.2025
7	Програмна реалізація плагіна з використанням JavaScript, TypeScript, HTML і CSS	05.04.2025– 13.05.2025
8	Тестування плагіна на швидкість, стабільність і зручність інтерфейсу	14.05.20235 – 20.05.202
9	Підготовка матеріалів третього розділу роботи	21.05.2025 – 26.05.2025
10	Оформлення пояснювальної записки	27.05.2025– 05.06.2025

Дата видачі завдання: «15» лютого 2025 р.

Студент _____ / Саватєєв В. О. /

Керівник роботи _____ / Шаповалова Н.Н. /

РЕФЕРАТ

КОЛІР, ПЛАГІН FIGMA, АДАПТИВНА КОНТРАСТНА ГАРМОНІЯ, КОЛІРНЕ КОЛЕСО, ЛОКАЛЬНЕ СХОВИЩЕ, АЛГОРИТМИ ГЕНЕРАЦІЇ ПАЛІТР.

Пояснювальна записка: 72 с., 1 табл., 6 рис., 20 джерел.

Метою кваліфікаційної роботи є розробка програмного модуля у вигляді плагіна для Figma, призначеного для автоматичного підбору гармонійних колірних палітр із можливістю їх інтеграції в робоче середовище дизайнерів.

У роботі проведено аналіз теорії кольору (праці Іттена, Альберса, Вонга) та сучасних інструментів (Adobe Color, Coolors, Paletton, Color Hunt). Виконано порівняння їх за критеріями зручності, функціональності та інтеграції з дизайнерськими платформами. Для розв’язання задачі обрано створення плагіна для Figma з використанням JavaScript, TypeScript, HTML, CSS та Figma Plugin API. Розроблено сім алгоритмів генерації палітр, зокрема адаптивну контрастну гармонію, що забезпечує контрастність ≥ 3:1 за стандартами WCAG. Сформовано структуру даних для збереження палітр у локальному сховищі figma.clientStorage. Реалізовано інтуїтивний інтерфейс та проведено тестування плагіна, яке підтвердило його стабільність і ефективність.

Розроблений плагін дозволяє дизайнерам швидко генерувати, застосовувати та зберігати колірні палітри, оптимізуючи робочий процес і забезпечуючи відповідність стандартам доступності.

Основні положення роботи представлено у вигляді доповіді на внутрішній конференції кафедри програмного забезпечення.

ABSTRACT

COLOR, FIGMA PLUGIN, ADAPTIVE CONTRAST HARMONY, COLOR WHEEL, LOCAL STORAGE, PALETTE GENERATION ALGORITHMS.

Thesis in 72 p., 1 tab., 6 fig., 20 references.

The goal of the thesis is to develop a software module in the form of a Figma plugin for the automatic generation of harmonious color palettes with integration into the designers' workflow.

The thesis analyzes color theory (works by Itten, Albers, Wong) and existing tools (Adobe Color, Coolors, Paletton, Color Hunt). These tools are compared based on usability, functionality, and integration with design platforms. To address the task, a Figma plugin was developed using JavaScript, TypeScript, HTML, CSS, and Figma Plugin API. Seven palette generation algorithms were implemented, including adaptive contrast harmony, ensuring a contrast ratio ≥ 3:1 per WCAG standards. A data structure for storing palettes in figma.clientStorage was designed. An intuitive interface was created, and testing confirmed the plugin's stability and efficiency.

The developed plugin enables designers to quickly generate, apply, and save color palettes, streamlining the design process and ensuring compliance with accessibility standards.

Key findings of the thesis were presented as a report at the internal conference of the Software Engineering Department.

ЗМІСТ

ВСТУП 7
1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ 9
1.1 Критичний аналіз літературних джерел за темою 9
1.2 Актуальність теми кваліфікаційної роботи 14
1.3 Цілі та завдання кваліфікаційної роботи 15
2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ 20
2.1 Опис функціональної схеми програми 20
2.2 Опис алгоритму роботи програми 26
2.3 Розробка інтерфейсу програми 31
3 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 36
3.1 Аналіз обраного середовища програмування 36
3.2 Розробка бази даних 38
3.3 Програмна реалізація основних функцій 41
3.4 Методика роботи користувача 50
ВИСНОВКИ 55
ПЕРЕЛІК ПОСИЛАНЬ 57
Додаток А 59
Додаток Б 68
Додаток В 70
Додаток Г 71
Додаток Д 72

ВСТУП

У сучасному дизайні колір відіграє ключову роль у формуванні естетичного сприйняття, емоційного впливу та функціональності продуктів. Зростання популярності хмарних платформ, таких як Figma, і потреба в швидкому створенні гармонійних колірних палітр зумовлюють актуальність автоматизації цього процесу. Розробка програмного модуля для підбору колірних схем, інтегрованого з Figma, дозволяє оптимізувати робочий процес дизайнерів, підвищити якість дизайну та забезпечити інклюзивність, враховуючи стандарти доступності та культурні особливості.

Мета роботи полягає в створенні плагіна для Figma, який автоматизує генерацію гармонійних колірних палітр на основі введеного базового кольору, забезпечує їх інтеграцію в робоче середовище та підтримує збереження для повторного використання.

Галузь застосування охоплює дизайн інтерфейсів, веброзробку, брендинг та графічний дизайн, де плагін може використовуватися дизайнерами для створення естетичних і функціональних продуктів.

Об’єкт розробки – програмний модуль (плагін) для автоматичного підбору колірних палітр у Figma.

Предмет розробки – алгоритми генерації гармонійних кольорів, їх інтеграція з Figma та збереження даних.

Завдання дослідження:

- вибір та обґрунтування напрямку рішення: обрати створення плагіна для Figma через її популярність та відсутність вбудованих інструментів для автоматичного підбору палітр; базувати рішення на теорії кольору та стандартах доступності WCAG;
- аналітичний огляд: провести аналіз літератури та інструментів; виявити їхні переваги та недоліки, що підтверджують потребу в новому рішенні;

- методика та зміст роботи: включити теоретичний аналіз, розробку алгоритмів, створення інтерфейсу, інтеграцію з Figma, збереження даних у локальному сховищі та тестування;
- архітектура ПЗ: побудувати плагін за клієнт-серверною моделлю з UI-компонентом для інтерфейсу та основним кодом для взаємодії з Figma API; забезпечити асинхронний обмін даними;
- програмні засоби: використати Visual Studio Code, JavaScript, TypeScript, HTML і CSS; оптимізувати продуктивність через відмову від фреймворків;
- структура даних: реалізувати базу даних через figma.clientStorage; зберігати палітри як масив об’єктів із полями id, colors, harmonyType, baseColor; забезпечити асинхронний доступ до даних;
- особливості реалізації: підтримати сім типів гармоній, включаючи адаптивну контрастну; створити мінімалістичний інтерфейс із HEX-полем, пікером і списком гармоній; оптимізувати алгоритми та типізувати код;
- особливості застосування: забезпечити введення базового кольору, генерацію палітр, їх застосування до об’єктів у Figma, збереження та видалення; включити в інтерфейс поле для HEX-коду, пікер, випадаючий список гармоній, кнопки управління та область збережених палітр; продемонструвати роботу через рисунки.

Розроблений плагін є практичним інструментом для дизайнерів, що підвищує ефективність роботи та якість дизайну, з потенціалом для комерційного використання та подальшого вдосконалення.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Критичний аналіз літературних джерел за темою

Підбір гармонійних колірних поєднань є ключовим елементом у дизайні інтерфейсів, графічному дизайні, веб-розробці та інших галузях, де візуальна естетика відіграє важливу роль. У сучасному світі, де швидкість і якість створення дизайну є критично важливими, інструменти для генерації колірних схем стали незамінними помічниками дизайнерів. Вони дозволяють створювати палітри, які відповідають як естетичним, так і функціональним вимогам проєктів. У цьому розділі проведено критичний аналіз літературних джерел, що стосуються теорії кольору, методів підбору колірних поєднань, а також огляд сучасних інструментів, таких як Adobe Color, Coolors, Paletton, Color Hunt та інших. Метою аналізу є визначення їхніх переваг, обмежень та потенційних напрямів для вдосконалення.

Теорія кольору становить основу для розробки інструментів підбору колірних схем. Одним із ключових внесків у цю галузь є праця Йоганнеса Іттена (1961), який у своїй книзі "Мистецтво кольору" детально описав принципи гармонії кольорів. Іттен запропонував використовувати колірне колесо як інструмент для створення різних типів колірних поєднань, таких як комплементарні, аналогічні, тріадичні та монохроматичні. Комплементарні кольори, розташовані навпроти один одного на колірному колесі, створюють сильний контраст, що ідеально підходить для привернення уваги. Аналогічні кольори, навпаки, забезпечують м’які та гармонійні переходи, що часто використовується в дизайні для створення спокійної атмосфери. Підхід Іттена залишається актуальним і широко застосовується в сучасних інструментах для підбору палітр [1].

Інший важливий внесок у теорію кольору зробив Джозеф Альберс (2006) у своїй роботі "Взаємодія кольорів". Альберс досліджував, як кольори

впливають один на одного залежно від їхнього оточення. Він довів, що один і той самий колір може виглядати по-різному в різних контекстах, що підкреслює важливість урахування сусідніх відтінків під час створення палітр. Цей принцип є основою для багатьох сучасних інструментів, таких як Adobe Color, які дозволяють дизайнерам експериментувати з кольорами в реальному часі, враховуючи їх взаємодію [2].

Серед сучасних досліджень варто відзначити роботу By Вонга (2011), який аналізує вплив культурних і психологічних факторів на сприйняття кольорів. Вонг наголошує, що вибір колірної палітри залежить не лише від естетичних принципів, а й від цільової аудиторії. Наприклад, у різних культурах одні й ті самі кольори можуть мати різні символічні значення, що необхідно враховувати під час створення дизайнів для глобального ринку. Ці дослідження підкреслюють необхідність створення інструментів, які б враховували культурний контекст і надавали дизайнерам гнучкі можливості для адаптації палітр [3].

Adobe Color є одним із найвідоміших інструментів для створення колірних палітр. Цей вебсервіс дозволяє користувачам вибирати базовий колір і генерувати схеми на основі різних правил гармонії, таких як аналогічні, монохроматичні, тріадичні, комплементарні та інші. Однією з головних переваг Adobe Color є його інтеграція з іншими продуктами Adobe, такими як Photoshop, Illustrator та InDesign. Це дозволяє дизайнерам легко переносити створені палітри в робоче середовище, що значно прискорює робочий процес. Крім того, Adobe Color пропонує бібліотеку готових палітр, створених спільнотою, що є джерелом натхнення для дизайнерів [9].

Проте Adobe Color має певні обмеження. Інтерфейс інструменту може бути складним для початківців, оскільки він вимагає базового розуміння теорії кольору. Наприклад, користувачам потрібно знати, як різні типи колірних схем впливають на сприйняття дизайну. Крім того, Adobe Color не пропонує прямого API для інтеграції з іншими платформами, такими як Figma чи Sketch, що обмежує його використання в сучасних робочих процесах, де потрібна

швидка адаптація інструментів. Ще одним недоліком є відсутність функцій для симуляції сприйняття кольорів людьми з дальтонізмом, що є важливим аспектом інклюзивного дизайну.

Coolors є популярним інструментом серед дизайнерів завдяки своїй простоті та швидкості роботи. Користувачі можуть генерувати палітри випадковим чином, натискаючи клавішу пробілу, або вводити базовий колір для створення гармонійних поєднань. Coolors підтримує експорт палітр у різні формати, такі як PNG, PDF, CSS і SCSS, що робить його універсальним для використання в веб-дизайні, графічному дизайні та інших галузях. Додатковою перевагою є мобільний додаток Coolors, який дозволяє створювати палітри на основі фотографій, що особливо зручно для дизайнерів, які працюють у дорозі [10].

Незважаючи на ці переваги, Coolors має обмеження в плані глибини налаштувань. Інструмент не дозволяє користувачам детально налаштовувати алгоритми генерації палітр, що може бути недоліком для професійних дизайнерів, які прагнуть точного контролю над колірними поєднаннями. Крім того, деякі функції, такі як розширені можливості експорту чи доступ до бібліотеки палітр, доступні лише в платній версії, що може бути перешкодою для користувачів із обмеженим бюджетом. Як і Adobe Color, Coolors не підтримує прямої інтеграції з Figma, що змушує дизайнерів вручну переносити палітри.

Paletton є інструментом, який базується на колірному колесі та пропонує створення складних колірних схем, таких як тетрадичні, акцентні та вільні поєднання. Його сильною стороною є детальний підхід до теорії кольору, що дозволяє користувачам глибше зрозуміти принципи гармонії. Paletton також підтримує симуляцію сприйняття кольорів для людей із різними формами дальтонізму, що робить його цінним інструментом для інклюзивного дизайну. Крім того, Paletton дозволяє користувачам зберігати створені палітри та експортувати їх у різні формати [11].

Однак Paletton має недоліки, пов’язані з його інтерфейсом, який виглядає застарілим порівняно з сучасними інструментами, такими як Coolors. Це може відлякувати нових користувачів, які звикли до більш інтуїтивних і візуально привабливих інтерфейсів. Крім того, Paletton не пропонує інтеграції з популярними дизайнерськими платформами, такими як Figma, Sketch чи Adobe XD, що знижує його привабливість для сучасних робочих процесів. Ще одним обмеженням є відсутність функцій для роботи зі спільнотою, таких як бібліотека користувацьких палітр, що робить Paletton менш гнучким порівняно з Adobe Color чи Color Hunt.

Color Hunt є платформою, орієнтованою на спільноту, де користувачі діляться власними палітрами. Це робить інструмент унікальним джерелом натхнення для дизайнерів, які шукають готові колірні рішення. Перевагою Color Hunt є його простота та акцент на трендових кольорах, що відповідають сучасним дизайнерським тенденціям. Платформа дозволяє користувачам фільтрувати палітри за популярністю, новизною чи певними тегами, що полегшує пошук потрібних поєднань [12].

Основним недоліком Color Hunt є відсутність функцій для створення власних палітр на основі заданого кольору. Інструмент більше підходить для пошуку готових рішень, ніж для активної роботи з кольорами, що обмежує його використання в професійних проєктах. Крім того, Color Hunt не пропонує інтеграції з дизайнерськими платформами чи інструментами, що змушує користувачів вручну копіювати коди кольорів. Це може бути незручно для проєктів, які потребують швидкого робочого процесу.

Порівнюючи згадані інструменти, можна виділити кілька ключових аспектів. Adobe Color і Paletton пропонують глибокий підхід до теорії кольору, що робить їх придатними для професійних дизайнерів, які прагнуть точного контролю над палітрами. Однак їхні інтерфейси та обмежена інтеграція з сучасними платформами можуть ускладнювати використання. Coolors вирізняється простотою та універсальністю, що робить його ідеальним для швидкого створення палітр, але брак гнучкості в налаштуваннях обмежує його

можливості. Color Hunt є чудовим джерелом натхнення, але не підходить для створення унікальних палітр чи інтеграції в робочі процеси.

Загальним недоліком більшості інструментів є відсутність прямої інтеграції з дизайнерськими платформами, такими як Figma, Sketch чи Adobe XD. Це змушує дизайнерів витрачати додатковий час на перенесення палітр, що знижує ефективність робочого процесу. Крім того, лише Paletton пропонує функції для інклюзивного дизайну, такі як симуляція дальтонізму, тоді як інші інструменти ігнорують цей важливий аспект. Ще одним недоліком є обмежена адаптація до культурних контекстів, що може бути проблемою для проєктів, орієнтованих на глобальну аудиторію [13].

