**Zaawansowany Plan Testów Manualnych dla AiCodeBuddy**

**Cel testów:**

* Zweryfikowanie poprawności działania wszystkich funkcjonalności aplikacji przed integracją z QScintilla.
* Identyfikacja potencjalnych błędów i problemów w obecnej wersji aplikacji.
* Zapewnienie stabilności i jakości aplikacji przed wprowadzeniem dużych zmian.

**Zakres testów:**

* Funkcjonalności edytora kodu.
* Operacje na plikach.
* Integracja z Git.
* Uruchamianie i debugowanie kodu.
* Linter i analiza kodu.
* Ustawienia i personalizacja.
* Interfejs użytkownika i użyteczność.

**Szczegółowy Plan Testów:**

**1. Operacje na plikach**

**1.1. Nowy plik**

* **Kroki:**
  1. Uruchom aplikację.
  2. W menu "Plik" wybierz opcję "Nowy".
* **Oczekiwany rezultat:**
  1. Powinna otworzyć się nowa zakładka z pustym edytorem kodu.
  2. Zakładka powinna mieć domyślną nazwę, np. "Untitled".

**1.2. Otwieranie plików**

* **Kroki:**
  + **1.2.1.** Otwieranie pliku Python:
    1. W menu "Plik" wybierz "Otwórz".
    2. Wybierz plik z rozszerzeniem .py.
  + **1.2.2.** Otwieranie pliku nieobsługiwanego (np. .docx):
    1. W menu "Plik" wybierz "Otwórz".
    2. Wybierz plik .docx.
* **Oczekiwany rezultat:**
  + **1.2.1.:**
    1. Plik zostaje otwarty w nowej zakładce.
    2. Kod jest wyświetlany poprawnie.
    3. Podświetlanie składni działa zgodnie z językiem Python.
  + **1.2.2.:**
    1. Aplikacja wyświetla komunikat o błędzie lub ostrzeżenie, że plik nie jest obsługiwany.
    2. Plik nie jest otwierany w edytorze.

**1.3. Zapisywanie pliku**

* **Kroki:**
  1. Otwórz nowy plik lub dokonaj zmian w istniejącym.
  2. W menu "Plik" wybierz "Zapisz".
* **Oczekiwany rezultat:**
  1. Jeśli plik nie był wcześniej zapisany, powinno pojawić się okno dialogowe "Zapisz jako".
  2. Po wskazaniu ścieżki plik zostaje zapisany.
  3. Gwiazdka \* przy nazwie zakładki znika.
  4. Jeśli plik był wcześniej zapisany, zmiany są zapisywane bez okna dialogowego.

**1.4. Zapisz jako**

* **Kroki:**
  1. W menu "Plik" wybierz "Zapisz jako...".
  2. Wybierz nową ścieżkę i nazwę pliku.
* **Oczekiwany rezultat:**
  1. Plik zostaje zapisany pod nową nazwą.
  2. Zakładka aktualizuje nazwę pliku.
  3. Gwiazdka \* przy nazwie zakładki znika.

**1.5. Auto-zapis**

* **Kroki:**
  1. Upewnij się, że w ustawieniach opcja auto-zapisu jest włączona.
  2. Dokonaj zmian w otwartym pliku.
  3. Poczekaj 5 minut (lub ustaw krótszy interwał na potrzeby testu).
* **Oczekiwany rezultat:**
  1. Po upływie czasu auto-zapis powinien się wykonać.
  2. Zmiany powinny być zapisane w pliku na dysku.
  3. Gwiazdka \* może zniknąć lub pozostać w zależności od implementacji.

**2. Edytor kodu**

**2.1. Podświetlanie składni**

* **Kroki:**
  1. Otwórz plik w języku Python, JavaScript, C++, Go.
  2. Sprawdź, czy składnia jest podświetlana odpowiednio dla każdego języka.
* **Oczekiwany rezultat:**
  1. Podświetlanie składni działa poprawnie dla każdego języka.
  2. Kolory i style są zgodne z wybranym motywem.