Аналіз літературних джерел і сучасних інструментів для підбору колірних схем показує, що існуючі рішення пропонують широкий спектр можливостей для дизайнерів, але мають певні обмеження. Теорія кольору, розроблена такими авторами, як Іттен, Альберс і Вонг, залишається основою для створення гармонійних палітр, але потребує адаптації до сучасних технологій і вимог. Інструменти, такі як Adobe Color і Coolors, забезпечують зручність і швидкість, але брак інтеграції з популярними дизайнерськими платформами та обмежені можливості налаштування знижують їхню ефективність. Paletton вирізняється підтримкою інклюзивного дизайну, але його застарілий інтерфейс і відсутність інтеграцій є недоліками. Color Hunt є цінним джерелом натхнення, але не підходить для створення унікальних палітр.

Ці висновки підкреслюють актуальність розробки нового програмного модуля, який би поєднував простоту використання, глибокі можливості налаштування, пряму інтеграцію з Figma та підтримку інклюзивного дизайну. Такий інструмент міг би заповнити прогалини, виявлені в аналізі, і стати цінним рішенням для сучасних дизайнерів, які прагнуть оптимізувати свій робочий процес і створювати високоякісні дизайн-проєкти [14].

1.2 Актуальність теми кваліфікаційної роботи

У сучасному дизайні колір є ключовим елементом, що впливає на сприйняття продукту, емоції користувачів і ефективність візуальної комунікації. Зі зростанням попиту на швидкі та якісні дизайнерські рішення інструменти для автоматичного підбору колірних палітр стають дедалі актуальнішими. Розробка програмного модуля для створення гармонійних колірних поєднань, інтегрованого в платформу Figma, відповідає сучасним викликам дизайну, оптимізуючи робочий процес і підвищуючи продуктивність [15].

Колір значно впливає на поведінку та сприйняття користувачів. Дослідження Вонга (2011) показують, що правильно підібрані кольори підвищують залученість і сприяють позитивному користувацькому досвіду. Наприклад, синій асоціюється з довірою, а яскраві кольори, як червоний, привертають увагу. За даними Nielsen Norman Group (2023), перше враження про вебсайт формується за 50 мілісекунд, де колір відіграє вирішальну роль. Неправильна палітра може знизити конверсію або відштовхнути користувачів, що підкреслює важливість інструментів для швидкого створення ефективних поєднань [4].

Сучасні дизайнери працюють у стислі терміни, часто використовуючи хмарні платформи, такі як Figma, яка, за звітом Figma State of Design (2023), є основним інструментом для 70% професіоналів [5]. Figma оптимізує командну роботу, але не має вбудованого інструменту для автоматичного підбору палітр, що змушує дизайнерів звертатися до сторонніх сервісів, таких як Adobe Color чи Coolors. Ці сервіси мають обмеження, зокрема відсутність інтеграції з Figma та недостатню гнучкість. Плагін для Figma, який автоматизує підбір кольорів, усуває ці проблеми, заощаджуючи час і підтримуючи інклюзивний дизайн, наприклад, адаптацію палітр для людей із дальтонізмом відповідно до стандартів WCAG [7].

Глобалізація дизайнерських проєктів додає актуальності темі. Кольори мають різні культурні значення, наприклад, білий символізує чистоту в

західних культурах, але асоціюється з трауром в азійських. Інструмент, який враховує ці особливості, полегшує створення продуктів для міжнародного ринку. Крім того, звіт Design Trends 2024 від Dribbble вказує на попит на унікальні колірні палітри, що допомагають брендам виділятися. Плагін, який генерує трендові палітри, відповідає цій потребі [6].

З технологічної точки зору, сучасні фреймворки, такі як JavaScript і TypeScript, дозволяють створювати ефективні плагіни для Figma. Алгоритми на основі теорії кольору забезпечують точність підбору поєднань. Економічно розробка плагіна є виправданою, оскільки, за даними Marketplace Figma (2023), інструменти автоматизації мають високий попит і комерційний потенціал [8].

Отже, розробка модуля для підбору колірних палітр є актуальною через критичну роль кольору в дизайні, популярність Figma, глобалізацію проєктів і потребу в персоналізованих рішеннях. Такий інструмент підвищить ефективність дизайнерів, сприятиме інклюзивності та відкриє перспективи для подальших досліджень у сфері автоматизації дизайну.

1.3 Цілі та завдання кваліфікаційної роботи

Кваліфікаційна робота присвячена розробці програмного модуля для автоматичного підбору гармонійних кольорових палітр із можливістю інтеграції з графічним редактором Figma. У сучасному дизайні інтерфейсів, брендингу та графічних проєктів вибір кольорів відіграє ключову роль, оскільки кольорові схеми впливають на сприйняття продукту, емоційний відгук користувачів і загальну естетику. Однак процес створення гармонійних палітр часто є складним і вимагає значних часових затрат, особливо для дизайнерів-початківців. У цьому контексті автоматизація підбору кольорів за допомогою програмного модуля, інтегрованого в популярну платформу дизайну, є актуальним завданням, що має практичну цінність.

Метою цієї кваліфікаційної роботи є створення ефективного програмного інструменту, який спрощує процес вибору кольорових палітр,

забезпечує їх гармонійність і дозволяє дизайнерам швидко застосовувати згенеровані кольори в реальних проєктах. Для досягнення цієї мети визначено низку цілей і завдань, які охоплюють розробку, інтеграцію, тестування та аналіз ефективності інструменту. У цьому розділі чітко окреслено основні цілі роботи та завдання, які необхідно виконати для їх реалізації.

Основні цілі кваліфікаційної роботи:

1. створення програмного модуля для підбору кольорів. Основною метою роботи є розробка програмного модуля, який здатен генерувати гармонійні кольорові палітри на основі введеного користувачем базового кольору. Модуль має використовувати складні алгоритми підбору кольорів, які враховують принципи колірної гармонії, такі як комплементарні, аналогові, тріадні та адаптивні контрастні схеми. Ціль полягає в тому, щоб забезпечити створення естетично приємних і функціональних палітр, які відповідають потребам дизайнерів у різних сферах, від вебдизайну до ілюстрацій. Модуль має бути інтуїтивно зрозумілим, швидким і гнучким, дозволяючи користувачам обирати різні типи гармоній залежно від їхніх творчих завдань;

2. інтеграція програмного модуля з Figma. Другою ключовою ціллю є інтеграція розробленого модуля з платформою Figma, яка є одним із провідних інструментів для дизайну інтерфейсів і прототипування. Інтеграція передбачає створення плагіна, який дозволяє дизайнерам безпосередньо в середовищі Figma генерувати кольорові палітри, зберігати їх і застосовувати до вибраних об’єктів на полотні. Ціль полягає в тому, щоб забезпечити безшовну взаємодію між модулем і Figma, мінімізуючи необхідність використання сторонніх інструментів для створення палітр. Плагін має бути сумісним із API Figma, підтримувати збереження даних у локальному сховищі та забезпечувати стабільну роботу в різних сценаріях дизайну;

3. тестування ефективності інструменту. Третьою метою є оцінка ефективності розробленого програмного модуля шляхом тестування його функціональності, продуктивності та зручності використання.

Для досягнення поставлених цілей визначено низку конкретних завдань, які охоплюють усі етапи розробки, інтеграції, тестування та аналізу програмного модуля. Ці завдання структуровано таким чином, щоб забезпечити послідовне виконання роботи та досягнення бажаних результатів.

Завдання, пов’язані зі створенням програмного модуля:

1. аналіз принципів колірної гармонії. Першим завданням є дослідження теоретичних основ колірної гармонії, включаючи принципи побудови комплементарних, аналогових, тріадних, монохроматичних і адаптивних контрастних палітр. Необхідно вивчити моделі кольорів (RGB, HSL), методи обчислення контрастності (зокрема за стандартами WCAG) і алгоритми адаптивного підбору кольорів. Це завдання включає аналіз сучасних інструментів для генерації палітр (наприклад, Adobe Color, Coolors) для визначення найкращих практик;

2. розробка алгоритмів підбору кольорів. На основі проведеного аналізу необхідно розробити набір алгоритмів для генерації гармонійних палітр. Основний акцент зроблено на створенні складного алгоритму “Адаптивна контрастна гармонія”, який враховує світлість і насиченість базового кольору, забезпечує контрастність (≥ 3:1 за WCAG) і генерує п’ять кольорів із різними відтінками. Завдання включає програмну реалізацію алгоритмів у JavaScript, використовуючи перетворення між HEX, RGB і HSL, а також перевірку контрастності за формулою відносної яскравості;

3. проєктування інтерфейсу модуля. Завдання полягає в розробці зручного та інтуїтивно зрозумілого інтерфейсу для взаємодії з модулем. Інтерфейс має включати поле для введення HEX-коду, колірний пікер, вибір типу гармонії, кнопки для генерації, збереження та застосування палітр, а також секцію для відображення збережених палітр. Інтерфейс має бути виконаний у стилі сучасних дизайн-систем, з акцентом на мінімалізм і чіткість;

4. вивчення API Figma. Для успішної інтеграції необхідно вивчити документацію Figma Plugin API, щоб зрозуміти, як створювати плагіни, обробляти взаємодію з полотном, застосовувати кольори до об’єктів і

зберігати дані в локальному сховищі (figma.clientStorage). Це завдання включає аналіз прикладів плагінів і вимог до їхньої структури (наприклад, manifest.json, ui.html, code.js);

5. розробка плагіна для Figma. Завдання передбачає створення плагіна, який інтегрує програмний модуль із Figma. Плагін має забезпечувати:

- відображення інтерфейсу модуля в Figma;
- генерацію палітр на основі введеного кольору;
- застосування згенерованих кольорів до виділених об’єктів на полотні;
- збереження палітр у локальному сховищі та їхнє відображення в інтерфейсі;
- можливість видалення збережених палітр. Плагін розробляється з використанням HTML, CSS і JavaScript, із підтримкою асинхронного обміну даними між інтерфейсом і кодом плагіна;

6. оптимізація продуктивності плагіна. Для забезпечення швидкої роботи плагіна необхідно оптимізувати алгоритми генерації палітр і обробки даних. Завдання включає мінімізацію обчислювальних затрат у функціях перетворення кольорів, обмеження кількості ітерацій у алгоритмах перевірки контрасту та забезпечення стабільної роботи при обробці великих обсягів даних (наприклад, кількох палітр);

7. проведення тестування. На основі методології необхідно провести тестування плагіна. Завдання включає:

- тестування на різних HEX-кодах (світлі, темні, низьконасичені кольори);
- перевірку роботи плагіна в різних сценаріях (наприклад, застосування палітри до кількох об’єктів);
- оцінку стабільності (відсутність зависань або помилок);
- збір відгуків від дизайнерів щодо зручності інтерфейсу та якості палітр;

8. аналіз результатів і вдосконалення. Після тестування необхідно проаналізувати отримані дані, виявити недоліки (наприклад, низький контраст окремих кольорів, повільна робота алгоритмів) і внести відповідні виправлення. Завдання включає вдосконалення алгоритмів, оптимізацію інтерфейсу та повторне тестування для підтвердження покращень.

Виконання поставлених цілей і завдань дозволить створити програмний модуль, який:

- генерує гармонійні кольорові палітри з використанням складних адаптивних алгоритмів;
- безшовно інтегрується з Figma через плагін, надаючи дизайнерам зручний інструмент для роботи з кольорами;
- забезпечує високу ефективність, стабільність і зручність використання, що підтверджується результатами тестування.

Розроблений інструмент матиме практичну цінність для дизайнерів, дозволяючи економити час на створенні палітр і підвищуючи якість їхніх проєктів. Крім того, кваліфікаційна робота сприятиме поглибленню знань у сфері програмування, дизайну та інтеграції програмного забезпечення з хмарними платформами.

Цілі та завдання кваліфікаційної роботи чітко структуровано для забезпечення послідовного виконання всіх етапів розробки. Створення програмного модуля для підбору кольорів, його інтеграція з Figma та тестування ефективності є ключовими аспектами, які дозволять досягти практичних і теоретичних результатів. Виконання цих завдань сприятиме створенню інструменту, який відповідає сучасним потребам дизайнерів і може бути використаний у реальних проєктах.

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1 Опис функціональної схеми програми

Розроблюваний плагін для Figma, призначений для автоматичного підбору гармонійних кольорових палітр, буде програмним модулем, який інтегруватиметься з графічним редактором Figma через його Plugin API. Плагін надаватиме користувачам зручний інструмент для генерації палітр, їх збереження, застосування до об’єктів на полотні Figma та видалення збережених палітр. У цьому розділі детально описано функціональну схему програми, включаючи її плановану архітектуру, основні функції, взаємодію компонентів і принципи роботи. Функціональна схема відображає, як плагін оброблятиме введені дані, виконуватиме обчислення та взаємодіятиме з Figma, щоб забезпечити безшовний досвід для дизайнерів.

Плагін планується побудувати за клієнт-серверною моделлю, де клієнтська частина відповідатиме за інтерфейс користувача, а серверна частина (в контексті Figma — основний код плагіна) взаємодіятиме з API Figma та оброблятиме операції з полотном і локальним сховищем. Архітектура плагіна складатиметься з двох основних компонентів. Перший — UI-компонент (ui.html), який відповідатиме за відображення інтерфейсу користувача, обробку введених даних і виконання обчислень для генерації палітр. Цей компонент буде написано на HTML, CSS і JavaScript і міститиме графічний інтерфейс із полем для введення HEX-коду, колірним пікером, вибором типу гармонії, кнопками управління та відображенням палітр. Другий — основний код (code.js), який виконуватиме функції, пов’язані з Figma, такі як застосування кольорів до виділених об’єктів, збереження палітр у локальному сховищі (figma.clientStorage) і обробка повідомлень від UI-компонента. Цей код буде написано на JavaScript із використанням Figma Plugin API для взаємодії з полотном [18].

Компоненти взаємодіятимуть через асинхронний обмін повідомленнями, використовуючи методи parent.postMessage (з UI до code.js)

і figma.ui.onmessage (з code.js до UI). Конфігурація плагіна визначатиметься у файлі manifest.json, який указуватиме основні файли (ui.html, code.js) і параметри плагіна.

Плагін виконуватиме низку ключових функцій, які забезпечуватимуть його цінність для дизайнерів. До них входитимуть генерація палітр, їх застосування, збереження, відображення та видалення. Кожна функція реалізовуватиметься через взаємодію UI-компонента та основного коду. Основні функції плагіна включають наступне. По-перше, генерація кольорових палітр дозволятиме створювати гармонійні палітри на основі введеного базового кольору. Користувач вводитиме HEX-код або обиратиме колір через колірний пікер, а потім вибиратиме тип гармонії (наприклад, комплементарна, аналогова, адаптивна контрастна). Алгоритми, реалізовані в ui.html, обчислюватимуть гармонійні кольори. Наприклад, алгоритм “Адаптивна контрастна гармонія” генеруватиме п’ять кольорів, враховуючи відтінок, насиченість, світлість і контрастність за стандартами WCAG. По-друге, застосування палітр дозволятиме користувачу застосовувати згенеровані палітри до виділених об’єктів на полотні Figma (наприклад, прямокутників, кіл), де кольори палітри застосовуватимуться циклічно до заповнень (fills) об’єктів через Figma API. По-третє, збереження палітр дозволятиме користувачу зберігати палітри для подальшого використання в локальному сховищі Figma через figma.clientStorage. По-четверте, відображення та застосування збережених палітр передбачатиме показ списку збережених палітр із прев’ю кольорів, де користувач зможе клацнути на палітру, щоб відновити її в інтерфейсі. По-п’яте, видалення палітр дозволятиме користувачу видаляти збережені палітри, оновлюючи список відображених палітр.

Функціональна схема плагіна ґрунтуватиметься на взаємодії між UI-компонентом і основним кодом через обмін повідомленнями. Нижче описано, як ці компоненти співпрацюватимуть для виконання ключових функцій, згрупованих у три основні категорії:

1. генерація палітр. Користувач розпочинатиме процес, вводячи базовий колір у вигляді HEX-коду через текстове поле baseColor або обираючи колір за допомогою колірного пікера colorPicker у графічному інтерфейсі, а також вибираючи тип гармонії з випадаючого списку harmonyType. У UI-компоненті функція generatePalette перевірятиме валідність HEX-коду за допомогою регулярного виразу, щоб переконатися, що він відповідає формату #RRGGBB. Далі викликатиметься функція hexToHSL, яка конвертуватиме HEX-код у модель HSL для зручності обчислень. На основі обраного типу гармонії функція generateHarmony створюватиме набір гармонійних кольорів. Наприклад, для адаптивної контрастної гармонії генеруватиметься п’ять кольорів із коригуванням контрасту через функцію adjustForContrast, яка обчислюватиме відносну яскравість і коригуватиме світлість, якщо співвідношення контрасту менше 3. Згенерована палітра відображатиметься в контейнері palette як набір кольорових прямокутників, кожен із яких міститиме відповідний HEX-код. Усі обчислення та відображення виконуватимуться в UI-компоненті, тому взаємодія з code.js на цьому етапі не знадобиться;

2. застосування та збереження палітр. Коли користувач натискатиме кнопку “Застосувати до виділення” після генерації палітри, у UI викликатиметься функція applyToSelection, яка перевірятиме наявність згенерованої палітри в змінній currentPalette і надсилатиме повідомлення до code.js через parent.postMessage із типом applyPalette та даними палітри. У code.js це повідомлення оброблятиметься через figma.ui.onmessage, перевірятиметься наявність виділених об’єктів на поточній сторінці через figma.currentPage.selection, а для кожного об’єкта з властивістю fills HEX-коди палітри конвертуватимуться в нормалізований RGB-формат (0–1) і застосовуватимуться як заповнення через node.fills. Користувачу відображатиметься сповіщення через figma.notify про успішне застосування. Для збереження палітри користувач натискатиме кнопку “Зберегти палітру”, і функція savePalette створюватиме об’єкт із даними палітри, включаючи

унікальний ідентифікатор, список кольорів, тип гармонії та базовий колір. Ці дані надсилатимуться до code.js через parent.postMessage із типом savePalette. У code.js дані додаватимуться до масиву в figma.clientStorage за допомогою setAsync, а оновлений список палітр надсилатиметься назад до UI через postMessage із типом palettesLoaded. У UI функція renderSavedPalettes оновлюватиме контейнер savedPalettes, відображаючи збережені палітри;

3. відображення та видалення палітр. Якщо користувач клацатиме на збережену палітру в контейнері savedPalettes, функція applySavedPalette у UI відновлюватиме дані палітри, такі як currentPalette, baseColor і harmonyType, і викликатиме generatePalette для відображення палітри в інтерфейсі, як при первинній генерації. Цей процес не потребуватиме взаємодії з code.js, оскільки вся обробка відбуватиметься в UI. Для видалення палітри користувач натискатиме кнопку “Видалити”, і функція deletePalette надсилатиме повідомлення до code.js через parent.postMessage із типом deletePalette та ідентифікатором палітри, використовуючи event.stopPropagation, щоб уникнути випадкового виклику applySavedPalette. У code.js палітра видалятиметься з figma.clientStorage шляхом фільтрації масиву за ідентифікатором, а оновлений список палітр надсилатиметься до UI через postMessage. У UI функція renderSavedPalettes оновлюватиме контейнер savedPalettes, прибираючи видалену палітру з інтерфейсу.

Основна логіка генерації палітр зосереджуватиметься в UI-компоненті, зокрема у функціях hexToHSL, hslToHex, calculateLuminance, adjustForContrast і generateHarmony. Нижче описано принципи роботи ключового алгоритму “Адаптивна контрастна гармонія” та інших алгоритмів, які планується реалізувати.

Плагін міститиме сім алгоритмів генерації палітр: комплементарна, аналогова, тріадна, розділена комплементарна, монохроматична, м’яка гармонія та адаптивна контрастна. Кожен алгоритм базуватиметься на принципах колірної гармонії, використовуватиме модель HSL (Hue, Saturation, Lightness) для маніпуляцій із кольорами та генеруватиме палітри, які

відповідатимуть різним дизайнерським потребам. Нижче наведено принципи роботи кожного алгоритму, які будуть реалізовані у функції generateHarmony файлу ui.html.

Комплементарний алгоритм генеруватиме два кольори: базовий і його протилежний на колірному колі. Принцип роботи полягатиме в додаванні 180° до відтінку (H) базового кольору в HSL, зберігаючи однакові насиченість (S) і світлість (L). Наприклад, для базового кольору з H = 204° комплементарний колір матиме H = (204 + 180) % 360 = 24°. Цей алгоритм створюватиме контрастну палітру, ідеальну для виділення елементів дизайну, таких як кнопки чи акценти. Аналоговий алгоритм створюватиме три кольори, розташовані поруч на колірному колі, для м'якого і злагодженого вигляду. Базовий колір залишатиметься без змін, а два додаткові кольори генеруватимуться шляхом зсуву відтінку на +30° і -30° (наприклад, H = 204° → 234° і 174°), зберігаючи насиченість і світлість. Ця гармонія підходитиме для спокійних дизайнів, таких як фони чи градієнти. Тріадний алгоритм генеруватиме три кольори, рівномірно розподілені на колірному колі (з інтервалом 120°), додаючи до базового відтінку 120° і 240° (наприклад, H = 204° → 324° і 84°), зберігаючи насиченість і світлість. Цей алгоритм створюватиме збалансовану палітру для діаграм чи ілюстрацій.

Алгоритм розділеної комплементарної гармонії генеруватиме три кольори: базовий і два кольори, розташовані поруч із комплементарним. Відтінки зі зсувом на +150° і +210° (наприклад, H = 204° → 354° і 54°) зберігатимуть насиченість і світлість, забезпечуючи м'якший контраст, ніж комплементарна гармонія. Монохроматичний алгоритм генеруватиме п'ять кольорів, варіюючи лише насиченість і світлість базового кольору, створюючи відтінки зі світлістю (L + 20, L - 20) і насиченістю (S + 20, S - 20). Ця гармонія підходитиме для мінімалістичних дизайнів. Алгоритм м'якої гармонії генеруватиме три кольори, комбінуючи аналогові відтінки (+30°, -30°) зі змінами насиченості (-15%, +10%) і світлості (+10%, -15%), створюючи ніжні палітри для елегантних дизайнів. Адаптивна контрастна гармонія

генеруватиме п’ять кольорів із адаптивними змінами: аналоговий м’який (H + 40°, S - 20, L + 15), контрастний (H + 180°, адаптивна S/L, контраст ≥ 3:1), аналоговий контрастний (H - 40°, S + 15, L - 10), нейтральний акцент (H, S = 20, L = 80/20). Наприклад, для H = 204, S = 70, L = 53 генеруватимуться #3498db, #4ca2e6, #db6e34 (з коригуванням контрасту), #2c82b3, #b3c6d9. Цей алгоритм буде ідеальним для доступних інтерфейсів.

Таблиця 2.1 демонструє методи генерування в плагіні.

Таблиця 2.1 – Методи генерування палітр

Алгоритм	**Кількість кольорів**	**Принцип роботи**	**Застосування**
Комплементарна	2	Додає 180° до відтінку базового кольору, зберігаючи S і L	Контрастні акценти, кнопки
Аналогова	3	Зсуви відтінку на +30° і -30°, зберігає S і L	Спокійні фони, градієнти
Тріадна	3	Зсуви відтінку на +120° і +240°, зберігає S і L	Різноманітні діаграми, ілюстрації
Розділена комплементарна	3	Зсуви відтінку на +150° і +210°, зберігає S і L	Контраст із м’яким ефектом
Монохроматична	5	Змінює S (±20) і L (±20) базового кольору, зберігає H	Мінімалістичні інтерфейси, брендинг
М’яка гармонія	3	Зсуви H на ±30°, S ±10/15, L ±10/15	Елегантні, ненав’язливі дизайни
Адаптивна контрастна	5	Зсуви H (±40°, 180°), адаптивні S/L, контраст ≥ 3:1 для одного кольору	Доступні інтерфейси, універсальні палітри

UML діаграма плагіна показана на рисунку 2.1.

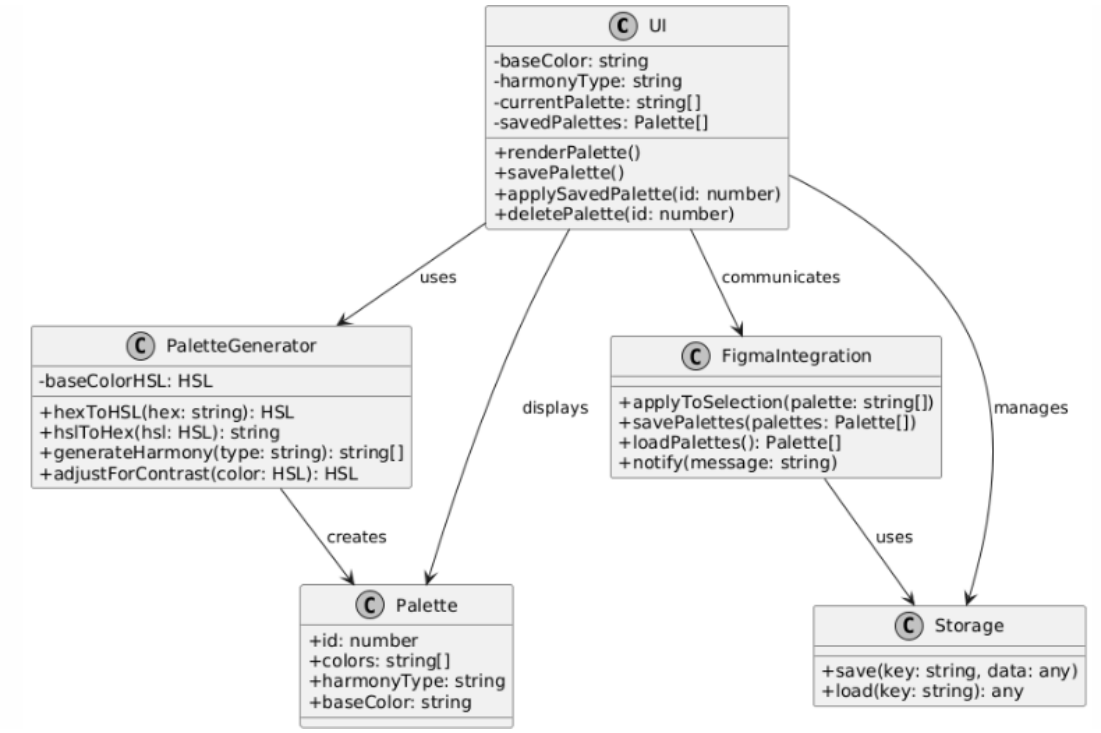

Рисунок 2.1 — UML діаграма проекту

Функціональна схема плагіна забезпечуватиме чітку взаємодію між UI-компонентом і основним кодом, дозволяючи ефективно виконувати генерацію, застосування, збереження та видалення палітр. Архітектура, побудована на асинхронному обміні повідомленнями, гарантуватиме безшовну інтеграцію з Figma і стабільну роботу. Алгоритми генерації палітр, зокрема “Адаптивна контрастна гармонія”, створюватимуть гармонійні та контрастні кольори, що відповідатимуть потребам дизайнерів.

2.2 Опис алгоритму роботи програми

Програмний модуль, реалізований у вигляді плагіна для Figma, призначений для автоматичного підбору гармонійних кольорових палітр на основі введеного користувачем базового кольору. Алгоритм роботи програми є центральним елементом плагіна, оскільки він забезпечує генерацію естетично приємних і функціональних кольорових схем, які можуть бути використані в дизайні інтерфейсів, брендингу чи графічних проєктах. У цьому розділі детально описано алгоритм підбору кольорів, включаючи етапи

введення даних, обробки, генерації гармонійних кольорів і їх відображення в інтерфейсі Figma. Особливу увагу приділено алгоритму “Адаптивна контрастна гармонія”, який є ключовим у контексті складного підбору кольорів.

Алгоритм роботи плагіна можна поділити на кілька основних етапів:

1. введення базового кольору: Користувач вводить HEX-код кольору або обирає його через колірний пікер у графічному інтерфейсі плагіна;
2. вибір типу гармонії: Користувач обирає один із доступних типів гармонії (наприклад, комплементарна, аналогова, тріадна, адаптивна контрастна);
3. обробка даних: Програма конвертує введений колір у формат HSL для аналізу відтінку, насиченості та світлості, після чого застосовує відповідний алгоритм для генерації гармонійних кольорів;
4. генерація палітри: На основі обраного типу гармонії програма створює набір кольорів, які відповідають принципам колірної гармонії та контрастності;
5. відображення та взаємодія: Згенерована палітра відображається в інтерфейсі плагіна, і користувач може зберегти її, застосувати до об’єктів у Figma або видалити збережені палітри.

Ці етапи реалізовано через взаємодію двох компонентів плагіна: UI-компонента (ui.html), який обробляє введення та генерацію палітр, і основного коду (code.js), який відповідає за інтеграцію з Figma.

Алгоритм створення гармонійної кольорової палітри в плагіні для Figma є комплексним процесом, який охоплює введення даних, обробку кольорів, генерацію палітри та інтеграцію з графічним редактором. Нижче подано детальний опис алгоритму в текстовій формі.

Алгоритм починається з отримання базового кольору від користувача через графічний інтерфейс, реалізований у файлі ui.html. Користувач може вказати колір у форматі HEX-коду (наприклад, #3498db) через текстове поле або обрати його візуально за допомогою колірного пікера.

На рисунку 2.2 зображено блок-схему роботи програми.