**2.2. Inteligentne wcięcia**

* **Kroki:**
  1. Upewnij się, że opcja "Inteligentne wcięcie" jest włączona w ustawieniach.
  2. Wpisz kod, który wymaga wcięcia (np. po dwukropku w Pythonie).
  3. Naciśnij Enter.
* **Oczekiwany rezultat:**
  1. Nowa linia jest automatycznie wcięta zgodnie z poprzednią linią.
  2. Dodatkowe wcięcie jest dodawane po liniach zakończonych dwukropkiem w Pythonie.

**2.3. Autouzupełnianie**

* **Kroki:**
  1. Upewnij się, że autouzupełnianie jest włączone w ustawieniach.
  2. Wpisz fragment kodu, np. zaczynając pisać nazwę funkcji lub zmiennej.
  3. Obserwuj, czy pojawia się lista podpowiedzi.
  4. Wybierz podpowiedź z listy.
* **Oczekiwany rezultat:**
  1. Lista podpowiedzi pojawia się po wpisaniu odpowiedniej liczby znaków.
  2. Podpowiedzi są zgodne z kontekstem (np. dostępne metody, zmienne).
  3. Wybrana podpowiedź zostaje wstawiona do kodu.

**2.4. Szukanie tekstu**

* **Kroki:**
  1. Otwórz plik z kodem.
  2. Naciśnij Ctrl+F lub wybierz opcję "Znajdź".
  3. Wpisz szukany tekst i naciśnij Enter.
* **Oczekiwany rezultat:**
  1. Dialog wyszukiwania pojawia się.
  2. Wystąpienia szukanego tekstu są podświetlane w kodzie.
  3. Kursor przeskakuje do kolejnego wystąpienia po naciśnięciu Enter.

**2.5. Podświetlanie bieżącej linii**

* **Kroki:**
  1. Kliknij w dowolne miejsce w edytorze.
* **Oczekiwany rezultat:**
  1. Bieżąca linia jest podświetlona innym kolorem tła.

**2.6. Numeracja linii**

* **Kroki:**
  1. Sprawdź, czy po lewej stronie edytora widoczne są numery linii.
* **Oczekiwany rezultat:**
  1. Numery linii są wyświetlane poprawnie.
  2. Zmieniają się dynamicznie przy przewijaniu i dodawaniu/usuwaniu linii.

**3. Integracja z Git**

**3.1. Wybór repozytorium**

* **Kroki:**
  1. W menu "Git" wybierz "Wybierz Repozytorium Git".
  2. Wskaż folder z repozytorium Git.
* **Oczekiwany rezultat:**
  1. Repozytorium zostaje załadowane.
  2. Ścieżka jest zapamiętana i dostępna po ponownym uruchomieniu aplikacji.

**3.2. Commit zmian**

* **Kroki:**
  1. Dokonaj zmian w pliku znajdującym się w repozytorium.
  2. W menu "Git" wybierz "Commit".
  3. Wpisz wiadomość commit i zatwierdź.
* **Oczekiwany rezultat:**
  1. Zmiany zostają zacommitowane.
  2. Otrzymujesz potwierdzenie o powodzeniu operacji.

**3.3. Push zmian**

* **Kroki:**
  1. W menu "Git" wybierz "Push".
* **Oczekiwany rezultat:**
  1. Zmiany zostają wypchnięte do zdalnego repozytorium.
  2. Otrzymujesz potwierdzenie o powodzeniu operacji.

**3.4. Pull zmian**

* **Kroki:**
  1. W menu "Git" wybierz "Pull".
* **Oczekiwany rezultat:**
  1. Najnowsze zmiany zostają pobrane z zdalnego repozytorium.
  2. Konflikty (jeśli wystąpią) są odpowiednio obsługiwane lub zgłaszane.