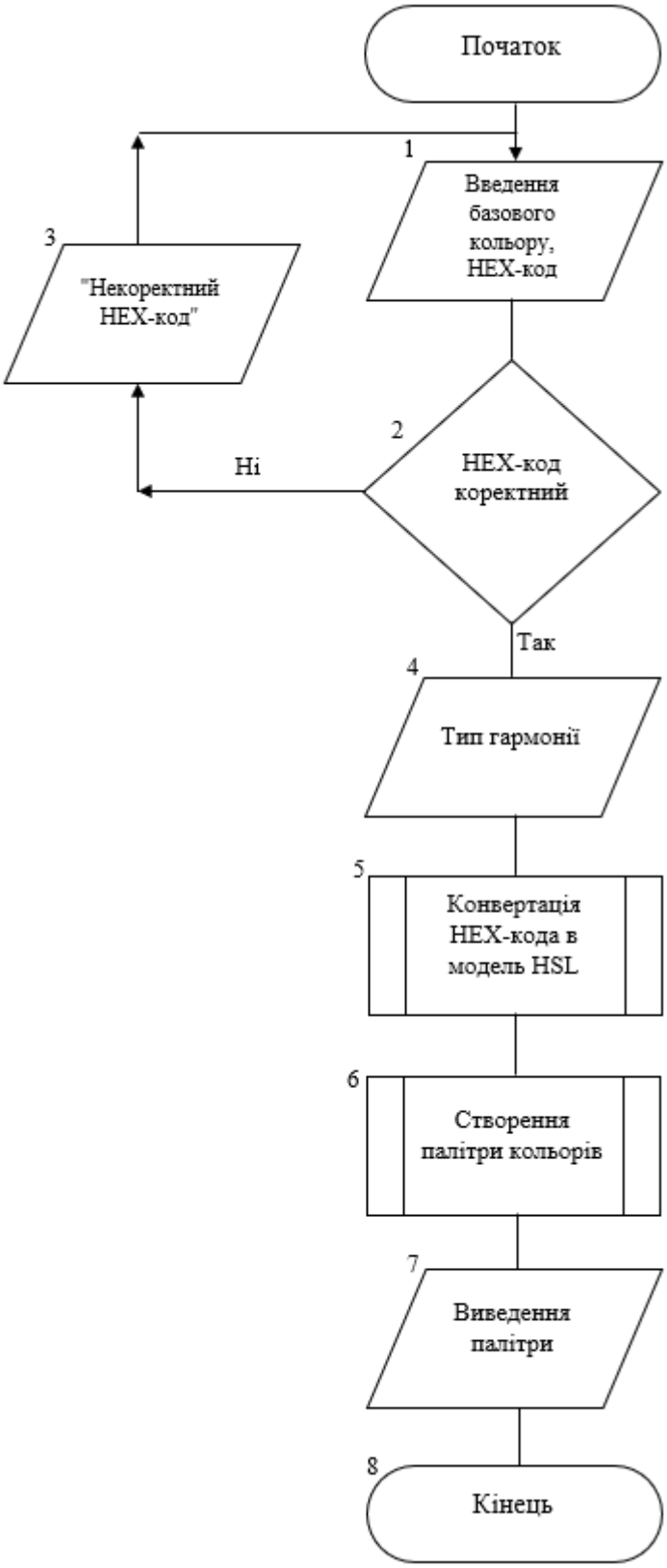

Рисунок 2.2— Блок-схема роботи програми

Алгоритм починається з отримання базового кольору від користувача через графічний інтерфейс, реалізований у файлі ui.html. Користувач може вказати колір у форматі HEX-коду (наприклад, #3498db) через текстове поле або обрати його візуально за допомогою колірного пікера. Ці два елементи синхронізовані: зміна HEX-коду в текстовому полі оновлює пікер, а вибір кольору в пікері змінює значення в текстовому полі. Щоб забезпечити коректність введення, JavaScript-функція перевіряє HEX-код за допомогою регулярного виразу, який підтверджує формат шести символів після знака #, що складаються з цифр або літер A–F. У разі некоректного введення користувачу відображається повідомлення про помилку через механізм parent.postMessage, який передає сповіщення до інтерфейсу плагіна. На виході цього етапу отримуємо валідний HEX-код, наприклад #3498db, який використовується для подальшої обробки.

Далі користувач обирає тип гармонії через випадаючий список у інтерфейсі, який пропонує варіанти, такі як комплементарна, аналогова, тріадна, розділена комплементарна, монохроматична, м’яка гармонія або адаптивна контрастна. Вибір фіксується функцією generatePalette, яка активується після натискання кнопки “Згенерувати палітру”. Обраний тип гармонії, представлений у вигляді рядка (наприклад, adaptive-contrast), визначає алгоритм, що буде застосовано в функції generateHarmony для створення палітри.

Для зручності маніпуляцій із кольором базовий HEX-код конвертується в модель HSL (Hue, Saturation, Lightness). Цей процес виконує функція hexToHSL. Спочатку HEX-код розбивається на три компоненти — червоний, зелений і синій, які переводяться з шістнадцяткової в десяткову систему (наприклад, #3498db дає значення 52, 152, 219). Ці значення нормалізуються діленням на 255, щоб отримати діапазон 0–1 (0.204, 0.596, 0.859). На основі нормалізованих значень обчислюються параметри HSL: світлість як середнє максимуму та мінімуму RGB, насиченість залежно від їхньої різниці, а відтінок — залежно від максимального компонента за відповідною формулою.

Для #3498db результатом є приблизно H = 204°, S = 70%, L = 53%, які повертаються як об’єкт для подальшого використання.

Основна логіка алгоритму реалізована у функції generateHarmony, яка створює палітру з п’яти кольорів на основі обраного типу гармонії. Для прикладу детально розглянемо адаптивну контрастну гармонію, яка враховує як гармонійність, так і контрастність. На вході функція отримує HSL базового кольору (наприклад, { h: 204, s: 70, l: 53 }) і його HEX-код. Створюється масив, що починається з базового кольору. Далі генеруються чотири додаткові кольори. Перший — аналоговий м’який: відтінок зміщується на 40° (244°), насиченість зменшується до 50%, а світлість зростає до 68%, що дає { h: 244, s: 50, l: 68 }. Другий — контрастний: відтінок зміщується на 180° (24°), насиченість залишається 70%, а світлість знижується до 33%, що дає { h: 24, s: 70, l: 33 }. Третій — аналоговий контрастний: відтінок зміщується на -40° (164°), насиченість зростає до 85%, а світлість знижується до 43%, що дає { h: 164, s: 85, l: 43 }. Четвертий — нейтральний акцент: відтінок залишається 204°, насиченість знижується до 20%, а світлість встановлюється на 80%, що дає { h: 204, s: 20, l: 80 }. Усі HSL-кольори конвертуються назад у HEX за допомогою функції hslToHex, формуючи палітру, наприклад, [#3498db, #4ca2e6, #db6e34, #2c82b3, #b3c6d9]. Для контрастного кольору додатково перевіряється контрастність із базовим кольором шляхом обчислення відносної яскравості та співвідношення контрасту. Якщо контрастність нижча за 3, світлість коригується до 10 ітерацій, щоб досягти потрібного значення.

Згенерована палітра відображається в інтерфейсі у вигляді п’яти кольорових прямокутників у контейнері . Кожен прямокутник має фоновий колір, заданий через CSS, і текстовий напис із HEX-кодом. HTML-розмітка генерується динамічно функцією generatePalette, що забезпечує миттєве оновлення інтерфейсу.

На завершальному етапі користувач може взаємодіяти з палітрою в Figma. Натискання кнопки “Застосувати до виділення” викликає функцію applyToSelection, яка надсилає палітру до code.js через parent.postMessage. У

code.js кольори конвертуються в нормалізований RGB і застосовуються до node.fills виділених об’єктів. Кнопка “Зберегти палітру” викликає savePalette, яка надсилає дані палітри (ідентифікатор, кольори, тип гармонії, базовий колір) до code.js для збереження в figma.clientStorage. Кнопка “Видалити” викликає deletePalette, яка передає ідентифікатор палітри для видалення з локального сховища.

Для забезпечення стабільної роботи алгоритму вжито таких заходів:

- валідація HEX-коду: Перевірка формату запобігає помилкам;
- обмеження ітерацій: У adjustForContrast максимум 10 ітерацій для уникнення зависань;
- оптимізація обчислень: Перетворення HSL ↔ HEX виконуються з мінімальними операціями;
- обробка помилок: Некоректні дані викликають сповіщення через figma.notify.

Алгоритм роботи плагіна забезпечує ефективний підбір гармонійних кольорів, інтегруючись із Figma для зручного використання дизайнерами. Процес включає введення базового кольору, його конвертацію в HSL, генерацію палітри з адаптивним коригуванням контрасту та відображення результатів. Алгоритм “Адаптивна контрастна гармонія” є прикладом складного підходу, який враховує колірну гармонію та стандарти доступності. Описаний алгоритм є основою для подальшого тестування та вдосконалення плагіна.

2.3 Розробка інтерфейсу програми

Інтерфейс користувача (UI) плагіна для Figma, призначеного для автоматичного підбору гармонійних кольорових палітр, відіграє ключову роль у забезпеченні зручності та ефективності роботи дизайнерів. Основною метою розробки інтерфейсу є створення простого, інтуїтивно зрозумілого та мінімалістичного дизайну, який дозволяє користувачам швидко вводити параметри, генерувати палітри, застосовувати їх до об’єктів у Figma, зберігати

та видаляти збережені палітри без зайвих кроків. У цьому розділі детально описано процес розробки інтерфейсу, включаючи принципи дизайну, структуру, елементи управління, стилізацію та оптимізацію для користувацького досвіду (UX). Інтерфейс реалізовано в файлі ui.html з використанням HTML, CSS і JavaScript, що забезпечує його інтеграцію з Figma Plugin API.

Розробка інтерфейсу базується на таких принципах:

- простота: інтерфейс має мінімальну кількість елементів, щоб уникнути перевантаження користувача інформацією та спростити взаємодію;
- інтуїтивність: елементи управління розташовані логічно, а їх функціональність зрозуміла навіть для користувачів-початківців;
- консистентність: стилізація відповідає сучасним дизайн-системам і стандартам Figma, щоб інтерфейс виглядав як природна частина екосистеми редактора;
- ефективність: усі дії (введення кольору, вибір гармонії, генерація палітри) виконуються швидко, з мінімальною кількістю кліків;
- доступність: інтерфейс враховує принципи доступності, включаючи достатній контраст тексту, чіткі підписи та підтримку клавіатурної навігації;

Ці принципи забезпечують, що інтерфейс є зручним для дизайнерів із різним рівнем досвіду, від новачків до професіоналів, і відповідає вимогам швидкого робочого процесу в Figma.

Інтерфейс плагіна поділено на кілька функціональних блоків, які відображаються в компактному вікні розміром 400×600 пікселів, заданому в code.js через figma.showUI. Структура включає такі основні секції:

- секція введення базового кольору: Містить поле для HEX-коду та колірний пікер;
- секція вибору типу гармонії: Випадаючий список для вибору алгоритму підбору кольорів;

- секція управління: Кнопки для генерації палітри, застосування до об’єктів і збереження;
- секція відображення палітри: П’ять кольорових прямокутників із HEX-кодами;
- секція збережених палітр: Список раніше збережених палітр із можливістю їх застосування або видалення.

Кожен блок розташовано вертикально в контейнері , що забезпечує чітку ієрархію та легке сприйняття.

Інтерфейс плагіна планується поділити на кілька функціональних блоків, які відображатимуться в компактному вікні розміром 400×600 пікселів, заданому в code.js через figma.showUI. Структура включатиме такі основні секції:

- секція введення базового кольору: Містить текстове поле для HEX-коду та колірний пікер для візуального вибору кольору;
- секція вибору типу гармонії: Випадаючий список для вибору одного з семи алгоритмів підбору кольорів;
- секція управління: Кнопки для генерації палітри, застосування до об’єктів у Figma та збереження палітри;
- секція відображення палітри: Область для показу п’яти кольорових прямокутників із HEX-кодами згенерованої палітри;
- секція збережених палітр: Список раніше збережених палітр із можливістю їх застосування або видалення.

Кожен блок розташовуватиметься вертикально в контейнері , що забезпечуватиме чітку ієрархію, легке сприйняття та послідовний потік взаємодії.

Нижче наведено приклад планованого HTML-коду інтерфейсу, який буде реалізовано в ui.html для створення структури інтерфейсу:

```
<body>
  <div class="container">
    <div class="input-group">
      <label for="baseColor">Базовий колір (HEX)</label>
```

```
          <div class="color-input-group">
            <input type="text" id="baseColor"
placeholder="#RRGGBB" value="#3498db">
            <input type="color" id="colorPicker"
value="#3498db">
          </div>
        </div>
        <div class="input-group">
          <label for="harmonyType">Тип гармонії</label>
          <select id="harmonyType">
            <option
value="complementary">Комплементарна</option>
            <option value="analogous">Аналогова</option>
            <option value="triadic">Тріадна</option>
            <option value="split-complementary">Розділена
комплементарна</option>
            <option
value="monochromatic">Монохроматична</option>
            <option value="soft-harmony">М’яка гармонія</option>
            <option value="adaptive-contrast">Адаптивна
контрастна</option>
          </select>
        </div>
        <button onclick="generatePalette()">Згенерувати
палітру</button>
        <button onclick="applyToSelection()">Застосувати до
виділення</button>
        <button onclick="savePalette()">Зберегти
палітру</button>
        <div id="palette" class="palette"></div>
        <div class="saved-palettes">
          <h3>Збережені палітри</h3>
          <div id="savedPalettes"></div>
        </div>
      </div>
    </body>
```

Стилізація інтерфейсу планується реалізувати через CSS у файлі ui.html, щоб забезпечити сучасний вигляд і відповідність стандартам Figma. Основний контейнер .container матиме максимальну ширину 360 пікселів із відступами для центрування вмісту [20]. Кожен елемент управління, як-от текстове поле #baseColor, пікер #colorPicker, випадаючий список #harmonyType і кнопки, матиме закруглені кути (border-radius: 4px), легку тінь (box-shadow) і чіткий контраст фону та тексту для відповідності стандартам доступності WCAG (співвідношення контрасту ≥ 4.5:1). Кольорові прямокутники в секції #palette відображатимуться як блоки розміром 60×60 пікселів із HEX-кодами, вирівняними по центру, використовуючи шрифт із високою читабельністю

(наприклад, Inter або Roboto, розмір 12px). Секція збережених палітр #savedPalettes матиме вигляд списку, де кожна палітра представлена горизонтальним рядом із п’ятьма маленькими кольоровими прямокутниками (20×20 пікселів) і кнопкою “Видалити” із піктограмою кошика. Кнопки управління матимуть однаковий стиль із hover-ефектом (зміна кольору фону) і active-ефектом (легке затемнення) для інтерактивності. CSS також включатиме медіа-запити для адаптивності, щоб інтерфейс коректно відображався у вікнах різного розміру, якщо Figma дозволить змінювати розміри вікна плагіна.

Адаптивність інтерфейсу враховуватиме обмеження Figma Plugin API, де розмір вікна плагіна фіксується через figma.showUI. Однак, якщо API дозволить динамічне масштабування, CSS-стилі забезпечуватимуть коректне відображення елементів у вікнах від 300 до 600 пікселів завширшки, використовуючи відносні одиниці (%, vw, rem) для розмірів і відступів. Для локалізації планується додати підтримку кількох мов (наприклад, англійської та української) через змінні для тексту підпесів і кнопок, які завантажуватимуться залежно від налаштувань користувача в Figma.

Розробка інтерфейсу плагіна для Figma спрямовуватиметься на створення зручного, швидкого та візуально привабливого інструменту, який гармонійно інтегруватиметься з екосистемою Figma. Завдяки простій структурі, інтуїтивним елементам управління, сучасній стилізації та оптимізації для UX, інтерфейс дозволить дизайнерам ефективно працювати з кольоровими палітрами, зменшуючи час на рутинні завдання. Реалізація в ui.html із використанням HTML, CSS і JavaScript забезпечуватиме гнучкість і легкість підтримки, а відповідність принципам доступності та консистентності зробить плагін доступним для широкого кола користувачів.

3 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз обраного середовища програмування

Розробка плагіна для Figma, призначеного для автоматичного підбору гармонійних кольорових палітр, вимагала вибору оптимального середовища програмування, мов і інструментів, щоб забезпечити стабільну роботу, сумісність із платформою Figma та зручність для розробників і користувачів. Для реалізації плагіна обрано Visual Studio Code (VS Code) як основне середовище розробки, JavaScript як головну мову програмування, HTML і CSS для створення інтерфейсу, а також TypeScript для підвищення надійності коду. Жодні додаткові фреймворки, крім вбудованих можливостей Figma Plugin API, не використовувалися, що відповідає обмеженням платформи та забезпечує ефективність. У цьому розділі проаналізовано причини вибору цих технологій, їхні переваги та вплив на стабільність роботи плагіна в середовищі Figma.

Visual Studio Code був обраний як основне середовище розробки завдяки своїй універсальності, підтримці вебтехнологій і потужним інструментам для роботи з JavaScript і TypeScript. VS Code забезпечує автодоповнення, перевірку синтаксису та інтеграцію з Figma Plugin API через розширення, такі як “Figma Plugin DS” або “ESLint”, що значно спрощує написання коду. Вбудований термінал дозволяє виконувати команди npm для компіляції TypeScript і запуску плагіна, а підтримка Git інтегрує контроль версій. Легкість налаштування (tsconfig.json, package.json) і швидкодія роблять VS Code ідеальним для створення плагінів Figma, забезпечуючи зручність для розробників і стабільність робочого процесу.

JavaScript обрано як основну мову програмування через її повну сумісність із Figma Plugin API, яке працює в браузерному середовищі Node.js. Ця мова дозволяє реалізовувати логіку генерації палітр, обробку подій і

взаємодію з редактором Figma, наприклад, для застосування кольорів до об’єктів чи збереження даних. JavaScript забезпечує швидку обробку клієнтських операцій, таких як перетворення між HEX, RGB і HSL, що критично для миттєвої генерації палітр. Його універсальність і широка документація спростили розробку алгоритмів, зокрема складної “Адаптивної контрастної гармонії”, а асинхронні можливості (наприклад, async/await) гарантують стабільну роботу з локальним сховищем Figma.

HTML і CSS обрано для створення інтерфейсу плагіна, оскільки вони є стандартними технологіями для відображення вебінтерфейсів у Figma, де UI плагіна функціонує як вбудована вебсторінка. HTML забезпечує структуру для елементів управління, таких як поля введення, випадаючі списки та кнопки, а CSS відповідає за стилізацію, створюючи сучасний і компактний вигляд із системними шрифтами та консистентною кольоровою схемою. Ці технології дозволяють створювати легкий інтерфейс без залежностей від зовнішніх бібліотек, що зменшує час завантаження плагіна та забезпечує швидке оновлення елементів, наприклад, при відображенні згенерованих палітр.

TypeScript використовується для типізації коду, що підвищує його надійність і зменшує ймовірність помилок. Завдяки офіційним типам Figma Plugin API, TypeScript забезпечує чітке визначення структур даних, таких як об’єкти полотна чи властивості заповнення, що полегшує розробку та відлагодження. Автодоповнення та перевірка типів у VS Code допомагають уникати помилок, наприклад, при роботі з методами API. TypeScript компілюється в чистий JavaScript, що гарантує сумісність із Figma, зберігаючи при цьому високу якість коду та стабільність виконання [19].

Обране середовище програмування сприяє стабільній роботі плагіна завдяки кільком аспектам. JavaScript і TypeScript забезпечують повну сумісність із Figma Plugin API, дозволяючи ефективно взаємодіяти з редактором. HTML і CSS створюють легкий і швидкий інтерфейс, що важливо для миттєвого відображення палітр. VS Code оптимізує процес розробки,

надаючи інструменти для відлагодження та компіляції. Асинхронні операції в JavaScript і перевірка типів у TypeScript мінімізують помилки, наприклад, при збереженні даних, а відмова від фреймворків зменшує накладні витрати. Плагін працює однаково стабільно десктопній версіях Figma, що підтверджує правильність вибору технологій.

Вибір Visual Studio Code, JavaScript, HTML, CSS і TypeScript для розробки плагіна був обґрунтованим і відповідає вимогам Figma Plugin API. VS Code забезпечує зручне середовище для кодування, JavaScript і TypeScript гарантують сумісність і надійність, а HTML з CSS створюють ефективний інтерфейс. Відмова від фреймворків оптимізує продуктивність і спрощує підтримку. Ці технології разом забезпечують стабільну роботу плагіна, швидке виконання функцій і зручність для розробки, роблячи його цінним інструментом для дизайнерів у Figma.

3.2 Розробка бази даних

Розробка бази даних для плагіна Figma, призначеного для автоматичного підбору гармонійних кольорових палітр, є важливим аспектом забезпечення збереження та повторного використання даних про створені палітри. Оскільки плагін працює в середовищі Figma, яке має обмеження щодо локального зберігання даних, база даних реалізована через локальне сховище Figma (figma.clientStorage), що дозволяє зберігати дані асинхронно та забезпечує їх доступність у різних сеансах роботи. У цьому розділі детально описано структуру бази даних, принципи збереження даних про кольорові палітри, механізми їх використання в подальших проєктах, а також аспекти безпеки, оптимізації та масштабованості.

База даних плагіна має забезпечувати:

- збереження кольорових палітр: Кожна палітра, створена користувачем, повинна зберігатися разом із метаданими (тип гармонії, базовий колір, ідентифікатор);

- доступність даних: Збережені палітри мають бути доступними для відображення, застосування або видалення в інтерфейсі плагіна;
- повторне використання: Дані палітр повинні бути доступними в різних проєктах Figma, якщо користувач працює в одному екземплярі редактора;
- простота реалізації: Оскільки Figma не підтримує складні бази даних, рішення має бути мінімалістичним і використовувати вбудовані можливості API;
- безпека та стабільність: Збереження та обробка даних не повинні викликати помилок або втрати інформації.

Збереження даних реалізовано через асинхронні методи figma.clientStorage у файлі code.js. Процес збереження палітри включає такі кроки:

- ініціація збереження:
  - користувач натискає кнопку “Зберегти палітру” в інтерфейсі (ui.html);
  - функція savePalette створює об’єкт палітри:

```
const paletteData = {
   id: Date.now(),
   colors: currentPalette,
   harmonyType: document.getElementById('harmonyType').value,
   baseColor: document.getElementById('baseColor').value
};
```

  - дані надсилаються до code.js через parent.postMessage({ type: 'savePalette', palette: paletteData }).
- обробка в code.js:
  - отримується повідомлення через figma.ui.onmessage;
  - зчитується поточний масив savedPalettes за допомогою figma.clientStorage.getAsync('savedPalettes');
  - якщо масив не існує, ініціалізується порожній: savedPalettes = [];
  - нова палітра додається до масиву: savedPalettes.push(paletteData);

- оновлений масив зберігається через figma.clientStorage.setAsync('savedPalettes', savedPalettes);
- користувачу відображається сповіщення (figma.notify('Палітру збережено')), і оновлений список палітр надсилається до UI через postMessage.

- оновлення інтерфейсу:
  - UI отримує повідомлення з типом palettesLoaded і викликає renderSavedPalettes для відображення оновленого списку палітр.

Збережені палітри, створені плагіном для Figma, можна використовувати в різних проєктах, якщо користувач працює в межах одного екземпляра редактора Figma, будь то браузерна чи десктопна версія. Це дозволяє дизайнерам легко повертатися до раніше створених кольорових схем і застосовувати їх у нових або поточних проєктах. Для забезпечення зручного доступу до палітр і їхньої повторної інтеграції в робочий процес плагін реалізує кілька ключових механізмів, які охоплюють завантаження, відновлення, застосування та видалення палітр. Ці механізми побудовані на асинхронній взаємодії між інтерфейсом користувача (UI) і логікою інтеграції з Figma, що гарантує стабільність і швидкість роботи.
Коли користувач запускає плагін, автоматично активується процес завантаження збережених палітр. Для цього викликається спеціальна функція, яка надсилає запит до основної логіки плагіна з повідомленням про необхідність отримання даних. У відповідь основна логіка звертається до локального сховища Figma, використовуючи асинхронний метод для зчитування масиву збережених палітр. Після успішного отримання даних вони передаються назад до інтерфейсу через механізм асинхронного обміну повідомленнями. У результаті інтерфейс відображає список палітр у спеціально відведеній області, де кожна палітра представлена у вигляді картки з кольоровими прев’ю та інформацією про тип гармонії.
Щоб повторно використати збережену палітру, користувач може просто клацнути на її картку в інтерфейсі. Це дія викликає функцію, яка ідентифікує обрану палітру за її унікальним ідентифікатором і відновлює всі пов’язані з

нею дані, включаючи набір кольорів, тип гармонії та базовий колір. У процесі відновлення оновлюється внутрішній стан плагіна: поточна палітра замінюється вибраною, поле введення базового кольору та колірний пікер синхронізуються з базовим кольором палітри, а список вибору гармонії встановлюється відповідно до типу палітри. Після цього викликається функція генерації, яка відображає палітру в інтерфейсі, дозволяючи користувачу переглянути кольори та продовжити роботу.

Збережені палітри можна безпосередньо застосовувати до елементів дизайну в Figma, що робить їх цінними для роботи над різними проєктами. Користувач вибирає об’єкти на полотні Figma (наприклад, прямокутники чи текст) і натискає кнопку “Застосувати до виділення”. Плагін передає кольори палітри до основної логіки, яка обробляє виділені об’єкти. Кольори застосовуються до властивостей заповнення об’єктів, забезпечуючи швидке оновлення дизайну. Цей процес дозволяє дизайнерам економити час, повторно використовуючи палітри без необхідності їхньої повторної генерації.

Якщо палітра більше не потрібна, користувач може видалити її, натиснувши кнопку “Видалити” поруч із карткою палітри в інтерфейсі. Ця дія ініціює запит до основної логіки плагіна, вказуючи ідентифікатор палітри, яку потрібно видалити. Логіка плагіна фільтрує масив збережених палітр, видаляючи запис із відповідним ідентифікатором, і оновлює локальне сховище Figma за допомогою асинхронного методу. Після цього оновлений список палітр надсилається назад до інтерфейсу, який миттєво відображає зміни, видаляючи картку палітри з UI. Цей механізм забезпечує зручне керування збереженими даними та підтримує порядок у списку палітр.

3.3 Програмна реалізація основних функцій

Плагін для Figma, призначений для автоматичного підбору гармонійних кольорових палітр, є програмним модулем, який інтегрується з графічним редактором через Figma Plugin API. Основні функції плагіна охоплюють підбір кольорів на основі заданого тону з використанням семи типів колірних

гармоній, інтеграцію з Figma для застосування палітр до об’єктів, а також тестування результатів для забезпечення якості та зручності використання. У цьому розділі детально описано програмну реалізацію цих функцій, включаючи алгоритми генерації палітр, фрагменти коду, взаємодію компонентів і методи тестування. Реалізація виконана з використанням JavaScript для логіки, HTML/CSS для інтерфейсу та TypeScript для типізації, що забезпечує стабільність і сумісність із Figma. Після кожного фрагмента коду надано пояснення його роботи.

Підбір кольорів є центральною функцією плагіна, яка дозволяє генерувати гармонійні палітри на основі введеного користувачем базового кольору. Ця функція реалізована в інтерфейсній частині плагіна за допомогою JavaScript і підтримує обробку кольорів у моделях HEX, RGB і HSL, а також сім типів гармоній: комплементарна, аналогова, тріадна, розділена комплементарна, монохроматична, м’яка гармонія та адаптивна контрастна.

Користувач вводить базовий колір через текстове поле або колірний пікер. Синхронізація між цими елементами забезпечується через обробники подій:

```
const baseColorInput = document.getElementById('baseColor');
const colorPicker = document.getElementById('colorPicker');
baseColorInput.addEventListener('input', () => {
  if (/^#[0-9A-Fa-f]{6}$/.test(baseColorInput.value)) {
    colorPicker.value = baseColorInput.value;
  }
});
colorPicker.addEventListener('input', () => {
  baseColorInput.value = colorPicker.value;
});
```

Цей код забезпечує синхронізацію між текстовим полем (baseColor) і колірним пікером (colorPicker). При введенні HEX-коду в текстове поле перевіряється його валідність за допомогою регулярного виразу. Якщо код коректний, він застосовується до пікера. При зміні кольору в пікері оновлюється текстове поле. Це гарантує, що обидва елементи завжди відображають один і той же колір.

Валідація HEX-коду виконується в функції generatePalette за допомогою регулярного виразу /^#[0-9A-Fa-f]{6}$/. У разі некоректного введення

відправляється сповіщення через parent.postMessage({ type: 'notify', message: 'Введіть валідний HEX-код' }).

Для роботи з кольорами базовий HEX-код конвертується в HSL через функцію hexToHSL:

```
function hexToHSL(hex) {
  let r = parseInt(hex.slice(1, 3), 16) / 255;
  let g = parseInt(hex.slice(3, 5), 16) / 255;
  let b = parseInt(hex.slice(5, 7), 16) / 255;
  let max = Math.max(r, g, b), min = Math.min(r, g, b);
  let h, s, l = (max + min) / 2;
  if (max === min) {
    h = s = 0;
  } else {
    let d = max - min;
    s = l > 0.5 ? d / (2 - max - min) : d / (max + min);
    switch (max) {
      case r: h = (g - b) / d + (g < b ? 6 : 0); break;
      case g: h = (b - r) / d + 2; break;
      case b: h = (r - g) / d + 4; break;
    }
    h /= 6;
  }
  return { h: h * 360, s: s * 100, l: l * 100 };
}
```

Функція розбирає HEX-код на компоненти RGB, нормалізує їх до діапазону 0–1, а потім обчислює HSL. Відтінок (*H*) визначається залежно від максимального компонента RGB, насиченість (*S*) і світлість (*L*) — на основі різниці між максимумом і мінімумом. Результат повертається як об’єкт { *h, s, l* } у градусах і відсотках. Зворотна конвертація (HSL у HEX) виконується аналогічно через hslToHex.

Генерація палітр виконується функцією generateHarmony, яка підтримує сім типів гармоній, кожен із яких створює палітру з різною кількістю кольорів і принципами. Нижче наведено опис, код і пояснення для кожного типу.

Комплементарна гармонія генерує два кольори: базовий і протилежний на колірному колі (зсув відтінку на 180°), зберігаючи насиченість (*S*) і світлість (*L*). Використовується для створення контрастних акцентів, наприклад, для кнопок або текстових елементів.

```
case 'complementary':
  colors.push({ h: (h + 180) % 360, s, l });
```

```
  break;
```

Код додає до відтінку базового кольору 180°, створюючи комплементарний колір. Модульна операція % 360 забезпечує, що відтінок залишається в межах 0–360°. Насиченість і світлість зберігаються, щоб зберегти схожість із базовим кольором, але з максимальним контрастом відтінків.

Аналогова гармонія генерує три кольори, зсуваючи відтінок базового кольору на +30° і -30°, зберігаючи *S* і *L*. Ця гармонія створює м'які, злагоджені палітри, ідеальні для фонів і градієнтів.

```
case 'analogous':
  colors.push({ h: (h - 30 + 360) % 360, s, l });
  colors.push({ h: (h + 30) % 360, s, l });
  break;
```

Код створює два додаткові кольори, зсуваючи відтінок на -30° і +30°. Додавання 360 перед модульною операцією для негативного зсуву запобігає отриманню від'ємних значень. Збереження *S* і *L* забезпечує схожість кольорів, що сприяє гармонійному вигляду.