**3.5. Zarządzanie branchami**

* **Kroki:**
  + **3.5.1. Tworzenie brancha:**
    1. W menu "Git" wybierz "Utwórz Branch".
    2. Wpisz nazwę nowego brancha.
  + **3.5.2. Przełączanie brancha:**
    1. W menu "Git" wybierz "Przełącz Branch".
    2. Wybierz istniejący branch z listy.
  + **3.5.3. Usuwanie brancha:**
    1. W menu "Git" wybierz "Usuń Branch".
    2. Wybierz branch do usunięcia.
* **Oczekiwany rezultat:**
  + **3.5.1.:** Nowy branch zostaje utworzony i aktywowany.
  + **3.5.2.:** Aktywny branch zostaje zmieniony.
  + **3.5.3.:** Wybrany branch zostaje usunięty (o ile nie jest aktualnie aktywny).

**4. Uruchamianie i debugowanie kodu**

**4.1. Uruchamianie skryptu**

* **Kroki:**
  1. Otwórz plik z kodem w obsługiwanym języku (np. Python).
  2. W menu "Uruchom" wybierz "Uruchom Skrypt" lub naciśnij F5.
* **Oczekiwany rezultat:**
  1. Kod zostaje uruchomiony.
  2. Wynik działania skryptu pojawia się w panelu wyjściowym.
  3. Błędy (jeśli wystąpią) są wyświetlane w panelu wyjściowym.

**4.2. Debugowanie skryptu**

* **Kroki:**
  1. Ustaw breakpointy w kodzie (jeśli jest taka funkcjonalność).
  2. W menu "Uruchom" wybierz "Debuguj Skrypt" lub naciśnij F6.
  3. Podłącz się do debuggera (np. za pomocą VSCode).
* **Oczekiwany rezultat:**
  1. Skrypt uruchamia się w trybie debugowania.
  2. Debugger zatrzymuje się na ustawionych breakpointach.
  3. Możliwość wykonywania krokowego, podglądu zmiennych itp.

**5. Linter i analiza kodu**

**5.1. Uruchomienie lintera**

* **Kroki:**
  1. W menu "Konfiguracja" wybierz "Sprawdź poprawność kodu" lub naciśnij Ctrl+L.
* **Oczekiwany rezultat:**
  1. Linter analizuje kod.
  2. Wyniki analizy (błędy, ostrzeżenia) są wyświetlane w panelu wyjściowym.
  3. Błędy są podświetlane w edytorze (np. podkreślenie faliste).

**5.2. Ustawienie konfiguracji Flake8**

* **Kroki:**
  1. Przygotuj plik konfiguracyjny dla Flake8.
  2. W menu "Konfiguracja" wybierz "Ustaw Konfigurację Flake8".
  3. Wskaż plik konfiguracyjny.
* **Oczekiwany rezultat:**
  1. Konfiguracja Flake8 zostaje załadowana.
  2. Kolejne uruchomienia lintera uwzględniają nową konfigurację.

**6. Ustawienia i personalizacja**

**6.1. Zmiana motywu**

* **Kroki:**
  1. W menu "Konfiguracja" wybierz "Ustawienia IDE".
  2. Wybierz inny motyw (np. z "Jasny" na "Ciemny").
  3. Zapisz zmiany.
* **Oczekiwany rezultat:**
  1. Wygląd aplikacji zmienia się zgodnie z wybranym motywem.
  2. Kolory w edytorze, panelu nawigacji i innych elementach są odpowiednio dostosowane.

**6.2. Zmiana rozmiaru czcionki**

* **Kroki:**
  1. W menu "Konfiguracja" wybierz "Ustawienia IDE".
  2. Zmień rozmiar czcionki na inny (np. z 12 na 16).
  3. Zapisz zmiany.
* **Oczekiwany rezultat:**
  1. Rozmiar czcionki w edytorze kodu zmienia się zgodnie z ustawieniem.

**6.3. Włączanie/wyłączanie opcji**

* **Kroki:**
  1. W menu "Konfiguracja" wybierz "Ustawienia IDE".
  2. Włącz lub wyłącz opcje takie jak "Automatyczne czyszczenie wklejania", "Inteligentne wcięcie", "Potwierdzenie przed usunięciem", "Tryb skupienia".
  3. Zapisz zmiany.
* **Oczekiwany rezultat:**
  1. Działanie aplikacji dostosowuje się do wybranych ustawień.
  2. Na przykład, w trybie skupienia panele boczne są ukrywane.