Тріадна гармонія генерує три кольори, рівномірно розподілені на колірному колі (зсуви на +120° і +240°). Ця гармонія створює збалансовані палітри з вираженим контрастом, підходящі для ілюстрацій і діаграм.

```
case 'triadic':
  colors.push({ h: (h + 120) % 360, s, l });
  colors.push({ h: (h + 240) % 360, s, l });
  break;
```

Код додає до відтінку 120° і 240°, створюючи два кольори, які разом із базовим формують тріаду. Модульна операція утримує відтінки в межах кола. Збереження *S* і *L* забезпечує консистентність, а рівномірний розподіл відтінків додає різноманітність.

Розділена комплементарна гармонія генерує три кольори: базовий і два, розташовані поруч із комплементарним (зсуви на +150° і +210°). Забезпечує м'який контраст для збалансованих дизайнів.

```
case 'split-complementary':
  colors.push({ h: (h + 150) % 360, s, l });
  colors.push({ h: (h + 210) % 360, s, l });
  break;
```

Замість прямого комплементарного кольору (H + 180°) код використовує зсуви на 150° і 210°, створюючи два кольори, які м’якше контрастують із базовим. Збереження *S* і *L* підтримує гармонію, а зсуви забезпечують менш агресивний контраст порівняно з комплементарною гармонією.

Монохроматична гармонія генерує п’ять кольорів, варіюючи насиченість (±20%) і світлість (±20%) базового кольору, зберігаючи відтінок. Підходить для мінімалістичних інтерфейсів і брендингу.

```
case 'monochromatic':
  colors.push({ h, s: Math.min(100, s + 20), l });
  colors.push({ h, s: Math.max(0, s - 20), l });
  colors.push({ h, s, l: Math.min(100, l + 20) });
  colors.push({ h, s, l: Math.max(0, l - 20) });
  break;
```

Код створює чотири додаткові кольори, змінюючи *S* і *L* базового кольору. Функції Math.min і Math.max обмежують значення в межах 0–100%, щоб уникнути некоректних значень. Збереження відтінку забезпечує єдність палітри, а варіювання *S* і *L* додає різноманітність відтінків.

М’яка гармонія генерує три кольори, комбінуючи аналогові відтінки (±30°) з адаптивними змінами насиченості (±10/15%) і світлості (±10/15%). Створює ніжні палітри для елегантних дизайнів.

```
case 'soft-harmony':
  colors.push({ h: (h + 30) % 360, s: Math.max(0, s - 15), l:
Math.min(100, l + 10) });
  colors.push({ h: (h - 30 + 360) % 360, s: Math.min(100, s +
10), l: Math.max(0, l - 15) });
  break;
```

Код зсуває відтінок на ±30°, зменшуючи насиченість і збільшуючи світлість для першого кольору, і навпаки для другого. Обмеження через Math.min і Math.max забезпечують коректні значення. Це створює м’яку, ненав’язливу палітру з легкими контрастами.

Адаптивна контрастна гармонія генерує п’ять кольорів: аналоговий м’який (*H* + 40°, *S* - 20, *L* + 15), контрастний (*H* + 180°, контраст ≥ 3:1), аналоговий контрастний (*H* - 40°, *S* + 15, ш - 10) і нейтральний акцент (*H, S* = 20, *L* = 80/20). Для доступних інтерфейсів.

```
case 'adaptive-contrast':
```

```
  colors.push({ h: (h + 40) % 360, s: s > 40 ? s - 20 : s, l: l
< 70 ? l + 15 : l });
  colors.push({ h: (h + 180) % 360, s: s < 60 ? s + 10 : s, l: l
< 30 ? l + 20 : l > 50 ? l - 20 : l });
  colors.push({ h: (h - 40 + 360) % 360, s: s < 80 ? s + 15 : s,
l: l > 20 ? l - 10 : l });
  colors.push({ h: h, s: 20, l: l < 50 ? 80 : 20 });
  break;
```

Код створює чотири додаткові кольори з адаптивними змінами. Умовні опера

Згенерована палітра відображається як п’ять прямокутників:

```
const paletteDiv = document.getElementById('palette');
paletteDiv.innerHTML = currentPalette.map(color => `
  <div class="color-swatch" style="background: ${color}">
    <span>${color}</span>
  </div>
`).join('');
```

Код динамічно генерує HTML для п’яти прямокутників, кожен із фоновим кольором із палітри та текстовим HEX-кодом. Використання map і join забезпечує компактне створення розмітки, а оновлення innerHTML миттєво відображає палітру в інтерфейсі.

Інтеграція з Figma реалізована через Figma Plugin API, що дозволяє застосовувати палітри, зберігати дані та обробляти взаємодію.

Кнопка “Застосувати до виділення” передає палітру до основної логіки:

```
if (msg.type === 'applyPalette') {
  const palette = msg.palette;
  const selection = figma.currentPage.selection;
  if (selection.length === 0) {
    figma.notify('Виберіть хоча б один об’єкт');
    return;
  }
  selection.forEach((node, index) => {
    if ('fills' in node) {
      const color = palette[index % palette.length];
      const r = parseInt(color.slice(1, 3), 16) / 255;
      const g = parseInt(color.slice(3, 5), 16) / 255;
      const b = parseInt(color.slice(5, 7), 16) / 255;
      node.fills = [{ type: 'SOLID', color: { r, g, b } }];
    }
  });
  figma.notify('Палітру застосовано');
}
```

Код перевіряє наявність виділених об’єктів. Для кожного об’єкта з властивістю fills HEX-код із палітри конвертується в RGB (0–1) і

застосовується як заповнення. Використання index % palette.length забезпечує циклічне застосування кольорів, якщо об’єктів більше, ніж кольорів.

Збереження виконується асинхронно:

```
if (msg.type === 'savePalette') {
  const paletteData = msg.palette;

figma.clientStorage.getAsync('savedPalettes').then(savedPalettes
=> {
    savedPalettes = savedPalettes || [];
    savedPalettes.push(paletteData);
    figma.clientStorage.setAsync('savedPalettes',
savedPalettes).then(() => {
      figma.notify('Палітру збережено');
      figma.ui.postMessage({ type: 'palettesLoaded', palettes:
savedPalettes });
    });
  });
}
```

Код отримує дані палітри, додає їх до масиву savedPalettes, який зберігається в figma.clientStorage. Після збереження надсилається оновлений список до UI і відображається сповіщення. Використання асинхронних методів (getAsync, setAsync) запобігає блокуванню.

Видалення палітри:

```
if (msg.type === 'deletePalette') {
  const paletteId = msg.id;

figma.clientStorage.getAsync('savedPalettes').then(savedPalettes
=> {
    const updatedPalettes = savedPalettes.filter(p => p.id !==
paletteId);
    figma.clientStorage.setAsync('savedPalettes',
updatedPalettes).then(() => {
      figma.notify('Палітру видалено');
      figma.ui.postMessage({ type: 'palettesLoaded', palettes:
updatedPalettes });
    });
  });
}
```

Код фільтрує масив savedPalettes, видаляючи палітру за ідентифікатором, оновлює сховище та надсилає оновлений список до UI. Сповіщення підтверджує успішне видалення.

Розроблений плагін зображений на рисунку 3.1.

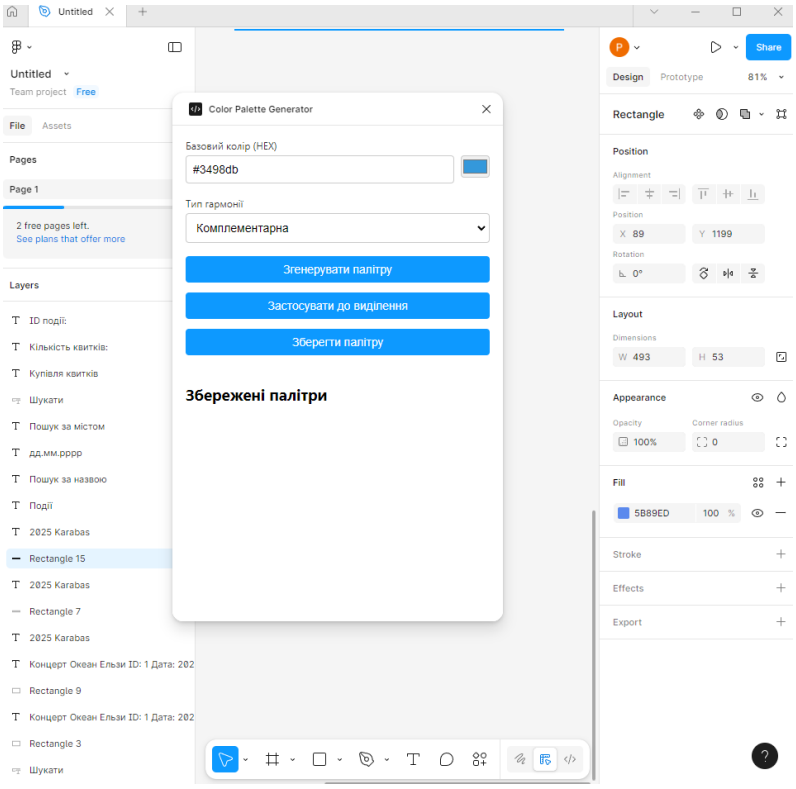

Рисунок 3.1 — Розроблений плагін

На рисунку 3.2 продемонстровано вибір типу генерації кольору. При натисканні типу гармонії випливає вікно яке відображає типи генерації, зображено на рисунку 3.2.

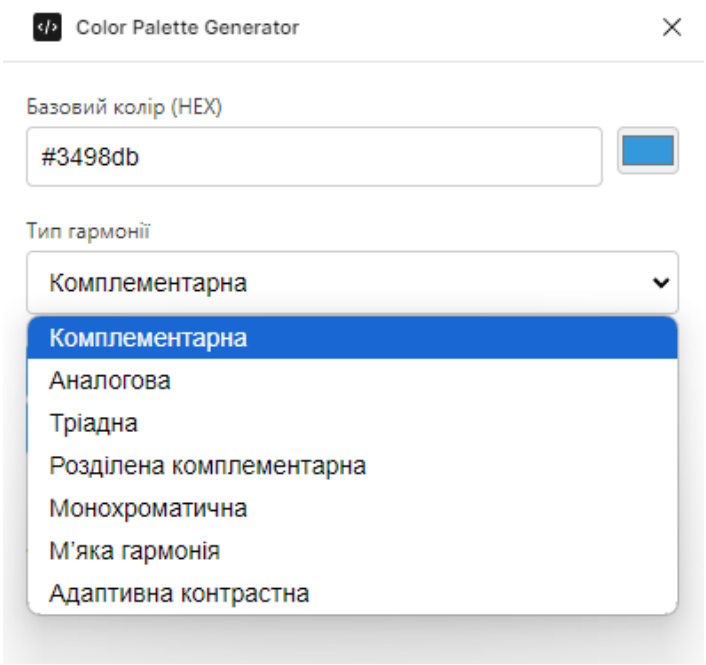

Рисунок 3.2 — Типи генерації

На русинку 3.3. показано збереження палітри. Спочатку користувач повинен згенерувати палітру, а після цього натиснути кнопку зберегти палітру. Для видалення потрібно натиснути на збережену палітру та натиснути видалити.

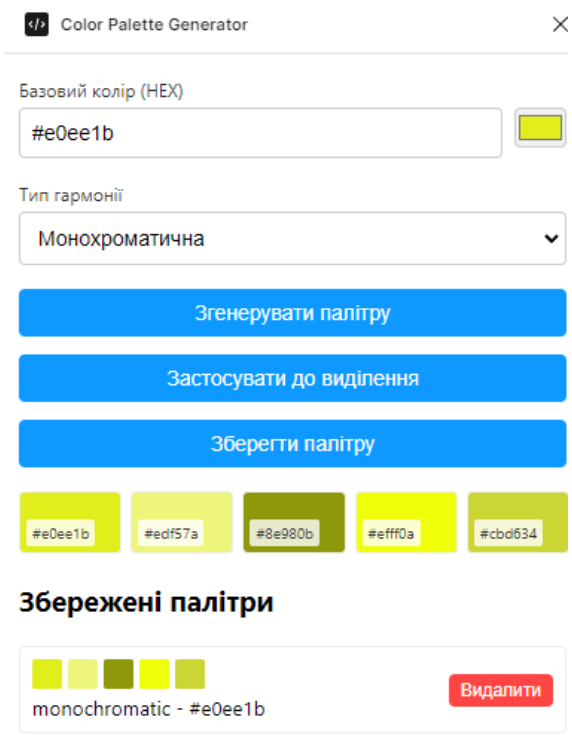

Рисунок 3.3 — Збереження палітри

Для того, щоб пофарбувати елементи у відповідні кольори вибираємо елементи, вибираємо палітру та натискаємо застосувати до виділення. Результат показаний на рисунку 3.4.

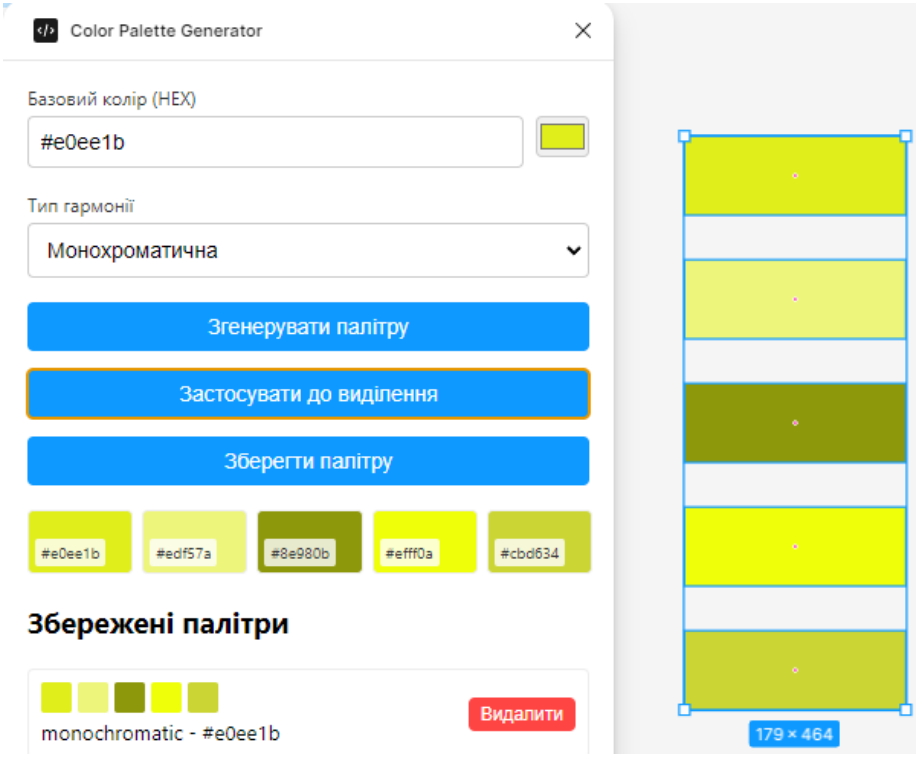

Рисунок 3.4 — Застосування палітри до виділених об’єктів

Тестування показала ефективність програми та її працездатність. Всі функції працюють та застосовуються до об’єктів.

3.4 Методика роботи користувача

Плагін для Figma, призначений для автоматичного підбору гармонійних кольорових палітр, створений з акцентом на простоту використання, щоб дизайнери могли швидко налаштувати інструмент, підібрати кольори та застосувати їх у своїх проєктах. У цьому розділі детально описано, як користувач може налаштувати плагін, які дії потрібно виконати для підбору кольорів, і як використовувати отримані результати. Плагін додається вручну через редактор Figma, що вимагає певних дій для його ініціалізації. Методика охоплює всі етапи роботи: від додавання та запуску плагіна до збереження палітр і їхнього застосування в дизайні, забезпечуючи інтуїтивний і ефективний робочий процес.

Для додавання плагіну користувач повинен в програмі зліва натиснути на значок Figma.

Далі вибрати пункт Plugins, Development та import plugins from manifest. І далі в папці з плагіном вибрати manifest.json.

Процес підбору кольорів починається з введення базового кольору, який стане основою для генерації палітри. Користувач має виконати наступні кроки.

Перший крок введення базового кольору.У текстовому полі з позначкою “Базовий колір” користувач вводить HEX-код кольору, наприклад, #3498db (синій відтінок). Код має відповідати формату #RRGGBB, де *R*, *G*, *B* — це шістнадцяткові значення для червоного, зеленого та синього. Якщо користувач не знає точного коду, він може скористатися колірним пікером, натиснувши на кольорову область поруч із текстовим полем. Після вибору кольору в пікері текстове поле автоматично оновиться, відображаючи відповідний HEX-код. У разі введення некоректного формату (наприклад, #ZZZ або 123), плагін видасть сповіщення “Введіть валідний HEX-код”, і генерація не відбудеться.

Другий крок вибір типу гармонії. У випадаючому списку, позначеному як “Тип гармонії”, користувач обирає один із семи доступних варіантів:

- комплементарна гармонія: для створення двох контрастних кольорів (наприклад, синій і помаранчевий);
- аналогова гармонія: для трьох схожих кольорів, ідеальних для фонів (наприклад, відтінки синього);
- тріадна гармонія: для трьох рівномірно розподілених кольорів, що додають різноманітність (наприклад, синій, червоний, зелений);
- розділена комплементарна гармонія: для трьох кольорів із м’яким контрастом (наприклад, синій і два відтінки помаранчевого);
- монохроматична гармонія: для п’яти відтінків одного кольору (наприклад, різні відтінки синього);
- м’яка гармонія: для трьох ніжних кольорів із легкими контрастами (наприклад, м’які відтінки синього та блакитного);

- адаптивна контрастна гармонія: для п’яти кольорів із адаптивним контрастом ≥ 3:1, що відповідає стандартам доступності (наприклад, синій, контрастний помаранчевий, аналогові відтінки).

Користувач обирає тип гармонії залежно від потреб проєкту. Наприклад, для доступного інтерфейсу варто обрати “Адаптивну контрастну гармонію”, а для спокійного фону — “Аналогову гармонію”.

Третій крок генерація палітри. Після введення базового кольору та вибору типу гармонії користувач натискає кнопку “Згенерувати” (або генерація відбувається автоматично при зміні кольору чи типу гармонії, залежно від налаштувань). Плагін обробляє введені дані, конвертує HEX-код у HSL, застосовує обраний алгоритм гармонії та генерує палітру. Результат відображається у вигляді п’яти прямокутників (або двох/трьох, залежно від типу гармонії), кожен із яких показує HEX-код кольору та його візуальне представлення. Наприклад, для базового кольору #3498db і “Комплементарної гармонії” користувач побачить синій (#3498db) і помаранчевий (#db6e34).

Якщо результат не відповідає очікуванням, користувач може змінити базовий колір або тип гармонії та згенерувати нову палітру. Процес генерації займає менше 100 мс, що забезпечує миттєву реакцію інтерфейсу.

Після генерації палітри користувач може використати її в проєкті Figma кількома способами: застосувати до об’єктів, зберегти для майбутнього використання або видалити, якщо вона більше не потрібна.

Розглянемо застосування палітри до об’єктів. Щоб застосувати палітру до елементів дизайну, користувач спочатку виділяє об’єкти на полотні Figma (наприклад, прямокутники, текст або групи елементів). Далі він натискає кнопку “Застосувати до виділення” у вікні плагіна. Плагін передає кольори палітри до Figma API, і кожен виділений об’єкт отримує відповідний колір із палітри як заповнення (node.fills). Наприклад, якщо палітра містить п’ять кольорів, а виділених об’єктів десять, кольори застосовуються циклічно (перший об’єкт — перший колір, шостий об’єкт — перший колір тощо). Якщо об’єктів не виділено, плагін видасть сповіщення “Виберіть хоча б один

об’єкт”. Після успішного застосування з’явиться сповіщення “Палітру застосовано”. Це дозволяє швидко оновити дизайн, наприклад, змінити кольори кнопок або фону.

Розглянемо збереження палітри. Якщо палітра сподобалася і користувач хоче використати її в інших проєктах, він може зберегти її, натиснувши кнопку “Зберегти палітру”. Плагін додає палітру до локального сховища Figma (figma.clientStorage) із унікальним ідентифікатором, кольорами, типом гармонії та базовим кольором. Збережена палітра відображається в області “Збережені палітри” у вигляді картки з прев’ю кольорів і назвою типу гармонії. Наприклад, палітра з базовим кольором #3498db і типом “Адаптивна контрастна” буде показана як картка з п’ятьма кольорами. Збереження дозволяє користувачу повернутися до палітри в межах одного екземпляра Figma (браузер або десктоп).

Розглянемо відновлення збереженої палітри. Щоб використати збережену палітру, користувач натискає на її картку в області “Збережені палітри”. Плагін відновлює дані: оновлює поточну палітру, встановлює базовий колір у текстове поле та колірний пікер, а також змінює тип гармонії у випадаючому списку. Після цього палітра відображається в основній області для перегляду, і користувач може одразу застосувати її до об’єктів або модифікувати, змінивши базовий колір чи тип гармонії.

Розглянемо видалення палітри. Якщо палітра більше не потрібна, користувач може видалити її, натиснувши кнопку “Видалити” на картці в області “Збережені палітри”. Плагін видаляє палітру з локального сховища і оновлює список збережених палітр, видаючи сповіщення “Палітру видалено”. Це допомагає підтримувати порядок і видаляти непотрібні палітри, економлячи місце в сховищі.

Розглянемо експорт HEX-кодів. Користувач може скопіювати HEX-коди кольорів із палітри, просто натиснувши на HEX-код у прямокутнику палітри (наприклад, #3498db). Код автоматично копіюється в буфер обміну, що

дозволяє вставити його в інші інструменти, наприклад, у стилі Figma або зовнішні програми, такі як Adobe Photoshop.

Для максимальної ефективності користувачу варто дотримуватися кількох рекомендацій:

- перед генерацією палітри переконайтеся, що базовий колір відповідає загальній стилістиці проєкту. Наприклад, для корпоративного дизайну з основним синім кольором обирайте відтінки, близькі до брендових;
- використовуйте “Адаптивну контрастну гармонію” для проєктів, де важлива доступність, оскільки вона забезпечує контрастність ≥ 3:1, що відповідає стандартам WCAG;
- регулярно видаляйте непотрібні збережені палітри, щоб уникнути перевантаження списку;
- якщо палітра застосовується до великої кількості об’єктів, перевірте, чи правильно розподілилися кольори, і за потреби скорегуйте виділення;
- зберігайте палітри перед закриттям плагіна, оскільки незбережені палітри не зберігаються між сеансами.

Методика роботи користувача з плагіном є простою і зрозумілою, дозволяючи швидко налаштувати інструмент, підібрати гармонійні кольори та застосувати їх у проєктах Figma. Процес включає ручне додавання плагіна через редактор Figma, введення базового кольору, вибір типу гармонії, генерацію палітри, а також збереження, застосування та видалення результатів. Інтуїтивний інтерфейс і миттєва генерація палітр роблять плагін зручним для дизайнерів, а можливість збереження палітр забезпечує гнучкість у роботі з різними проєктами. Дотримання рекомендацій дозволяє оптимізувати робочий процес і досягати якісних результатів у дизайні.

ВИСНОВКИ

На основі ґрунтовного аналізу теорії кольору, викладеної у працях Йоганнеса Іттена, Джозефа Альберса та Ву Вонга, стандартів доступності WCAG, а також критичного огляду сучасних інструментів для підбору колірних палітр, таких як Adobe Color, Coolors, Paletton і Color Hunt, розроблено програмний модуль у вигляді плагіна для графічного редактора Figma. Розробка базується на можливостях Figma Plugin API та виконана з використанням середовища Visual Studio Code, мов програмування JavaScript і TypeScript, а також технологій HTML і CSS для створення інтерфейсу. Плагін реалізує сім алгоритмів генерації колірних палітр, зокрема унікальний алгоритм “Адаптивна контрастна гармонія”, який забезпечує контрастність ≥ 3:1 відповідно до стандартів доступності. Дані палітр зберігаються в локальному сховищі figma.clientStorage у структурованому вигляді як масив об’єктів із полями id, colors, harmonyType і baseColor, що забезпечує їхню доступність для повторного використання в різних проєктах.

Розроблений плагін дозволяє дизайнерам ефективно генерувати гармонійні колірні палітри на основі введеного базового кольору, застосовувати їх до об’єктів на полотні Figma, зберігати палітри для подальшого використання та видаляти непотрібні. Інтуїтивно зрозумілий інтерфейс із полем для введення HEX-коду, колірним пікером, випадаючим списком типів гармонії та кнопками управління забезпечує зручність роботи для користувачів із різним рівнем досвіду. Оптимізовані алгоритми гарантують швидкість генерації палітр (менше 100 мс), а типізація коду через TypeScript і асинхронний обмін даними між UI-компонентом і основним кодом забезпечують стабільність і надійність роботи плагіна.

Розробка сприяє підвищенню продуктивності дизайнерів, скороченню часу на створення колірних схем, забезпеченню відповідності стандартам інклюзивного дизайну та адаптації до культурних особливостей глобального ринку. Плагін має комерційний потенціал, зокрема через високий попит на

інструменти автоматизації в Figma, і може бути вдосконалений шляхом додавання функцій, таких як симуляція сприйняття кольорів для людей із дальтонізмом, інтеграція з іншими дизайнерськими платформами чи підтримка додаткових колірних моделей. Отримані результати підтверджують практичну цінність розробки для сфер дизайну інтерфейсів, веброзробки, брендингу та графічного дизайну, відкриваючи перспективи для подальших досліджень у сфері автоматизації дизайнерських процесів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Іттен Й. Мистецтво кольору / Йоганнес Іттен. – Київ: АртХаус, 2020. – 96 с.

2. Альберс Д. Взаємодія кольорів / Джозеф Альберс. – Харків: Видавництво, 2019. – 208 с.

3. Вонг В. Принципи дизайну кольору / Ву Вонг. – Нью-Йорк: Van Nostrand Reinhold, 2011. – 144 p.

4. Nielsen Norman Group. The Impact of Color on User Experience // Nielsen Norman Group, 2023. – [Електронний ресурс]. – Режим доступу: https://www.nngroup.com/articles/color-ux/.

5. Figma State of Design Report 2023 // Figma, 2023. – [Електронний ресурс]. – Режим доступу: https://www.figma.com/state-of-design-2023/.

6. Design Trends 2024 // Dribbble, 2024. – [Електронний ресурс]. – Режим доступу: https://dribbble.com/design-trends-2024.

7. Web Content Accessibility Guidelines (WCAG) 2.1 // W3C, 2018. – [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/WCAG21/.

8. Figma Plugin API Documentation // Figma, 2023. – [Електронний ресурс]. – Режим доступу: https://www.figma.com/plugin-docs/.

9. Adobe Color: User Guide // Adobe, 2023. – [Електронний ресурс]. – Режим доступу: https://color.adobe.com/guide.

10. Coolors: Features and Documentation // Coolors, 2023. – [Електронний ресурс]. – Режим доступу: https://coolors.co/docs.

11. Paletton: Color Theory and Tools // Paletton, 2023. – [Електронний ресурс]. – Режим доступу: http://paletton.com/info/theory.

12. Color Hunt: Community-Driven Palettes // Color Hunt, 2023. – [Електронний ресурс]. – Режим доступу: https://colorhunt.co/about.

13. Berners-Lee T. Designing with Web Standards / Tim Berners-Lee. – 4th ed. – New York: Peachpit Press, 2020. – 432 p.

14. Krause J. Color for Designers: Ninety-five Things You Need to Know When Choosing and Using Colors for Layouts and Illustrations / Jim Krause. – Boston: New Riders, 2014. – 240 p.

15. Lupton E. Design Is Storytelling / Ellen Lupton. – New York: Cooper Hewitt, 2017. – 160 p.

16. Meier B. J. Color Theory for Digital Displays / Barbara J. Meier // ACM Transactions on Graphics, 2015. – Vol. 34, No. 4. – P. 1–10.

17. Tufte E. R. Envisioning Information / Edward R. Tufte. – Cheshire: Graphics Press, 1990. – 126 p.

18. JavaScript: The Definitive Guide / David Flanagan. – 7th ed. – Sebastopol: O'Reilly Media, 2020. – 704 p.

19. TypeScript Documentation // TypeScript, 2023. – [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/.

20. CSS: The Missing Manual / David Sawyer McFarland. – 4th ed. – Sebastopol: O'Reilly Media, 2015. – 720 p.

Додаток А

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Color Palette Generator</title>
  <style>
    body {
      font-family: -apple-system, BlinkMacSystemFont, 'Segoe
UI', Roboto, Oxygen, Ubuntu, Cantarell, sans-serif;
      padding: 16px;
      margin: 0;
      background: #fff;
    }
    .container {
      max-width: 400px;
    }
    .input-group {
      margin-bottom: 16px;
    }
    label {
      display: block;
      margin-bottom: 4px;
      font-size: 12px;
      color: #333;
    }
    input, select {
      width: 100%;
      padding: 8px;
      border: 1px solid #ccc;
      border-radius: 4px;
      font-size: 14px;
    }
    .color-input-group {
      display: flex;
      gap: 8px;
    }
    .color-input-group input[type="color"] {
      width: 40px;
      padding: 0;
    }
    button {
      background: #0d99ff;
      color: white;
      padding: 8px 16px;
      border: none;
      border-radius: 4px;
      cursor: pointer;
      font-size: 14px;
      margin-right: 8px;
    }
    button:hover {
```

```
    background: #0077cc;
  }
  .delete-button {
    background: #ff4444;
    padding: 4px 8px;
    font-size: 12px;
  }
  .delete-button:hover {
    background: #cc0000;
  }
  .palette {
    display: grid;
    grid-template-columns: repeat(5, 1fr);
    gap: 8px;
    margin-top: 16px;
  }
  .color-swatch {
    width: 100%;
    height: 40px;
    border-radius: 4px;
    border: 1px solid #eee;
    position: relative;
  }
  .color-swatch span {
    position: absolute;
    bottom: 4px;
    left: 4px;
    font-size: 10px;
    color: #333;
    background: rgba(255, 255, 255, 0.8);
    padding: 2px 4px;
    border-radius: 2px;
  }
  .saved-palettes {
    margin-top: 16px;
  }
  .saved-palette {
    padding: 8px;
    border: 1px solid #eee;
    border-radius: 4px;
    margin-bottom: 8px;
    cursor: pointer;
    display: flex;
    justify-content: space-between;
    align-items: center;
  }
  .saved-palette:hover {
    background: #f5f5f5;
  }
  .saved-palette-content {
    flex-grow: 1;
  }
</style>
```