**7. Panel nawigacji kodu**

**7.1. Wyświetlanie funkcji i klas**

* **Kroki:**
  1. Otwórz plik z kodem zawierającym funkcje i klasy.
  2. Sprawdź panel nawigacji kodu.
* **Oczekiwany rezultat:**
  1. Panel nawigacji wyświetla listę funkcji i klas.
  2. Kliknięcie na element przenosi kursor do odpowiedniego miejsca w kodzie.

**7.2. Aktualizacja panelu nawigacji**

* **Kroki:**
  1. Dodaj nową funkcję lub klasę w kodzie.
  2. Sprawdź, czy panel nawigacji został zaktualizowany.
* **Oczekiwany rezultat:**
  1. Nowo dodane elementy pojawiają się w panelu nawigacji.
  2. Poprawnie działają linki do nich.

**8. Interfejs użytkownika i użyteczność**

**8.1. Reakcja na błędne operacje**

* **Kroki:**
  1. Spróbuj wykonać operację bez wymaganych kroków, np. "Commit" bez wybranego repozytorium Git.
* **Oczekiwany rezultat:**
  1. Aplikacja wyświetla odpowiedni komunikat o błędzie.
  2. Użytkownik jest informowany o konieczności wykonania wcześniejszych kroków.

**8.2. Sprawdzanie stabilności**

* **Kroki:**
  1. Wykonuj różne operacje w szybkim tempie (np. otwieranie/zamykanie zakładek, zmiana ustawień).
  2. Obserwuj, czy aplikacja zachowuje stabilność.
* **Oczekiwany rezultat:**
  1. Aplikacja działa płynnie, bez zawieszania się czy awarii.

**8.3. Zachowanie po ponownym uruchomieniu**

* **Kroki:**
  1. Wprowadź zmiany w ustawieniach (np. zmień motyw).
  2. Zapisz pliki.
  3. Zamknij aplikację i uruchom ją ponownie.
* **Oczekiwany rezultat:**
  1. Ustawienia są zachowane.
  2. Ostatnio wybrane repozytorium Git jest pamiętane.
  3. Ewentualnie ostatnio otwarte pliki są ponownie otwarte (jeśli taka funkcjonalność jest zaimplementowana).

**9. Linter i błędy składniowe**

**9.1. Wprowadzanie błędów w kodzie**

* **Kroki:**
  1. W kodzie wprowadź celowe błędy składniowe lub stylistyczne (np. brak dwukropka po if w Pythonie).
  2. Uruchom lintera.
* **Oczekiwany rezultat:**
  1. Linter wykrywa błędy i wyświetla je w panelu wyjściowym.
  2. Błędne linie są podkreślone w edytorze.

**9.2. Poprawianie błędów**

* **Kroki:**
  1. Popraw błędy wykryte przez lintera.
  2. Uruchom lintera ponownie.
* **Oczekiwany rezultat:**
  1. Linter nie zgłasza już tych błędów.
  2. Podkreślenia błędów znikają z edytora.

**Uwagi dodatkowe:**

* **Testy funkcjonalności niezaimplementowanych lub z problemami:** Jeśli wiesz, że pewne funkcje mają problemy (np. autouzupełnianie), możesz je uwzględnić w testach, aby dokładnie zrozumieć ich zachowanie przed integracją z QScintilla.
* **Dokumentowanie wyników:** Podczas testowania zapisuj wyniki testów, zwłaszcza jeśli wystąpią nieoczekiwane zachowania. To pomoże w późniejszej analizie i naprawie błędów.
* **Testowanie na różnych plikach:** Używaj różnych plików kodu, o różnej wielkości i złożoności, aby upewnić się, że aplikacja działa poprawnie w różnych scenariuszach.
* **Testowanie wydajności:** Jeśli to możliwe, obserwuj wykorzystanie zasobów (CPU, pamięć) podczas pracy z aplikacją, zwłaszcza przy dużych plikach lub intensywnych operacjach.