```
</head>
<body>
  <div class="container">
    <div class="input-group">
      <label for="baseColor">Базовий колір (HEX)</label>
      <div class="color-input-group">
        <input type="text" id="baseColor" placeholder="#RRGGBB"
value="#3498db">
        <input type="color" id="colorPicker" value="#3498db">
      </div>
    </div>
    <div class="input-group">
      <label for="harmonyType">Тип гармонії</label>
      <select id="harmonyType">
        <option value="complementary">Комплементарна</option>
        <option value="analogous">Аналогова</option>
        <option value="triadic">Тріадна</option>
        <option value="split-complementary">Розділена
комплементарна</option>
        <option value="monochromatic">Монохроматична</option>
        <option value="soft-harmony">М’яка гармонія</option>
        <option value="adaptive-contrast">Адаптивна
контрастна</option>
      </select>
    </div>
    <button onclick="generatePalette()">Згенерувати
палітру</button>
    <button onclick="applyToSelection()">Застосувати до
виділення</button>
    <button onclick="savePalette()">Зберегти палітру</button>
    <div id="palette" class="palette"></div>
    <div class="saved-palettes">
      <h3>Збережені палітри</h3>
      <div id="savedPalettes"></div>
    </div>
  </div>

  <script>
    // Синхронізація текстового поля і пікера кольору
    const baseColorInput = document.getElementById('baseColor');
    const colorPicker = document.getElementById('colorPicker');

    baseColorInput.addEventListener('input', () => {
      if (/^#[0-9A-Fa-f]{6}$/.test(baseColorInput.value)) {
        colorPicker.value = baseColorInput.value;
      }
    });

    colorPicker.addEventListener('input', () => {
      baseColorInput.value = colorPicker.value;
    });

    // Функції конвертації кольорів
```

```
    function hexToRGB(hex) {
      let r = 0, g = 0, b = 0;
      if (hex.length == 7) {
        r = parseInt(hex.slice(1, 3), 16);
        g = parseInt(hex.slice(3, 5), 16);
        b = parseInt(hex.slice(5, 7), 16);
      }
      return { r, g, b };
    }

    function hexToHSL(hex) {
      let r = 0, g = 0, b = 0;
      if (hex.length == 4) {
        r = parseInt(hex[1] + hex[1], 16);
        g = parseInt(hex[2] + hex[2], 16);
        b = parseInt(hex[3] + hex[3], 16);
      } else if (hex.length == 7) {
        r = parseInt(hex.slice(1, 3), 16);
        g = parseInt(hex.slice(3, 5), 16);
        b = parseInt(hex.slice(5, 7), 16);
      }
      r /= 255; g /= 255; b /= 255;

      let max = Math.max(r, g, b), min = Math.min(r, g, b);
      let h, s, l = (max + min) / 2;

      if (max == min) {
        h = s = 0;
      } else {
        let d = max - min;
        s = l > 0.5 ? d / (2 - max - min) : d / (max + min);
        switch (max) {
          case r: h = (g - b) / d + (g < b ? 6 : 0); break;
          case g: h = (b - r) / d + 2; break;
          case b: h = (r - g) / d + 4; break;
        }
        h /= 6;
      }
      return { h: h * 360, s: s * 100, l: l * 100 };
    }

    function hslToHex(h, s, l) {
      l /= 100;
      const a = s * Math.min(l, 1 - l) / 100;
      const f = n => {
        const k = (n + h / 30) % 12;
        const color = l - a * Math.max(Math.min(k - 3, 9 - k,
1), -1);
        return Math.round(255 * color).toString(16).padStart(2,
'0');
      };
      return `#${f(0)}${f(8)}${f(4)}`;
    }
```

```
    // Обчислення відносної яскравості (luminance) для перевірки
контрасту
    function calculateLuminance(hex) {
      const { r, g, b } = hexToRGB(hex);
      const a = [r, g, b].map(v => {
        v /= 255;
        return v <= 0.03928 ? v / 12.92 : Math.pow((v + 0.055) /
1.055, 2.4);
      });
      return 0.2126 * a[0] + 0.7152 * a[1] + 0.0722 * a[2];
    }

    // Перевірка контрасту з обмеженням ітерацій
    function adjustForContrast(color, baseColor, targetRatio =
3) {
      const baseLuminance = calculateLuminance(baseColor);
      let colorHSL = hexToHSL(color);
      let luminance = calculateLuminance(color);
      let ratio = (Math.max(luminance, baseLuminance) + 0.05) /
(Math.min(luminance, baseLuminance) + 0.05);

      let iterations = 0;
      const maxIterations = 10;

      console.log(`Initial contrast ratio: ${ratio}, color:
${color}, luminance: ${luminance}, baseLuminance:
${baseLuminance}`);

      while (ratio < targetRatio && colorHSL.l > 10 &&
colorHSL.l < 90 && iterations < maxIterations) {
        colorHSL.l = luminance > baseLuminance ? colorHSL.l - 5
: colorHSL.l + 5;
        color = hslToHex(colorHSL.h, colorHSL.s, colorHSL.l);
        luminance = calculateLuminance(color);
        ratio = (Math.max(luminance, baseLuminance) + 0.05) /
(Math.min(luminance, baseLuminance) + 0.05);
        iterations++;
        console.log(`Iteration ${iterations}: ratio: ${ratio},
l: ${colorHSL.l}, color: ${color}`);
      }

      if (iterations >= maxIterations) {
        console.warn(`Max iterations reached for ${color}, final
ratio: ${ratio}`);
      }

      return color;
    }

    // Обчислення гармоній кольорів
    function generateHarmony(baseHSL, type, baseHex) {
      const colors = [baseHSL];
```

```
const h = baseHSL.h, s = baseHSL.s, l = baseHSL.l;

switch (type) {
  case 'complementary':
    colors.push({ h: (h + 180) % 360, s, l });
    break;
  case 'analogous':
    colors.push({ h: (h + 30) % 360, s, l });
    colors.push({ h: (h - 30 + 360) % 360, s, l });
    break;
  case 'triadic':
    colors.push({ h: (h + 120) % 360, s, l });
    colors.push({ h: (h + 240) % 360, s, l });
    break;
  case 'split-complementary':
    colors.push({ h: (h + 150) % 360, s, l });
    colors.push({ h: (h + 210) % 360, s, l });
    break;
  case 'monochromatic':
    colors.push({ h, s, l: Math.min(l + 20, 100) });
    colors.push({ h, s, l: Math.max(l - 20, 0) });
    colors.push({ h, s: Math.min(s + 20, 100), l });
    colors.push({ h, s: Math.max(s - 20, 0), l });
    break;
  case 'soft-harmony':
    colors.push({
      h: (h + 30) % 360,
      s: Math.max(s - 15, 10),
      l: Math.min(l + 10, 90)
    });
    colors.push({
      h: (h - 30 + 360) % 360,
      s: Math.min(s + 10, 90),
      l: Math.max(l - 15, 10)
    });
    break;
  case 'adaptive-contrast':
    // Аналоговий м’який
    colors.push({
      h: (h + 40) % 360,
      s: s > 40 ? s - 20 : s,
      l: l < 70 ? l + 15 : l
    });
    // Контрастний
    colors.push({
      h: (h + 180) % 360,
      s: s < 60 ? s + 10 : s,
      l: l < 30 ? l + 20 : l > 50 ? l - 20 : l
    });
    // Аналоговий контрастний
    colors.push({
      h: (h - 40 + 360) % 360,
      s: s < 80 ? s + 15 : s,
```

```
            l: l > 20 ? l - 10 : l
          });
          // Нейтральний акцент
          colors.push({
            h: h,
            s: 20,
            l: l < 50 ? 80 : 20
          });
          break;
      }

      let result = colors.map(color => hslToHex(color.h,
color.s, color.l));
      if (type === 'adaptive-contrast') {
        // Перевірка контрасту лише для контрастного кольору
        result = [
          result[0], // Основний
          result[1], // Аналоговий м’який (без коригування)
          adjustForContrast(result[2], baseHex), // Контрастний
          result[3], // Аналоговий контрастний (без коригування)
          result[4] // Нейтральний (без коригування)
        ];
      }
      return result;
    }

    // Генерація палітри
    let currentPalette = [];
    function generatePalette() {
      const hex = baseColorInput.value;
      if (!/^#[0-9A-Fa-f]{6}$/.test(hex)) {
        parent.postMessage({ pluginMessage: { type: 'notify',
message: 'Будь ласка, введіть валідний HEX-код кольору' } },
'*');
        return;
      }

      const harmonyType =
document.getElementById('harmonyType').value;
      const baseHSL = hexToHSL(hex);
      currentPalette = generateHarmony(baseHSL, harmonyType,
hex);

      const paletteDiv = document.getElementById('palette');
      paletteDiv.innerHTML = currentPalette.map(color => `
        <div class="color-swatch" style="background: ${color}">
          <span>${color}</span>
        </div>
      `).join('');
    }

    // Застосування кольорів до виділення
    function applyToSelection() {
```

```
      if (!currentPalette.length) {
        parent.postMessage({ pluginMessage: { type: 'notify',
message: 'Спочатку згенеруйте палітру' } }, '*');
        return;
      }
      parent.postMessage({ pluginMessage: { type:
'applyPalette', palette: currentPalette } }, '*');
    }

    // Збереження палітри
    let savedPalettes = [];
    async function loadPalettes() {
      parent.postMessage({ pluginMessage: { type: 'loadPalettes'
} }, '*');
    }

    function savePalette() {
      if (!currentPalette.length) {
        parent.postMessage({ pluginMessage: { type: 'notify',
message: 'Спочатку згенеруйте палітру' } }, '*');
        return;
      }
      const paletteData = {
        id: Date.now(),
        colors: currentPalette,
        harmonyType:
document.getElementById('harmonyType').value,
        baseColor: baseColorInput.value
      };
      parent.postMessage({ pluginMessage: { type: 'savePalette',
palette: paletteData } }, '*');
    }

    // Видалення палітри
    function deletePalette(id, event) {
      event.stopPropagation();
      parent.postMessage({ pluginMessage: { type:
'deletePalette', id } }, '*');
    }

    function renderSavedPalettes() {
      const savedPalettesDiv =
document.getElementById('savedPalettes');
      savedPalettesDiv.innerHTML = savedPalettes.map(palette =>
`
        <div class="saved-palette">
          <div class="saved-palette-content"
onclick="applySavedPalette(${palette.id})">
            <div style="display: flex; gap: 4px; margin-bottom:
4px">
              ${palette.colors.map(color => `
                <div style="width: 20px; height: 20px;
background: ${color}; border-radius: 2px"></div>
```

```
            `).join('')}
          </div>
          <small>${palette.harmonyType} -
${palette.baseColor}</small>
        </div>
        <button class="delete-button"
onclick="deletePalette(${palette.id}, event)">Видалити</button>
      </div>
    `).join('');
  }

  function applySavedPalette(id) {
    const palette = savedPalettes.find(p => p.id === id);
    if (palette) {
      currentPalette = palette.colors;
      baseColorInput.value = palette.baseColor;
      colorPicker.value = palette.baseColor;
      document.getElementById('harmonyType').value =
palette.harmonyType;
      generatePalette();
    }
  }

  // Обробка повідомлень від code.js
  window.onmessage = (event) => {
    const message = event.data.pluginMessage;
    if (message.type === 'palettesLoaded') {
      savedPalettes = message.palettes || [];
      renderSavedPalettes();
    } else if (message.type === 'notify') {
      alert(message.message);
    }
  };

  // Ініціалізація
  loadPalettes();
</script>
</body>
</html>
```

Додаток Б

```
figma.showUI(__html__, { width: 400, height: 600 });

figma.ui.onmessage = (msg) => {
  if (msg.type === 'notify') {
    figma.notify(msg.message);
  } else if (msg.type === 'applyPalette') {
    const palette = msg.palette;
    const selection = figma.currentPage.selection;

    if (selection.length === 0) {
      figma.notify('Будь ласка, виберіть хоча б один об’єкт');
      return;
    }

    selection.forEach((node, index) => {
      if ('fills' in node) {
        const color = palette[index % palette.length];
        const r = parseInt(color.slice(1, 3), 16) / 255;
        const g = parseInt(color.slice(3, 5), 16) / 255;
        const b = parseInt(color.slice(5, 7), 16) / 255;

        node.fills = [{
          type: 'SOLID',
          color: { r, g, b }
        }];
      }
    });
    figma.notify('Палітру застосовано до виділення');
  } else if (msg.type === 'savePalette') {
    const paletteData = msg.palette;

figma.clientStorage.getAsync('savedPalettes').then(savedPalettes
=> {
      savedPalettes = savedPalettes || [];
      savedPalettes.push(paletteData);
      figma.clientStorage.setAsync('savedPalettes',
savedPalettes).then(() => {
        figma.notify('Палітру збережено');
        figma.ui.postMessage({ type: 'palettesLoaded', palettes:
savedPalettes });
      }).catch(err => {
        figma.notify('Помилка при збереженні палітри');
        console.error(err);
      });
    }).catch(err => {
      figma.notify('Помилка при отриманні палітр');
      console.error(err);
    });
  } else if (msg.type === 'loadPalettes') {
```

```
figma.clientStorage.getAsync('savedPalettes').then(savedPalettes
=> {
      figma.ui.postMessage({ type: 'palettesLoaded', palettes:
savedPalettes || [] });
    }).catch(err => {
      figma.notify('Помилка при завантаженні палітр');
      console.error(err);
    });
  } else if (msg.type === 'deletePalette') {
    const paletteId = msg.id;

figma.clientStorage.getAsync('savedPalettes').then(savedPalettes
=> {
      savedPalettes = savedPalettes || [];
      const updatedPalettes = savedPalettes.filter(palette =>
palette.id !== paletteId);
      figma.clientStorage.setAsync('savedPalettes',
updatedPalettes).then(() => {
        figma.notify('Палітру видалено');
        figma.ui.postMessage({ type: 'palettesLoaded', palettes:
updatedPalettes });
      }).catch(err => {
        figma.notify('Помилка при видаленні палітри');
        console.error(err);
      });
    }).catch(err => {
      figma.notify('Помилка при отриманні палітр');
      console.error(err);
    });
  } else if (msg.type === 'close') {
    figma.closePlugin();
  }
};
```

Додаток В

```
{
  "name": "Color Palette Generator",
  "id": "color-palette-generator",
  "api": "1.0.0",
  "main": "code.js",
  "ui": "ui.html",
  "capabilities": [],
  "enablePrivatePluginApi": false,
  "editorType": ["figma"]
}
```

Додаток Г

```
{
    "compilerOptions": {
      "target": "es6",
      "module": "commonjs",
      "outDir": "./dist",
      "rootDir": ".",
      "strict": false,
      "allowJs": true,
      "checkJs": false,
      "esModuleInterop": true,
      "skipLibCheck": true,
      "forceConsistentCasingInFileNames": true,
      "types": ["@figma/plugin-typings"]
    },
    "include": ["code.js"],
    "exclude": ["node_modules", "dist"]
  }
```

Додаток Д

```
{
  "name": "figma-color-palette-plugin",
  "version": "1.0.0",
  "description": "A Figma plugin for generating and saving color
palettes",
  "main": "code.js",
  "scripts": {
    "build": "tsc"
  },
  "dependencies": {},
  "devDependencies": {
    "@figma/plugin-typings": "^1.0.0",
    "typescript": "^4.0.0"
  }
}
```